

**Silvio Moreto, Matt Lambert,
Benjamin Jakobus, Jason Marah**

Bootstrap 4 - Responsive Web Design

Learning Path

A step-by-step practical course enabling you to nail
Bootstrap and make your web designs responsive



Packt

Bootstrap 4 – Responsive Web Design

A step-by-step practical course enabling you to nail
Bootstrap and make your web designs responsive

A course in three modules



BIRMINGHAM - MUMBAI

Bootstrap 4 – Responsive Web Design

Copyright © 2016 Packt Publishing

All rights reserved. No part of this course 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 course to ensure the accuracy of the information presented. However, the information contained in this course 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 course.

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

Published on: September,2016

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

ISBN 978-1-78839-731-5

www.packtpub.com

Credits

Authors

Silvio Moreto

Matt Lambert

Benjamin Jakobus

Jason Marah

Content Development Editor

Roshan Kumar

Graphics

Jason Monterio

Reviewers

Paula Barcante

Sherwin Robles

Marija Zaric

Production Coordinator

Melwyn D'sa

Preface

Bootstrap has become a very popular tool in front-end projects over the years. The framework's ease-of-use along with its cross-browser compatibility, support for mobile user interfaces, and responsive web design capabilities, make it an essential building block for any modern web application. The course will enable you to rapidly build elegant, powerful, and responsive interfaces for professional-level web pages using Bootstrap 4.

At the beginning of the course, take a deep dive into the Bootstrap frontend framework with the help of examples that will illustrate the usage of each element and component in a proper way. By seeing examples, you will get a better understanding of what is happening and where you want to reach. With this guide, along the examples, you will get confident with the framework and develop some very common examples using Bootstrap. These are a landing page, a web application, and a dashboard, which is desired by 10 out 10 web developers.

Second module is comprehensive tutorial, which will teach you everything that you need to know to start building websites with Bootstrap 4 in a practical way. You'll learn about build tools such as Node, Grunt, and many others. You'll also discover the principles of mobile-first design in order to ensure your pages can fit any screen size and meet the responsive requirements. This guide will make sure you're geared up and ready to build amazingly beautiful and responsive websites in a jiffy.

By third module, get to the grips with Bootstrap's key features and quickly discover the various ways in which Bootstrap can help you develop web-interfaces. You will know how to both extend the framework, integrate it with third-party components and frameworks, as well as optimize and automate your Bootstrapped builds.

What this learning path covers

Module 1, Bootstrap 4 By Example, This module will take a deep dive into the Bootstrap frontend framework with the help of examples that will illustrate the usage of each element and component in a proper way. By seeing examples, you will get a better understanding of what is happening and where you want to reach. Along the examples, you will be able to nail the framework and develop some very common examples using Bootstrap. These are a landing page, a web application, and a dashboard, which is desired by 10 out 10 web developers. You will face these kind of page countless number of times during your life as a developer, and you will do that using Bootstrap at its finest, including component customizations, animations, event handling, and external library integration.

Module 2, Learning Bootstrap 4, This module will help you to design elegant, powerful, and responsive interfaces for professional-level web pages. This module will introduce a wide range of new features that make frontend web design even more simple and exciting. You'll get a feel of build tools such as Node, Grunt, and more to start building your project. You'll discover the principles of mobile-first design to ensure your pages can adapt to fit any screen size and meet the responsive requirements of the modern age. You'll get to play with Bootstrap's grid system and base CSS to ensure your designs are robust and beautiful, and that your development process is speedy and efficient. Then, you'll find out how you can extend your current build with some cool JavaScript plugins, and throw in some Sass to spice things up and customize your themes. If you've tinkered with Bootstrap before and are planning on migrating to the latest version, we'll give you just the right tricks to get you there. This module will make sure you're geared up and ready to build amazingly beautiful and responsive websites in a jiffy.

Module 3, Mastering Bootstrap 4, The motivation behind this module is to provide a comprehensive, step-by-step guide for developers that wish to build a complete, production-ready, website using Bootstrap 4. Once you turn the final pages of this module, you should be mastering the framework's ins and outs, and building highly customizable and optimized web interfaces. You will know how to both extend the framework, integrate it with third-party components and frameworks, as well as optimize and automate your Bootstrapped builds.

What you need for this learning path

Module 1:

To follow this module's developments, you will need a web browser on your computer, preferably Google Chrome, because it will be used in some examples. But other browsers can work as well. Also, you will need some basic knowledge in HTML, CSS, and JavaScript beforehand. Despite the fact that we will initially talk in a slow pace about those technologies, it will be good for you to know some basic concepts about them. Another plus is knowledge of the jQuery library, which is a dependency of Bootstrap. We will actually use jQuery in Chapter 7 of this module, *Of Course, You Can Build a Web App!*, and they will be very simple examples. So just keep in mind to train some jQuery skills.

Module 2:

To get started using Bootstrap 4 there are a few tools we would recommend installing on your computer. First of all you'll need a text editor like Sublime Text or Notepad. Secondly you'll need a command line tool. If you're on a MAC you can use Terminal which is included with OSX. If you're on Windows we would recommend downloading Cygwin. That's all you need to get started with the module. In Chapter 2, we will cover the installation of a few other tools that we'll be using in our project like: Node.js, NPM, Grunt.js, Jekyll, Harp.js and Sass by using Bootstrap Build Tools. If you already have those tools installed great! If not, don't worry we'll go through it step by step later in the module.

Module 3:

This module targets intermediate-level frontend web developers. The module is not intended to be an introduction to web development. As such, we assume that readers have a firm grasp of the basic concepts behind web development, essential HTML, JavaScript, and CSS skills, as well as practical experience of applying this knowledge. Furthermore, the reader should have an understanding of jQuery. Elementary knowledge about AngularJS, build tools, and React are desirable for the module's final two chapters. However, unfamiliarity with the two frameworks will not stop readers from completing and understanding the examples

Who this learning path is for

If you're a web developer with little or no knowledge of Bootstrap, then this course is for you. The course offers support for the version 4 of Bootstrap, however, it will offer support for version 3 as well. So, you will be ready for whatever comes your way. Prior knowledge of HTML, CSS, and JavaScript is expected.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this course – what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail feedback@packtpub.com, and mention the course's title in 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 course, see our author guide at www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt course, 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 this course from your account at <http://www.packtpub.com>. If you purchased this course elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

You can download the code files by following these steps:

1. Log in or register to our website using your e-mail address and password.
2. Hover the mouse pointer on the **SUPPORT** tab at the top.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the course in the **Search** box.
5. Select the course for which you're looking to download the code files.
6. Choose from the drop-down menu where you purchased this course from.
7. Click on **Code Download**.

You can also download the code files by clicking on the **Code Files** button on the course's webpage at the Packt Publishing website. This page can be accessed by entering the course's name in the **Search** box. Please note that you need to be logged in to your Packt account.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR / 7-Zip for Windows
- Zipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

The code bundle for the course is also hosted on GitHub at <https://github.com/PacktPublishing/Bootstrap-4-Responsive-Web-Design>. We also have other code bundles from our rich catalog of books and videos available at <https://github.com/PacktPublishing/>. Check them out!

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 courses—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this course. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your course, 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 to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to <https://www.packtpub.com/books/content/support> and enter the name of the course in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted 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

If you have a problem with any aspect of this course, you can contact us at questions@packtpub.com, and we will do our best to address the problem.

Module 1: Bootstrap 4 By Example

Chapter 1: Getting Started	3
Getting Bootstrap	4
Setting up the framework	5
Folder structure	6
Warming up the sample example	6
Bootstrap required tags	8
Building our first Bootstrap example	10
The container tag	11
Optionally using the CDN setup	14
Community activity	15
Tools	16
Bootstrap and web applications	16
Browser compatibility	17
Summary	18
Chapter 2: Creating a Solid Scaffolding	19
Understanding the grid system	19
Building our scaffolding	20
Setting things up	21
Offset columns	23
Completing the grid rows	24
Nesting rows	24
Finishing the grid	26
Fluid container	28
We need some style!	28
There are headings everywhere	30
Playing with buttons	31

More typography and code tags	32
Manipulating tables	37
Styling the buttons	40
Like a boss!	41
Final thoughts	42
Box-sizing	42
Quick floats	43
Clearfix	43
Font definitions for typography	44
Summary	44
Chapter 3: Yes, You Should Go Mobile First	47
Making it greater	47
Bootstrap and the mobile-first design	49
How to debug different viewports at the browser	49
Cleaning up the mess	52
Creating the landing page for different devices	53
Mobile and extra small devices	54
Tablets and small devices	59
Desktop and large devices	61
Summary	62
Chapter 4: Applying the Bootstrap Style	65
Changing our grid layout	65
Starting over the grid system	67
The header	67
The introduction header	68
The about section	71
The features section	73
The price table section	75
The footer	78
Forming the forms	81
Newsletter form	81
Contact form	83
The sign-in form	87
Images	89
Helpers	91
Floating and centering blocks	91
Context colors	91
Spacing	92
Responsive embeds	93
Summary	94

Chapter 5: Making It Fancy	95
Using Bootstrap icons	95
Paying attention to your navigation	99
Until the navigation collapse	100
Using different attachments	102
Coloring the bar	103
Dropping it down	103
Customizing buttons dropdown	106
Making an input grouping	107
Getting ready for flexbox!	109
Understanding flexbox	109
Playing with Bootstrap and flexbox	111
Summary	112
Chapter 6: Can You Build a Web App?	113
Understanding web applications	113
Creating the code structure	114
Adding the navigation	114
Adding the search input	117
Time for the menu options!	118
The option at the thumbnail	118
Adding the Tweet button	119
Customizing the navigation bar	120
Setting up the custom theme	120
Fixing the list navigation bar pseudo-classes	121
You deserve a badge!	122
Fixing some issues with the navigation bar	123
Do a grid again	127
Playing the cards	127
Learning cards in Bootstrap 4	128
Creating your own cards	129
Adding Cards to our web application	130
Another card using thumbnails	133
Implementing the main content	135
Making your feed	136
Doing some pagination	142
Creating breadcrumbs	143
Finishing with the right-hand-side content	144
Summary	149

Chapter 7: Of Course, You Can Build a Web App!	151
Alerts in our web app	152
Dismissing alerts	153
Customizing alerts	153
Waiting for the progress bar	155
Progress bar options	156
Animating the progress bar	157
Creating a settings page	158
Pills of stack	159
Tabs in the middle	163
Adding the tab content	165
Using the Bootstrap tabs plugin	165
Creating content in the user info tab	166
The stats column	169
Labels and badges	171
Summary	173
Chapter 8: Working with JavaScript	175
Understanding JavaScript plugins	175
The library dependencies	176
Data attributes	176
Bootstrap JavaScript events	177
Awesome Bootstrap modals	177
Modal general and content	179
The modal header	179
The modal body	180
The modal footer	180
Creating our custom modal	181
A tool for your tip	183
Pop it all over	186
Popover events	189
Making the menu affix	191
Finishing the web app	193
Summary	198
Chapter 9: Entering in the Advanced Mode	199
The master plan	200
The last navigation bar with flexbox	201
The navigation search	205
The menu needs navigation	206
Checking the profile	211

Filling the main fluid content	212
From the side stacked menu	213
I heard that the left menu is great!	214
Learning the collapse plugin	216
Using some advanced CSS	220
Filling the main content	221
Rounding the charts	223
Creating a quick statistical card	226
Getting a spider chart	229
Overhead loading	232
Fixing the toggle button for mobile	234
Summary	235
Chapter 10: Bringing Components to Life	237
Creating the main cards	237
The other card using Bootstrap components	241
Creating our last plot	244
Fixing the mobile viewport	246
Fixing the navigation menu	250
The notification list needs some style	253
Adding the missing left menu	254
Aligning the round charts	256
Learning more advanced plugins	258
Using the Bootstrap carousel	258
Customizing carousel items	260
Creating slide indicators	260
Adding navigation controls	262
Other methods and options for the carousel	263
The Bootstrap spy	264
Summary	269
Chapter 11: Making It Your Taste	271
Customizing a Bootstrap component	271
The taste of your button	272
Using button toggle	273
The checkbox toggle buttons	274
The button as a radio button	275
Doing the JavaScript customization	276
Working with plugin customization	276
The additional Bootstrap plugins	282
Creating our Bootstrap plugin	283
Creating the plugin scaffold	284

Defining the plugin methods	289
The initialize method and plugin verifications	289
Adding the Bootstrap template	290
Creating the original template	292
The slide deck	293
The carousel indicators	294
The carousel controls	295
Initializing the original plugin	295
Making the plugin alive	296
Creating additional plugin methods	298
Summary	300

Module 1

Bootstrap 4 By Example

Learn responsive web development with Bootstrap 4's front end framework

1

Getting Started

With the advent and increase in popularity of the mobile web, developers have had to adapt themselves to handling new challenges, such as different layouts in different resolutions, the new user experience paradigm, and optimization for low-bandwidth connections. While facing this, there were also a lot of old problems related to browser compatibility and lack of patterns in the community.

This was the outline scenario when the Bootstrap framework arrived. Developed by Twitter, the main goal of Bootstrap is to provide a web frontend framework for responsive developing with cross-browser compatibility. It is awesome! Developers fell in love with it and started to adopt it right away.

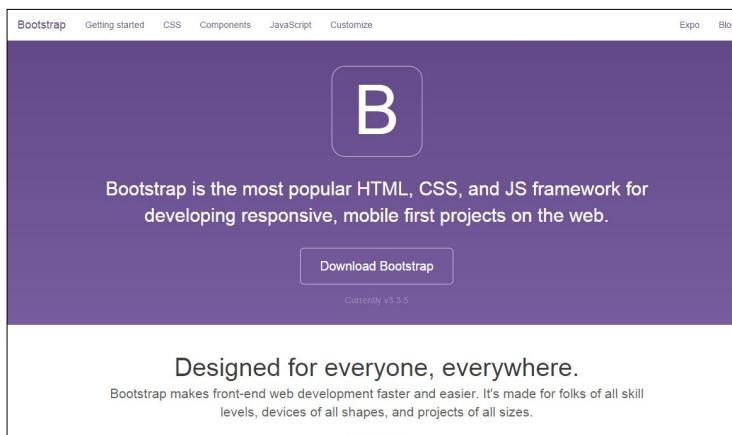
Therefore, to cover this book's objective of presenting the Bootstrap framework to build responsive, mobile-first websites faster than ever before, we must get started by setting up our work environment in the best recommended way. Thus, the topics that we will cover in this chapter are:

- Getting Bootstrap
- Setting up Bootstrap in a web page
- Building the first Bootstrap example
- The container element tag
- Support sources
- Framework compatibility

Getting Bootstrap

There are several versions of the framework, but in this book, we will provide support for the latest Bootstrap 3 version (which is v3.3.5), along with the newest version 4 (which is 4.0.0-alpha). When a feature or component is differently supported by one of these versions, we will point it out properly.

First of all, access the official website at <http://getbootstrap.com/> and click on the **Download Bootstrap** button, as shown in the following screenshot:



Downloading the example code

You can download the example code files for this book 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.

You can download the code files by following these steps:



- Log in or register to our website using your e-mail address and password.
- Hover the mouse pointer on the **SUPPORT** tab at the top.
- Click on **Code Downloads & Errata**.
- Enter the name of the book in the **Search** box.
- Select the book for which you're looking to download the code files.
- Choose from the drop-down menu where you purchased this book from.
- Click on **Code Download**.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR / 7-Zip for Windows
- Zipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

Then you will be redirected to another page that contains these buttons:

- **Download Bootstrap:** This is the release with the files already compiled.
- **Download source:** Use this if you want to get the source for customization. This requires knowledge of the Less language.
- **Download Sass:** Here, you can get the source code in the Sass language.

Click on the **Download Bootstrap** button to get the framework, since we will cover the full framework using, not Sass, but just HTML, CSS, and JavaScript. After the download, extract the files and you will see that the framework is organized in folders.

Other versions and releases

Check out the official repository at <https://github.com/twbs/bootstrap/> to pick up other versions and see the new releases under development. You will also be able to see other features and community activity.

If you want to go hands-on straightforward with version 4, go to <http://v4-alpha.getbootstrap.com/> and download it, or enter the GitHub repository and select the corresponding branch of version 4.

After you've extracted the files, you will see some folders. The first one, in alphabetical order, is `css`. Here, you will find the main CSS file (named `bootstrap.css`), other files related to the minified version, and a `bootstrap-theme.css` file, which is a simple theme of using the Bootstrap components.

There is also a `fonts` folder; it contains the files used for the icon components that we will see in future chapters. Finally, there is a folder named `js`, where we can find the `bootstrap.js` file, the minified version, and the specification for `npm`.

What is the npm file?

The `npm` is the most famous package manager for JavaScript. It is set as the default package manager in the Node.js environment.

Setting up the framework

Now that we have downloaded the framework and covered its basic file architecture, we will advance to setting up Bootstrap on a web page.

Folder structure

First, let's explicit the folder structure that we will be using in this book. In a folder that we will call `main_folder`, we extract the Bootstrap contents and create a file called `hello_world.html` at the same level. Inside the Bootstrap contents will be some folders for fonts, CSS, and JavaScript. The final layout should be like this:

```
main_folder
- hello_world.html
- css
    - bootstrap.css
- fonts
    - glyphicons-halflings-regular.eot
    - glyphicons-halflings-regular.svg
    - glyphicons-halflings-regular.ttf
    - glyphicons-halflings-regular.woff
    - glyphicons-halflings-regular.woff2
- js
    - bootstrap.js
```

Warming up the sample example

Now, we will add the recommended setup of the Bootstrap framework to the `hello_world.html` file. Open it in your preferred code editor and add the outline HTML code, like this:

```
<!DOCTYPE html>
<html>
<head>
    <title>Hello World!</title>
</head>
<body>
    Hello World
</body>
</html>
```

Next, add the code for loading `css` inside the `head` tag:

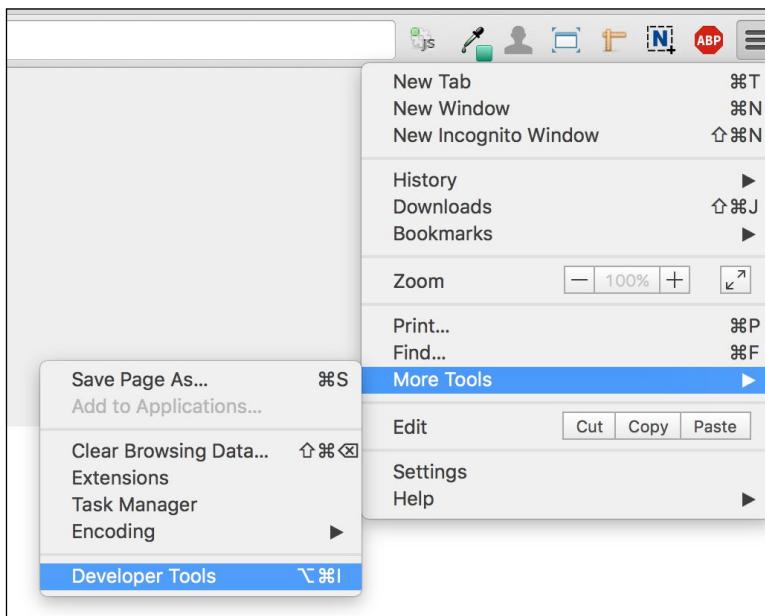
```
<!DOCTYPE html>
<html>
<head>
    <title>Hello World!</title>
    <link rel="stylesheet" href="css/bootstrap.css">
```

```
</head>
<body>
    Hello World
</body>
</html>
```

And at the end of the body tag, load the JavaScript file:

```
<!DOCTYPE html>
<html>
<head>
    <title>Hello World!</title>
    <link rel="stylesheet" href="css/bootstrap.css">
</head>
<body>
    Hello World
    <script src="js/bootstrap.js"></script>
</body>
</html>
```

Open the `hello_world.html` file in a browser (we will use Google Chrome in this book) and open the JavaScript console. In Chrome, it can be found at *Options* button (the hamburger button on right upper corner. Go to **More Tools | Developer Tools**, just as shown in the following screenshot, and click on **Console** in the opened window. You will see a message saying **Bootstrap's JavaScript requires jQuery**:



jQuery is a cross-platform JavaScript library, and it is the only third-party requirement for Bootstrap. To get it, we recommend the download from the official website and the latest version (<https://jquery.com/download/>). Bootstrap requires version 1.9 or higher.



Just use versions above 2.x if you do not want to add support for Internet Explorer 6, 7, and 8. In this book, we will use version 1.11.3.



Copy the jQuery file inside the `js` folder, and load it in the HTML code at the end of the body tag but before the `bootstrap.js` loads, like this:

```
<script src="js/jquery-1.11.3.js"></script>
<script src="js/bootstrap.js"></script>
```

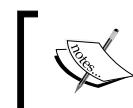
Bootstrap required tags

Bootstrap has three tags that must be at the beginning of the `<head>` tag. These tags are used for text encoding and improved visualization on mobile devices:

```
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1">
```

The `viewport` tag is related to the mobile-first philosophy. By adding it, you ensure proper rendering in mobile devices and touch zooming.

You can also disable the zoom functionality by appending `user-scalable=no` in the `content` key. With this option, users will only be able to scroll on the web page, resulting in a feel of using a native mobile application.



If you are going to use this tag, you must be sure that users need not use the zoom feature and it will create a good user experience. Therefore, use it with caution.



Also, if you want to add support for older versions of the Internet Explorer (IE) browser (older than version 9), you must add some libraries to have fallback compatibility for the HTML5 and CSS3 elements. We will add them via CDN, which is the Bootstrap recommendation. So, add these lines at the end of the `<head>` tag:

```
<!--[if lt IE 9]>
<script
src="https://oss.maxcdn.com/html5shiv/3.7.2/html5shiv.min.js"></
script>
```

```
<script  
src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js">  
</script> <! [endif] -->
```



Do you know what CDN is?

CDN, the abbreviation of **Content Delivery Network**, is a term used to describe a network of computers that are connected in order to deliver some content. A CDN should provide high availability and performance.

At this point, the file should be like this:

```
<!DOCTYPE html>  
<html>  
  <head>  
    <meta charset="utf-8">  
    <meta http-equiv="X-UA-Compatible" content="IE=edge">  
    <meta name="viewport" content="width=device-width,  
initial-scale=1">  
    <title>Hello World!</title>  
  
    <link rel="stylesheet" href="css/bootstrap.css">  
  
    <!--[if lt IE 9 ]>  
      <script  
src="https://oss.maxcdn.com/html5shiv/3.7.2/html5shiv.min.js">  
        </script>  
      <script  
src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js">  
        </script>  
    <! [endif] -->  
  </head>  
  <body>  
    Hello World!  
  
    <script src="js/jquery-1.11.3.js"></script>  
    <script src="js/bootstrap.js"></script>  
  </body>  
</html>
```

This is our base page example! Keep it for the purpose of coding every example of this book and for any other web page that you will develop.

We would like to point out that Bootstrap requires the doctype HTML5 style before the `<html>` tag:

```
<!DOCTYPE html>
<html>
    ... <!--rest of the HTML code -->
</html>
```

Building our first Bootstrap example

Now we are all set for the framework. Replace the `Hello World!` line in the `body` tag with this:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-
scale=1">
        <title>Hello World!</title>

        <link rel="stylesheet" href="css/bootstrap.css">

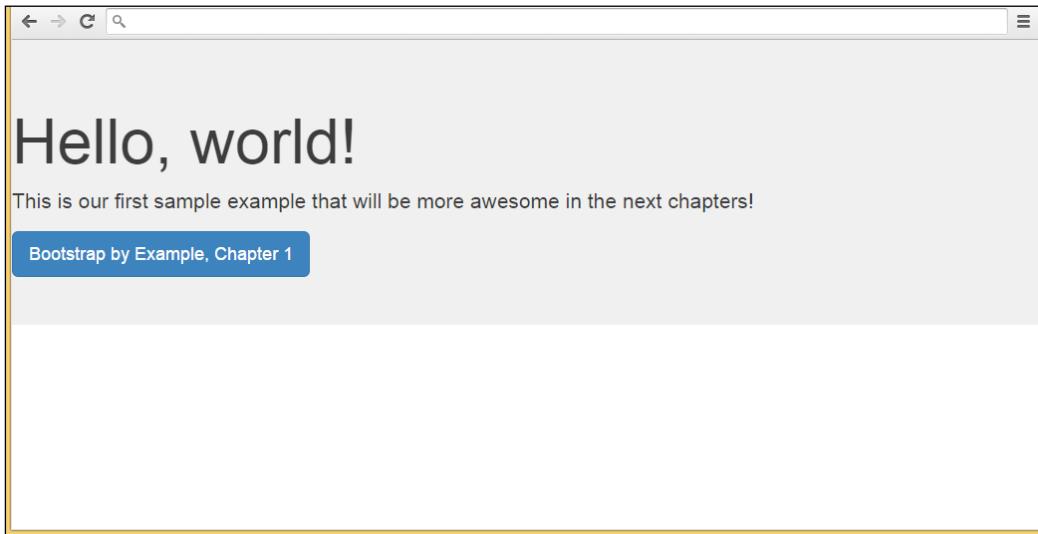
        <!--[if lt IE 9]>
            <script
src="https://oss.maxcdn.com/html5shiv/3.7.2/html5shiv.min.js">
        </script>
            <script
src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js">
        </script>
        <!--[endif]-->
    </head>
    <body>

        <div class="jumbotron">
            <h1>Hello, world!</h1>
            <p>This is our first sample example that will be more
awesome in the next chapters!</p>
            <a class="btn btn-primary btn-lg" href="#" role="button">
                Bootstrap by Example, Chapter 1
            </a>
        </div>
    </body>
</html>
```

```
</a>
</div>

<script src="js/jquery-1.11.3.js"></script>
<script src="js/bootstrap.js"></script>
</body>
</html>
```

Open the `hello_world.html` file in the browser, and it must appear like what is shown in the following screenshot:



Congratulations! You have created your first Bootstrap web page. It is simple but very important to understand the details of how to set the framework correctly to keep the recommendation pattern.

Furthermore, we added some components in this example that will be explained in future chapters, but you can start becoming familiar with the CSS classes used and the placement of the elements.

The container tag

You may notice that in our example, the page content is too close to the left-hand side and without a margin/padding. This is because Bootstrap has a required element called `container` that we have not added in the example.

The container tag must be placed to wrap the site content and nest the grid system (we will present the grid system, called scaffolding, in the next chapter). There are two options for using the container element.

The first one is for creating a web page responsive with a fixed-width container. This one will add responsive margins depending on the device viewport:

```
<div class="container">  
    ...  
</div>
```

In case you want a full-width container, covering the entire width of the viewport, use container-fluid:

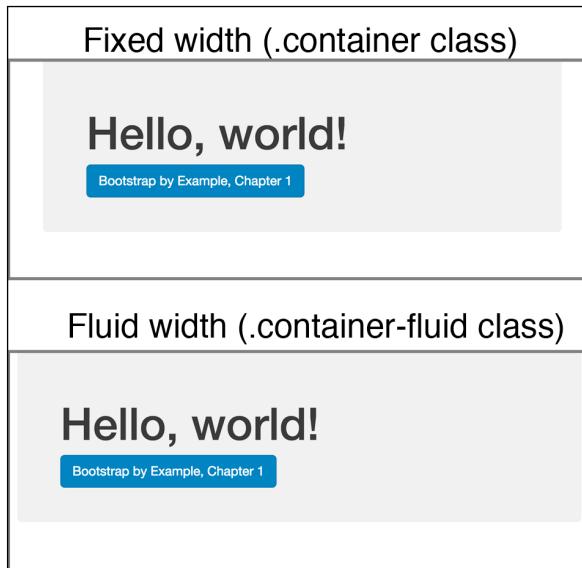
```
<div class="container-fluid">  
    ...  
</div>
```

In our example, we will create a fixed-width responsive website. So, our code will be like this:

```
<!DOCTYPE html>  
<html>  
    <head>  
        <meta charset="utf-8">  
        <meta http-equiv="X-UA-Compatible" content="IE=edge">  
        <meta name="viewport" content="width=device-width, initial-  
scale=1">  
        <title>Hello World!</title>  
  
        <link rel="stylesheet" href="css/bootstrap.css">  
  
        <!-- [if lt IE 9]>  
            <script src="https://oss.maxcdn.com/html5shiv/3.7.2/html5shiv.  
min.js">  
        </script>  
        <script  
src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js">  
    </script>  
        <![endif]-->  
    </head>  
    <body>  
        <div class="container">  
            <div class="jumbotron">  
                <h1>Hello, world!</h1>
```

```
<p>This is our first sample example that will be more  
awesome in the next chapters!</p>  
<a class="btn btn-primary btn-lg" href="#"  
role="button">  
    Bootstrap by Example, Chapter 1  
</a>  
</div>  
</div>  
  
<script src="js/jquery-1.11.3.js"></script>  
<script src="js/bootstrap.js"></script>  
</body>  
</html>
```

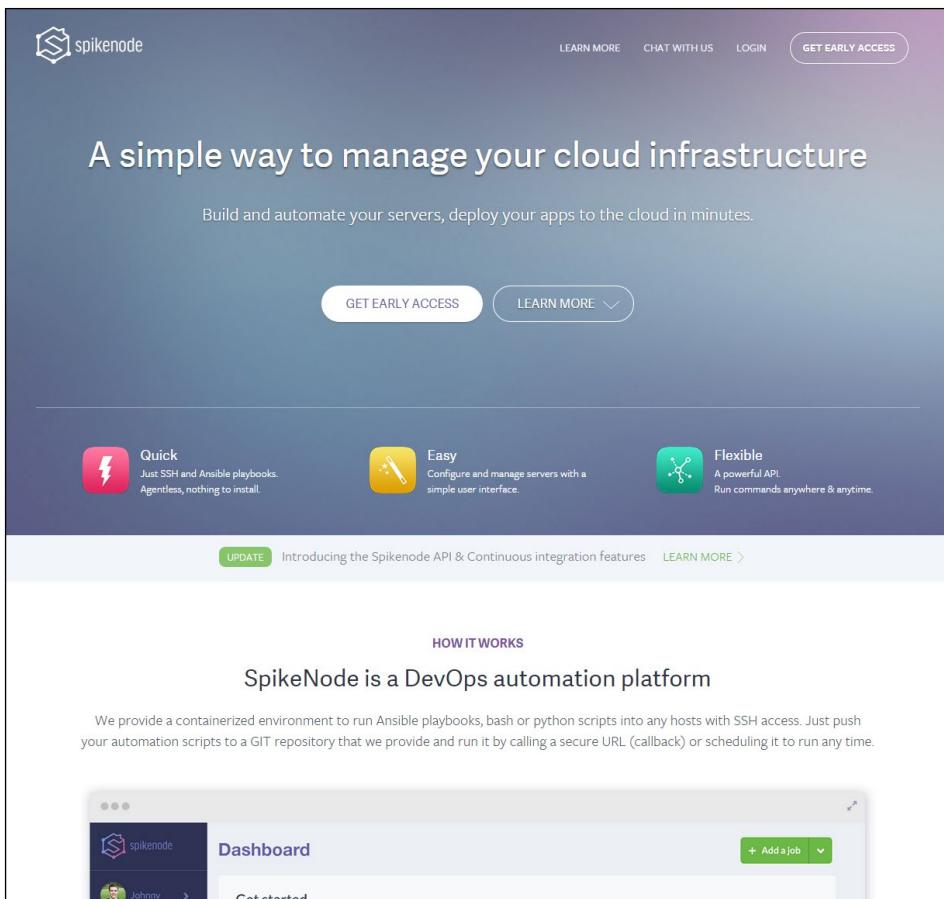
The next screenshot shows what our example looks like with the addition of the container class. I recommend for practicing and complete understanding, that you change the container class to `.container-fluid` and see what happens. Change your viewport by resizing your browser window and see how Bootstrap adapts your page visualization:



The preceding image shows the differences between using `.container` and `.container-fluid`. See the differences of the margins in the sides.

Soon during this book, you will be able to create more complex and beautiful websites, using more advanced Bootstrap components such as the show case shown in the following screenshot, which is an example of a landing page.

Do not worry. We will start at a slow pace to reveal the basics of Bootstrap and how to use it properly on our web page. The following example is our first goal when we develop a landing page example. Just keep in mind that we will always use the same basis presented in this chapter.



Optionally using the CDN setup

Bootstrap also offers a setup using CDN to load the framework. It's much easier to set up but comes with some regards. Instead of the `<link>` that we created to load the CSS, we must load it from CDN using this:

```
<link rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/
      bootstrap.min.css">
```

And to load the JavaScript file, replace the JavaScript `<script>` tag with the following line:

```
<script  
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.  
js"></script>
```

There is some discussion on whether or not to use CDN. We will not touch upon this point, but the main pro is having the content provided faster with high availability. The main con is that you cannot have direct control over what is in the content provided, having unreliable imported code.

The decision of whether or not to use CDN depends on the case. You should consider the different arguments and choose an option that best fits your web page. There is no right or wrong, just different points of view.

Community activity

The Bootstrap framework is discussed in several places across the Internet. It is important to have an engaged community that keeps evolving the framework and supporting it. You can have support and acquire more knowledge by going to some other resources, such as the following:

- The Bootstrap official GitHub repository. Here (<https://github.com/twbs/bootstrap>), you can find the road map development and the newest releases, report issues, and solve them by making a pull request.
- The Bootstrap official documentation (<http://getbootstrap.com/>) provides some additional information of the framework's usage and support.
- Bootstrap Blog (<http://blog.getbootstrap.com/>) is the best way to follow news about Bootstrap and read the releases notes.
- Bootstrap Expo (<http://expo.getbootstrap.com/>) is a showcase web page where you can see some beautiful websites that use the framework and resources to be used within such as plugins, themes, and so on.
- Stack Overflow questions related to Bootstrap (<http://stackoverflow.com/questions/tagged/twitter-bootstrap>). This is one of the best means of communication to get help from for your issues. Search for questions related to yours, and if you can't find something related, I guarantee that you will have an answer very soon.

There are many other resources spread across the Internet. Use them all to your advantage, and appreciate the taste of developing in fast pace with the best frontend framework of our time.

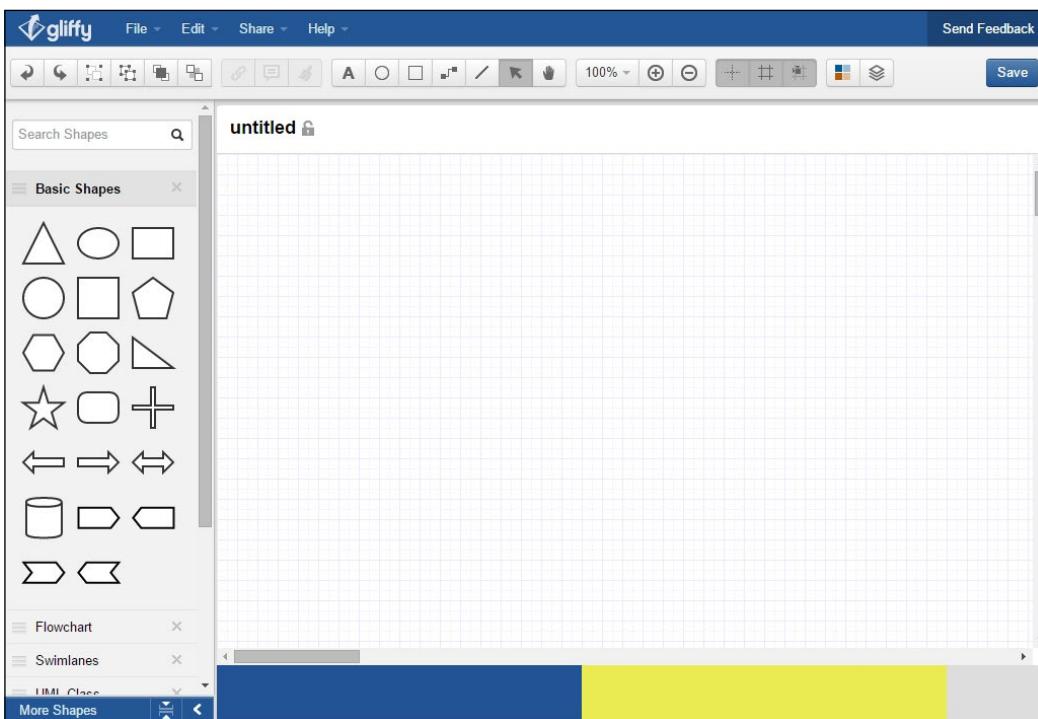
Tools

Bootstrap has an official HTML lint called Bootlint (<https://github.com/twbs/bootlint>). It checks the DOM for mistakes in using Bootstrap elements and components. Add Bootlint to avoid mistakes that delay your development. Check out the repository for installation and usage instructions.

Bootstrap and web applications

Bootstrap is one of the best frameworks for building web apps. Since you may use the same layout pattern across the web app, with premade classes and themes provided by the framework, you can speed up your development while maintaining the coherence of the elements used.

After the framework's release, Twitter adopted it like many other web apps as well. The following screenshot shows a great example of a fluid web app that uses Bootstrap with a fluid container:



Browser compatibility

The Bootstrap framework supports the most recent versions of the most used browsers. However, depending on the browser, the elements' rendering might look a little different from others, such as in the Chrome and Firefox Linux versions.

Internet Explorer's (IE) old versions do not have some properties from CSS3 and HTML5 that are used in the framework, so be aware of this when supporting these browsers. The following table presents the official browser compatibility.

Also, with the new version 4 of the framework, some compatibilities have been dropped. They decided to drop the support that existed for IE8, since it was dragging down the addition of new features, and now Bootstrap is able to take advantage of the use of some new CSS features.

With regard to this, version 4 moved from pixels to rems and ems measures to make responsive and resizing easier, and with that, they dropped support for iOS 6 as well:

	Chrome	Firefox	Internet Explorer	Opera	Safari
Android	Yes	Yes	N/A	No	N/A
iOS	Yes	N/A	N/A	No	Yes (iOS 7 + for v4)
OS X	Yes	Yes	N/A	Yes	Yes
Windows	Yes	Yes	Yes (IE9 + for v4)	Yes	No

The meaning of em and rem



The units em and rem have moved from trending to reality! They are enforced as present in our context and have now gained the support of Bootstrap. The main difference between em and rem is that they are relative unit metrics, while pixels are not. em is a unit relative to the parent font size and rem is a unit relative to the root element, perfectly fitting this responsivity development context.

Summary

In this chapter, you learned some basic concepts about using the Bootstrap framework. These are the key points for creating web sites with high quality. Knowing them in depth gives you a huge advantage and helps with the handling of future problems.

The chapter's goal was to show the recommended setup for the Bootstrap framework, presenting the placement of the tags, libraries import, and creating a very simple web page. Remember that consistency across the website is the main thing about Bootstrap, saving your precious time.

Also keep in mind that when starting a new web page, you have to guarantee a good placement of the main tags and components no matter how you created it (manually, boilerplate, or in other ways). Many problems stem from inadequate groundwork.

We also presented some resources from which you can acquire further knowledge or any kind of help. You now belong to a big "open arms" community that you can always count on.

Now that we have this background, let's attack some real-world problems! In the next chapter, we will start developing a very common real-life example, which is a landing page, while presenting some Bootstrap components, HTML elements, and grid systems.

2

Creating a Solid Scaffolding

In this chapter, you will start learning some new Bootstrap elements and components. By doing this, you will first understand the concepts of the Bootstrap grid system and move on to some basic components. Now, we are going to start the development of a responsive sample landing page. First, we will use the base theme of the framework, and in future, we will apply our own style.

The main structure of this chapter is as follows:

- The Bootstrap grid system
- Typography
- Tables
- Buttons

Understanding the grid system

The basis of the Bootstrap framework relies in its grid system. Bootstrap offers components that allow us to structure a website in a responsive grid layout.

To exemplify this, imagine an electronic square sheet table that can be divided into many rows and columns, as shown in the following screenshot. In the table, you can create as many lines as you want while merging cells. But what would happen if you wanted to change the layout of a single row? That could be painful.

	A	B	C	D	E
1	Sauce year selling				
2			2014	2015	Total
3	Ketchup	Restaurants	12000	18000	30000
4		Black market	5000	7500	12500
5		Reatilers	48500	72750	121250
6	Mustard	Black market	6590	9885	16475
7		Retailers	1576	2364	3940
8	Relish	Black market	589600	884400	1474000
9			663266	994899	1658165
10					

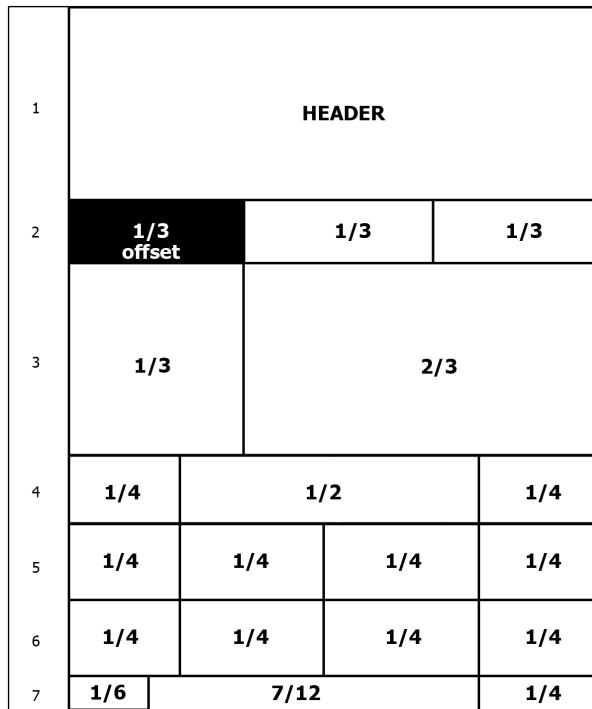
The Bootstrap grid system works in a different way. By letting you define a set of rows, each one having a set of independent columns, it allows you to build a solid grid system for your web page. Also, each column can have different sizes to perfectly fit your template.

This not being enough, the Bootstrap grid system adapts for every viewport and resolution, which we call responsiveness.

To start learning about the grid system, we will introduce it using the example of a landing page. As you will see, Bootstrap will allow us to create a complete scaffolding that will automatically adjust the content for any viewport.

Building our scaffolding

For our landing page, we will use the grid presented in the following image. As you can see, it is represented by seven rows, each containing a different number of columns. In this first example, we will use a nonmobile viewport, which we will discuss in the next chapter.



Setting things up

To start that, let's use our default layout presented in *Chapter 1, Getting Started*. Add inside the `div.container` tag another `div` with the `.row` class:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1">
    <title>Landing page</title>

    <link rel="stylesheet" href="css/bootstrap.css">

    <!-- [if lt IE 9 ]>
      <script
        src="https://oss.maxcdn.com/html5shiv/3.7.2/html5shiv.min.js">
    </script>
```

```
<script  
src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js">  
</script>  
<! [endif] -->  
</head>  
<body>  
  <div class="container">  
    <div class="row"></div>  
  </div>  
  
  <script src="js/jquery-1.11.3.js"></script>  
  <script src="js/bootstrap.js"></script>  
</body>  
</html>
```

The hierarchy for the grid system always follows the sequence of a container that wraps rows and multiple columns. Keep in mind to always use this sequence to get a proper output.

Now that we have our `.container` with the first `.row`, let's create our first column. Every row is subdivided into 12 parts, so you can have up to 12 columns in a single row.

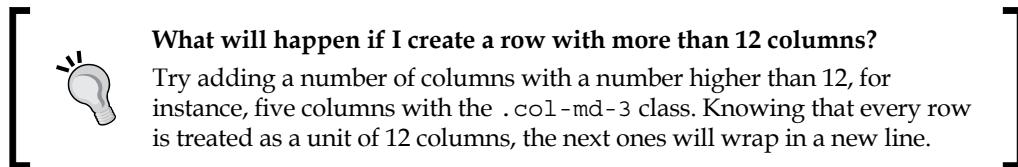
To identify a column, we must follow the template of `.col-*-*-*`. The first asterisk means the viewport for the column, and the other one means the size of the column. We will explain more about that, but to create our first column, we create a column identified by `.col-md-12` inside our row:

```
<div class="container">  
  <div class="row">  
    <header class="col-md-12">  
      HEADER  
    </header>  
  </div>  
</div>
```

In this column, the `md` in `.col-md-12` means that for the viewport medium (which means the `md` identifier), the column must have a 12-column width. In other words, the column fills the entire width of this row. This column will fill the complete width because it is our header, and as we can see in the previous image, this row is composed of just a single row.

So, to create a column in the Bootstrap grid system, you must follow the recipe of `.col-*-*` for every column. While you can set an integer from 1 to 12 for the width, for the viewport, you must set the correct class prefix. In this table, you can see the breakdown of class prefix usage and on which resolution it can be used:

	Extra small devices (phones < 544 px / 34 em)	Small devices (tablets ≥ 544 px / 34 em and < 768 px / 48 em)	Medium devices (desktops ≥ 768 px / 48 em and < 900 px / 62 em)	Large devices (desktops ≥ 900 px / 62 em and < 1,200 px / 75 em)	Extra large devices (desktops ≥ 1,200 px / 75 em)
Grid behavior	Horizontal lines at all times	Collapse at start and fit the column grid			
Container's fixed width	Auto	544 px or 34 rem	750 px or 45 rem	970 px or 60 rem	1170 px or 72.25 rem
Class prefix	<code>.col-xs-*</code>	<code>.col-sm-*</code>	<code>.col-md-*</code>	<code>.col-lg-*</code>	<code>.col-xl-*</code>
Number of columns	12				
Column fixed width	Auto	~ 44 px or 2.75 rem	~ 62 px or 3.86 rem	~ 81 px or 5.06 rem	~ 97 px or 6.06 rem



Offset columns

Our second row is divided into three equal-sized columns, and the first one is an offset column, or in other words, an empty column that will be filled by a left margin. Therefore, the second row will be like this:

```
<div class="row">
  <div class="col-md-offset-4 col-md-4">1/3</div>
  <div class="col-md-4">1/3</div>
</div>
```

As you can see, by adding the `.col-md-offset-4` class, we create a margin to the left four, sized in just this `.row`. By having each row independent of the others, we can properly customize the layout to appear just as it is supposed to be.



What happens if I add more than two offsets in a single column?

If you do that, you will find yourself in a funny situation. As a tip, only one offset is applied for an element, but which one? The answer is, the smallest offset!

Completing the grid rows

Now we will advance to the third row in our scaffolding. If you've got the spirit, you should have no problems with this row. For training, try doing it by yourself and check the solution in the book afterwards! I am sure you can handle it.

So, this row is composed of two columns. The first column must fill 4 out of the 12 parts of the row and the other column will fill the rest. The row in the HTML should look like this:

```
<div class="row">
  <div class="col-md-4"></div>
  <div class="col-md-8"></div>
</div>
```

About the fourth row—it is composed of a quarter divisor, followed by a half divisor, followed by a last quarter divisor. Using this in base 12, we will have the following grid in the row:

```
<div class="row">
  <div class="col-md-3">1/4</div>
  <div class="col-md-6">1/2</div>
  <div class="col-md-3">1/4</div>
</div>
```

Nesting rows

In the fifth and sixth rows, we will show how you can create a row using two options. In the fifth row, we will create just as we are doing in the other rows, while in the sixth row, we will use the concept of nesting rows.

So, in the fifth row, create it just as you were doing before; create a row with four equally sized rows, which means that each column will have the `.col-md-3` class:

```
<div class="row">
  <div class="col-md-3">1/4</div>
  <div class="col-md-3">1/4</div>
  <div class="col-md-3">1/4</div>
  <div class="col-md-3">1/4</div>
</div>
```

For the sixth row, we will use nesting rows. So, let's create the first `.row`, having three columns:

```
<div class="row">
  <div class="col-md-3">1/4</div>
  <div class="col-md-6"></div>
  <div class="col-md-3">1/4</div>
</div>
```

As you can see, the first and the last column use the same class of columns in row five—the `.col-md-3` class—while the middle column is double the size, with the `.col-md-6` class.

Let's nest another `.row` inside the middle column. When you create a new nested row, the columns inside of it are refreshed and you have another set of 12-sized columns to put inside it. So, inside this new row, create two columns with the `.col-md-6` class to generate two columns representing a fourth of the row:

```
<div class="row">
  <div class="col-md-3">1/4</div>
  <div class="col-md-6">
    <div class="row">
      <div class="col-md-6">1/4</div>
      <div class="col-md-6">1/4</div>
    </div>
  </div>
  <div class="col-md-3">1/4</div>
</div>
```

The concept of nesting rows is pretty complex, since you can infinitely subdivide a row, although it is great to create small grid components inside your page that can be used in other locations.

Finishing the grid

To create the last row, we need to create the `.col-md-2` column, followed by `.col-md-7` and `.col-md-3`. So, just create a row using the `<footer>` tag with those columns. The complete scaffolding will be this:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1">
    <title>Landing page</title>
    <link rel="stylesheet" href="css/bootstrap.css">
    <!-- [if lt IE 9]>
        <script
src="https://oss.maxcdn.com/html5shiv/3.7.2/html5shiv.min.js">
    </script>
        <script
src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js">
    </script>
    <!-- [endif]-->
  </head>
  <body>
    <div class="container">
      <!-- row 1 -->
      <div class="row">
        <header class="col-md-12">
          HEADER
        </header>
      </div>

      <!-- row 2 -->
      <div class="row">
        <div class="col-md-offset-4 col-md-4">1/3</div>
        <div class="col-md-4">1/3</div>
      </div>

      <!-- row 3 -->
      <div class="row">
        <div class="col-md-4"></div>
        <div class="col-md-8"></div>
      </div>
```

```
<!-- row 4 -->
<div class="row">
  <div class="col-md-3">1/4</div>
  <div class="col-md-6">1/2</div>
  <div class="col-md-3">1/4</div>
</div>

<!-- row 5 -->
<div class="row">
  <div class="col-md-3">1/4</div>
  <div class="col-md-3">1/4</div>
  <div class="col-md-3">1/4</div>
  <div class="col-md-3">1/4</div>
</div>

<!-- row 6 - nesting rows -->
<div class="row">
  <div class="col-md-3">1/4</div>
  <div class="col-md-6">
    <div class="row">
      <div class="col-md-6">1/4</div>
      <div class="col-md-6">1/4</div>
    </div>
  </div>
  <div class="col-md-3">1/4</div>
</div>

<!-- row 7 -->
<div class="row">
  <div class="col-md-2">1/2</div>
  <div class="col-md-7">7/12</div>
  <div class="col-md-3">1/4</div>
</div>

</div>

<script src="js/jquery-1.11.3.js"></script>
<script src="js/bootstrap.js"></script>
</body>
</html>
```

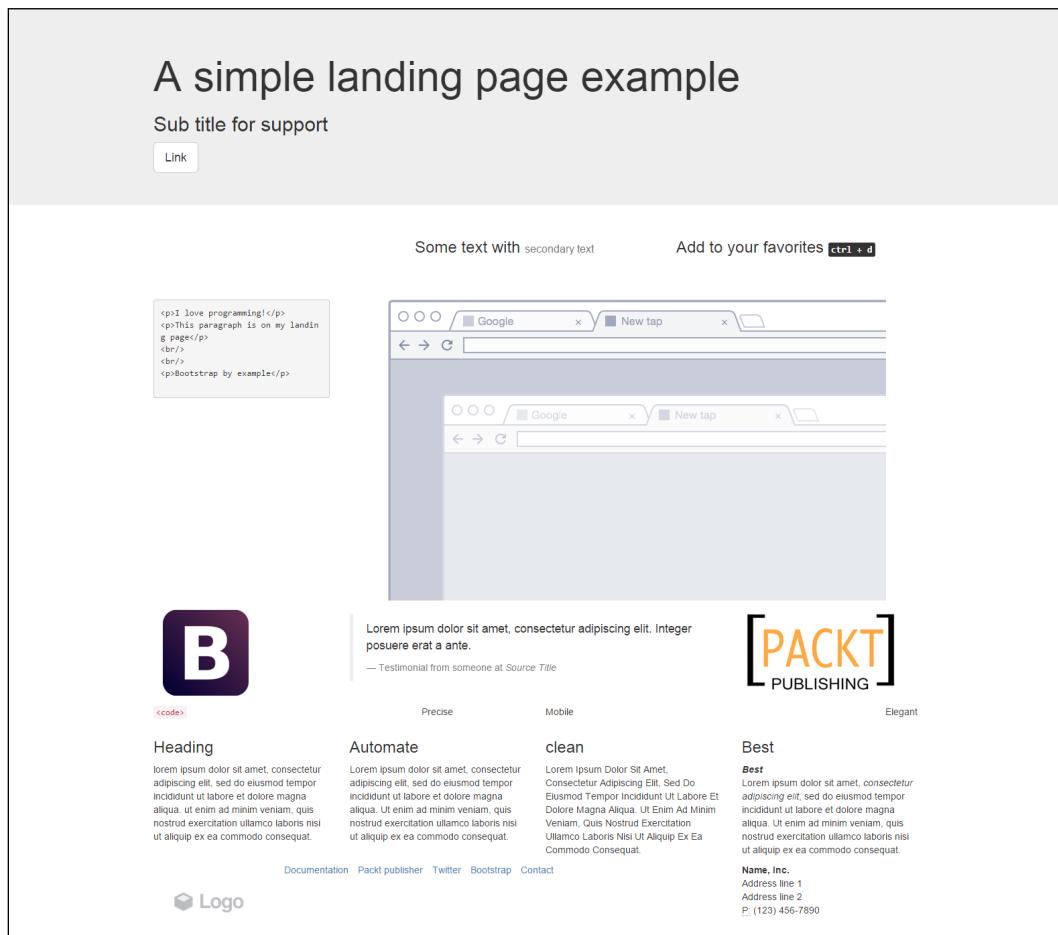
Fluid container

You can easily switch the actual example grid with a fluid full-width layout. To do so, replace the farthest `.container` with `.container-fluid`:

```
<div class="container-fluid">  
  ...  
</div>
```

We need some style!

Now, we will start using some of the CSS provided for Bootstrap to make our components responsive and more elegant. Our main goal is to make our grid page like what is shown in this screenshot:



Let's break down each row and learn about typography and some other components. We will do this without using a single custom line of CSS!

Getting started with the first row, you may see that this row has a gray background, which is not present in the rest of the layout. To create this, we must make a change in our grid by creating a new `.container` for this row. So, create another `.container` and place it inside the first row:

```
<div class="container">
  <!-- row 1 -->
  <div class="row">
    <header class="col-md-12">
      </header>
    </div>
  </div>
  <div class="container">
    <!-- the others rows (2 to 7) -->
  </div>
```

Now, to make the gray area, we will use a class in Bootstrap called `.jumbotron`. The `jumbotron` is a flexible Bootstrap component that can extend to the entire viewport to showcase some content, in this case the header. So, wrap the container inside a `div.jumbotron`:

```
<div class="jumbotron">
  <div class="container">
    <!-- row 1 -->
    <div class="row">
      <header class="col-md-12">
        </header>
      </div>
    </div>
  </div>
```

Inside the header, as we can see in the layout, we must create a title, a subtitle, and a button. For the title, let's use the `<h1>` and `<h2>` heading elements. For the button, let's create a link with the `.btn`, `.btn-default`, and `.btn-lg` classes. We will mention more about these components in the next subsections:

```
<div class="jumbotron">
  <div class="container">
    <!-- row 1 -->
    <div class="row">
      <header class="col-md-12">
        <h1>A simple landing page example</h1>
```

```
<h2>Sub title for support</h2>
<a class="btn btn-default btn-lg" href="#" role="button">
    Link
</a>
</header>
</div>
</div>
</div>
```

There are headings everywhere

Bootstrap provides styled headings from `h1` to `h6`. You should always use them in order of importance, from `<h1>` to `<h6>` (the least important).



Do you know why headings are important?

Headings are very important for **Search Engine Optimization (SEO)**. They suggest to search engines what is important in your page context. You must keep the hierarchy for page coherence, and do not skip any tag (that is, jump from heading 3 to heading 5). Otherwise, the structure will be broken and not relevant for SEO.

The heading has classes for identifying its style. So, if your most important phrase is not the biggest one at times, you can swap the sizes by adding heading classes, just as in the following example:

```
<h1 class="h3">Heading 1 styled as heading 3</h1>
<h2 class="h1">Heading 2 styled as heading 1</h2>
<h3 class="h2">Heading 3 styled as heading 2</h3>
```

Heading 1 styled as heading 3

Heading 2 styled as heading 1

Heading 3 styled as heading 2

Playing with buttons

The other element of the first row is a button! We can apply button classes for hyperlinks, button elements, and inputs. To make one of these elements a button, just add the `.btn` class followed by the kind of button, in this case the kind `.btn-default`, which is a blue button. The next table shows every possibility of color classes for a button:

Button class	Output
<code>.btn-default</code>	 Default
<code>.btn-primary</code>	 Primary
<code>.btn-success</code>	 Success
<code>.btn-info</code>	 Info
<code>.btn-warning</code>	 Warning
<code>.btn-danger</code>	 Danger
<code>.btn-link</code>	 Link

We have also added the `.btn-lg` class in the first row button. This class will increase the size of the button. Bootstrap also provides some other button size classes, such as `.btn-sm` for small buttons and `.btn-xs` for even smaller ones.

You can also make a button span the full width of the parent element with the `.btn-block` class, changing the display of the button to `block`.

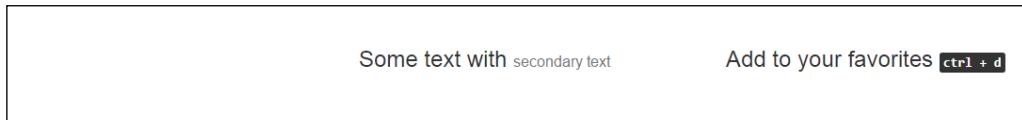
More typography and code tags

With regards to the second row, we have a row that contains a heading and complementary small text after that.

To add lighter and secondary text to the heading, we can add a `<small>` tag or any other tag with the `.small` class inside the heading. The HTML for the first column in the second row should be like the following:

```
<div class="row">
  <div class="col-md-offset-4 col-md-4">
    <h3>
      Some text with <small>secondary text</small>
    </h3>
  </div>
  <div class="col-md-4">
    <h3>
      Add to your favorites
      <small>
        <kbd><kbd>ctrl</kbd> + <kbd>d</kbd></kbd>
      </small>
    </h3>
  </div>
</div>
```

Note that inside the `small` tag, we have added a `<kbd>` tag, which is an HTML element that creates a user-like input keyboard. Refresh the web browser and you will see this row as shown here:

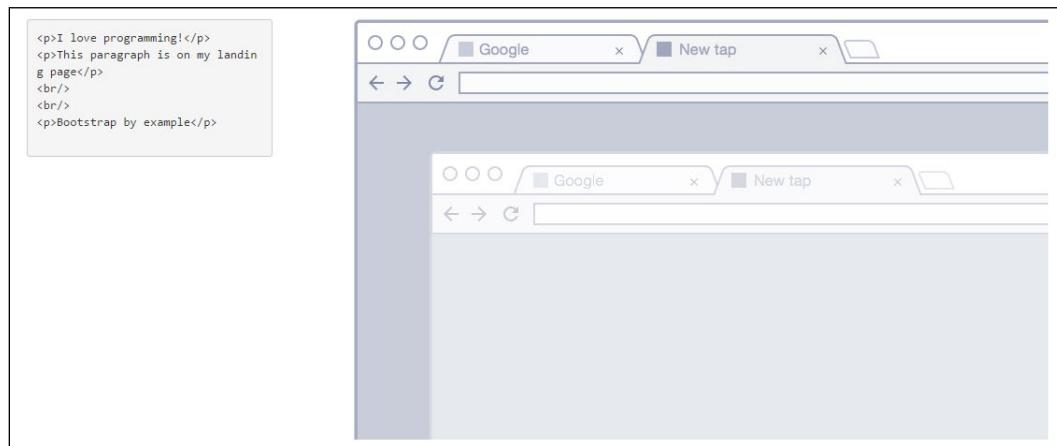


For the third row, we have a code snippet and an image. To create a code snippet, use the `<pre>` tag for multiple lines of code. The `<pre>` tag is present in HTML for creating preformatted text, such as a code snippet. You have the option of adding the `.pre-scrollable` class, which will add a scrollbar if the code snippet reaches the maximum height of 350 px (or 21.8 em).

For this row, in the right column, we have an image. For that, just create an `` tag and add the `.img-responsive` class, which will make the images automatically responsive-friendly to the viewport. The HTML for the third row is as follows:

```
<div class="row">
  <div class="col-md-3">
    <pre>&lt;p&gt;I love programming!&lt;/p&gt;
&lt;p&gt;This paragraph is on my landing page&lt;/p&gt;
&lt;br/&gt;
&lt;br/&gt;
&lt;p&gt;Bootstrap by example&lt;/p&gt;
    </pre>
  </div>
  <div class="col-md-9">
    
  </div>
</div>
```

Refresh your browser and you will see the result of this row as shown in the following screenshot:



In the fourth row, we have images in both the left and right columns and a testimonial in the middle. Bootstrap provides a typographic theme for doing block quotes, so just create a `<blockquote>` tag. Inside it, create a `<footer>` tag to identify the source, and wrap the name in a `<cite>` tag, like this:

```
<div class="row">
  <div class="col-md-3">
    
  </div>
  <div class="col-md-6">
    <blockquote>
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.  
Integer posuere erat a ante.</p>
      <footer>Testimonial from someone at <cite title="Source  
Title">Source Title</cite></footer>
    </blockquote>
  </div>
  <div class="col-md-3">
    
  </div>
</div>
```



Moving on, we must advance to the fifth row. This row is here just to show the different ways in which we can apply typography and coding elements tags using Bootstrap. Let's go through each one to describe its usage.

In the first column, we have a piece of inline code. To do that, wrap the snippet in a `<code>` tag. From the first to the fourth column of this row, we are presenting the alignment classes. Using these, you can easily realign text content in a paragraph tag. The code for the row is as follows:

```
<div class="row">
  <div class="col-md-3">
    <p class="text-left"><code>&lt;Left&gt;</code></p>
  </div>
  <div class="col-md-3">
    <p class="text-center">Center</p>
  </div>
  <div class="col-md-3">
    <p class="text-justify">Justify</p>
  </div>
</div>
```

```

</div>
<div class="col-md-3">
    <p class="text-right">Right</p>
</div>
</div>

```

Just use the right classes for a proper alignment. The result in the browser should look like this:

<Left>	Center	Justify	Right
--------	--------	---------	-------

The sixth row is composed of four equally divided columns, but in this case, we are using the nesting rows option. On the first three columns, we added Bootstrap text transformation classes to make the text lowercase, uppercase, and capitalized, respectively. The code for this row should be like the following:

```

<div class="row">
    <div class="col-md-3">
        <h3>Lowercase</h3>
        <p class="text-lowercase">
            Lorem ipsum dolor ... consequat.
        </p>
    </div>
    <div class="col-md-6">
        <div class="row">
            <div class="col-md-6">
                <h3>Uppercase</h3>
                <p class="text-uppercase">
                    Lorem ipsum dolor ... consequat.
                </p>
            </div>
            <div class="col-md-6">
                <h3>Capitalize</h3>
                <p class="text-capitalize">
                    Lorem ipsum dolor ... consequat.
                </p>
            </div>
        </div>
    </div>
    <div class="col-md-3">
        <h3>Strong and italic</h3>
        <p>
            <strong>Lorem ipsum</strong> dolor ... <em>consequat</em>.
        </p>
    </div>
</div>

```

Pay attention to the last column, where we are using the `` tags to make the text bold and the `` tag to make the text italic. Refresh your web browser and see the result, like this:

Lowercase	Uppercase	Capitalize	Strong and italic
<p>lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.</p>	<p>LOREM IPSUM DOLOR SIT AMET, CONSEQUETUR ADIPISCING ELIT, SED DO EIUSMOD TEMPOR INCIDIDUNT UT LABORE ET DOLORE MAGNA ALIQUA. UT ENIM AD MINIM VENIAM, QUIS NOSTRUD EXERCITATION ULLAMCO LABORIS NISI UT ALIQUIP EX EA COMMODO CONSEQUAT.</p>	<p>Lorem Ipsum Dolor Sit Amet, Consectetur Adipiscing Elit, Sed Do Eiusmod Tempor Incididunt Ut Labore Et Dolore Magna Aliqua. Ut Enim Ad Minim Veniam, Quis Nostrud Exercitation Ullamco Laboris Nisi Ut Aliquip Ex Ea Commodo Consequat.</p>	<p>Lorem ipsum dolor sit amet, <i>consequet adipiscing elit</i>, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo <i>consequat</i>.</p>

Alternative usage of bold and italic elements



You can use the `` and `<i>` tags to make the text bold and italic, respectively. Although, in HTML5 the `` tag is now used to stylistically offset, such keywords in paragraphs and the `<i>` tag are used for alternate voice markup.

Finally, we are going through the footer, which is the last row. If you take a look at the full layout image (the one presented at the beginning of this section), you will notice that it is composed of three columns. The first column contains a logo image, the middle one contains an inline list, and the last one has the company's contact address.

For the first column, we should just create an `` tag with the `.img-responsive` class. For the second column, the inline list, we must create a `` tag. By default, every `` inside a `` has the bullets on the left-hand side. To remove them, apply the `.unstyled` Bootstrap class. Also, a `` will create the `` elements as a block. In our case, we want the `` to appear side by side, so we use the `.list-inline` Bootstrap class to create this effect.

To present contact information in the last column, we will use the `<address>` tag. Bootstrap offers a CSS theme for this tag; you just need to keep the formatting along with the `
` tags, as shown in this code:

```
<footer class="row jumbotron">
  <div class="col-md-2">
    
  </div>
  <div class="col-md-7">
    <ul class="list-inline list-unstyled">
      <li><a href="#">Documentation</a></li>
      <li><a href="#">Packt publisher</a></li>
      <li><a href="#">Twitter</a></li>
      <li><a href="#">Bootstrap</a></li>
```

```

<li><a href="#">Contact</a></li>
</ul>
</div>
<div class="col-md-3">
<address>
  <strong>Name, Inc.</strong><br>
  Address line 1<br>
  Address line 2<br>
  <abbr title="Phone">P:</abbr> (123) 456-7890
</address>
</div>
</footer>

```

Pay attention to the `<footer>` tag. We added the `.jumbotron` class to make it rounded and give it a gray background. The following screenshot presents to us the result of the footer:



Manipulating tables

The Bootstrap framework offers a wide variety for table customization. To present them, we will create a new row before the `<footer>` and a price table for our landing page, like this:

Free plan	Standard plan	Premium plan
\$ 0	\$ 99	\$ 999
Lorem ipsum	Lorem ipsum	Lorem ipsum
Lorem ipsum	Lorem ipsum	Lorem ipsum
Dolor sit amet	Lorem ipsum	Lorem ipsum
-	Dolor sit amet	Lorem ipsum
-	-	Lorem ipsum
Purchase	Purchase	Purchase

To do this, we must create a regular table with the `<table>`, `<thead>`, `<tbody>`, `<tr>`, `<th>`, and `<td>` tags. The table will have three columns and eight rows. Therefore, the HTML code should be like this:

```
<div class="row">
  <div class="col-md-10 col-md-offset-1">
    <table>
      <thead>
        <tr>
          <th>
            <h4>Free plan</h4>
          </th>
          <th>
            <h4>Standard plan</h4>
          </th>
          <th>
            <h4>Premium plan</h4>
          </th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td>
            <h3>$ 0</h3>
          </td>
          <td>
            <h3>$ 99</h3>
          </td>
          <td>
            <h3>$ 999</h3>
          </td>
        </tr>
        <tr>
          <td>Lorem ipsum</td>
          <td>Lorem ipsum</td>
          <td>Lorem ipsum</td>
        </tr>
        <tr>
          <td>Lorem ipsum</td>
          <td>Lorem ipsum</td>
          <td>Lorem ipsum</td>
        </tr>
        <tr>
          <td>Dolor sit amet</td>
```

```
<td>Lorem ipsum</td>
<td>Lorem ipsum</td>
</tr>
<tr>
    <td>-</td>
    <td>Dolor sit amet</td>
    <td>Lorem ipsum</td>
</tr>
<tr>
    <td>-</td>
    <td>-</td>
    <td>Lorem ipsum</td>
</tr>
<tr>
    <td><a href="#">Purchase</a></td>
    <td><a href="#">Purchase</a></td>
    <td><a href="#">Purchase</a></td>
</tr>
</tbody>
</table>
</div>
</div>
```

Right now, we have no secrets in our table. Let's start using CSS Bootstrap styles! First of all, add the `.table` class to the `<table>` tag (duh!). This seems redundant, but it's an option of the framework used to make it explicit.

Then, we will apply some specific styles. To make the rows striped, we add the `.table-striped` class to `<table>` as well. We want this table to have borders, so we add the `.table-bordered` class to make it like that. Last but not least, add the `.table-hover` class to enable hover state in the `<tbody>` rows.

Now we will move on to the `<tr>` tag inside `<thead>`. To make its background green, we add the `.success` class. Similar to buttons, every cell, row, or table in a `<table>` tag can receive color classes, officially called Bootstrap contextual classes.

Contextual classes follow the same colors meant for buttons. So, for the second column, we apply the `.info` class to get a cyan background color, and we use a `.danger` class to get a red background color in the last column.

[ The framework also offers the `.active` class, which offers the same color of hover and the `.warning` class, which offers a yellow color.]

Inside each `<th>` tag, we have an `<h4>` typography tag. If you take a look at the image showing how the table should look, you will notice that the heading texts are in the center. You may remember how to do that; just apply the `.text-center` class in the headings.

The themed `<thead>` tag will be like this:

```
<thead>
  <tr>
    <th class="success">
      <h4 class="text-center">Free plan</h4>
    </th>
    <th class="info">
      <h4 class="text-center">Standard plan</h4>
    </th>
    <th class="danger">
      <h4 class="text-center">Premium plan</h4>
    </th>
  </tr>
</thead>
```

Now we will move on to the first table row in `<tbody>`, which is the price row. We just need to center `<h4>` in the same way as we did in the `<thead>` row — by adding the `.text-center` class:

```
<h3 class="text-center">$ 0</h3>
```

The next five rows have no specific style, but the last one has buttons and some tricks!

Styling the buttons

Do you remember how to apply the color theme in the buttons? You just need to follow the `<thead>` column color style, prepending `.btn-*` in the Bootstrap contextual classes. For instance, the first one will have the `.btn-success` class to turn a green button.

Furthermore, the button must fill the full width of the cell. To make the button span the complete parent width, add the `.btn-block` class and the magic is completely done! The code for the last row is as follows:

```
<tr>
  <td><a href="#" class="btn btn-success btn-block">Purchase</a></td>
  <td><a href="#" class="btn btn-info btn-block">Purchase</a></td>
  <td><a href="#" class="btn btn-danger btn-block">Purchase</a></td>
</tr>
```

Like a boss!

Right now, we have finished the first version of our landing page! It should look like what is shown in the next screenshot. Note that we did it without a single line of custom CSS!:

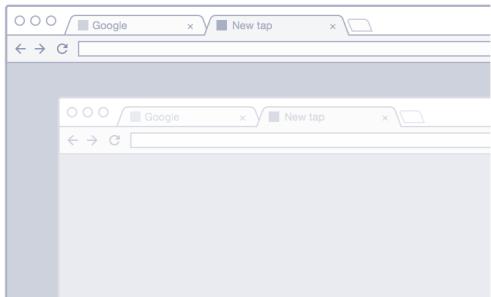
A simple landing page example

Sub title for support

[Link](#)

Some text with secondary text
Add to your favorites **ctrl + d**

```
<p>I love programming!</p>
<p>This paragraph is on my landing page.</p>
<br/>
<br/>
<p>@Bootstrap by example</p>
```





<Left>

Lowercase

lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.

— Testimonial from someone at Source Title

Center

Uppercase

LOREM IPSUM DOLOR SIT AMET, CONSECTETUR ADIPISCING ELIT, SED DO EIUSMOD TEMPOR INCIDUNT UT LABORE ET DOLORE MAGNA ALIQUA. UT ENIM AD MINIM VENIAM, QUIS NOSTRUD EXERCITATION ULLAMCO LABORIS NISI UT ALIQUIP EX EA COMMODO CONSEQUAT.

Justify

CapitalizE

lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Right

Strong and italic

LOREM IPSUM dolor sit amet, consectetur adipisc**i**ng el**i**t, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitatio**n** ullamco laboris nisi ut aliquip ex ea commodo consequat.

Free plan	Standard plan	Premium plan
\$ 0	\$ 99	\$ 999
Lorem ipsum	Lorem ipsum	Lorem ipsum
Lorem ipsum	Lorem ipsum	Lorem ipsum
Dolor sit amet	Lorem ipsum	Lorem ipsum
-	Dolor sit amet	Lorem ipsum
-	-	Lorem ipsum
Purchase	Purchase	Purchase

Documentation | Packt publisher | Twitter | Bootstrap | Contact

 Name, Inc.
Address line 1
Address line 2
P: (123) 456-7890

This is the power of Bootstrap. Can you feel it? You can do beautiful things very quickly. Bootstrap is a must-have tool for prototyping!

Change the viewport of your page by resizing the window and you will see how nicely Bootstrap adapts to any resolution. You don't have to worry about it if you have done a great job at making the grid and placing the elements.

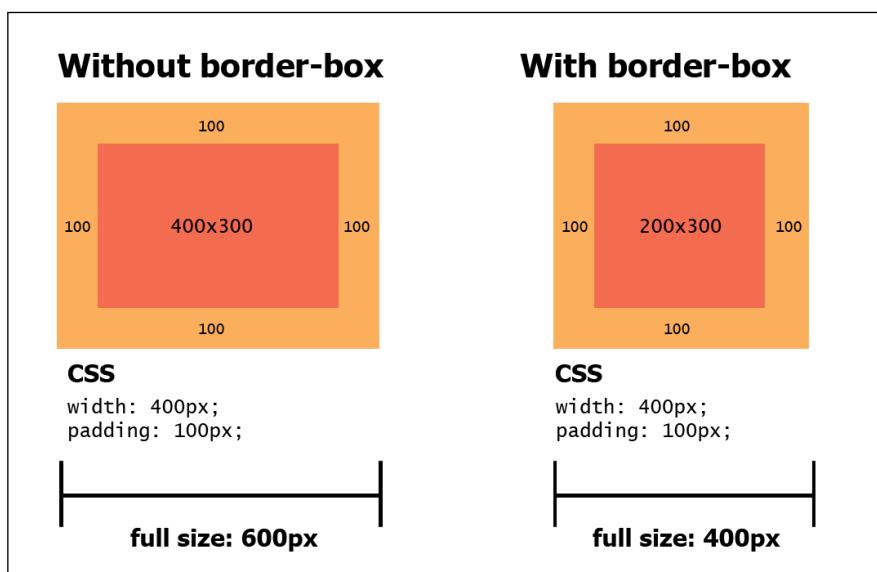
Final thoughts

Before ending this chapter, we must get some things clear. Bootstrap offers some helper classes mixins and some vendor's mixins, which offer cross-browser compatibility support.

Box-sizing

Bootstrap 3 started using box-sizing: border-box for all elements and pseudo-elements. With this enabled, the width and height properties start to include the padding and the border but not the margin.

With that, it is easier to set the right sizing for your elements. This is because any change that you make in the element, such as customizing the padding of a `.row`, will reflect the width and height of the whole element. The following figure shows you the differences of box-sizing.





Pseudo-elements are the text placed after the `:` sign in the CSS style. They are used to style specific parts of elements, such as `:after` and `:before`, which are the most common pseudo-elements.

Quick floats

Bootstrap offers quick classes to make an element float. Add the `.pull-left` or `.pull-right` class to make the elements float left or right, respectively. Keep in mind that both classes apply the `!important` modifier to avoid override issues:

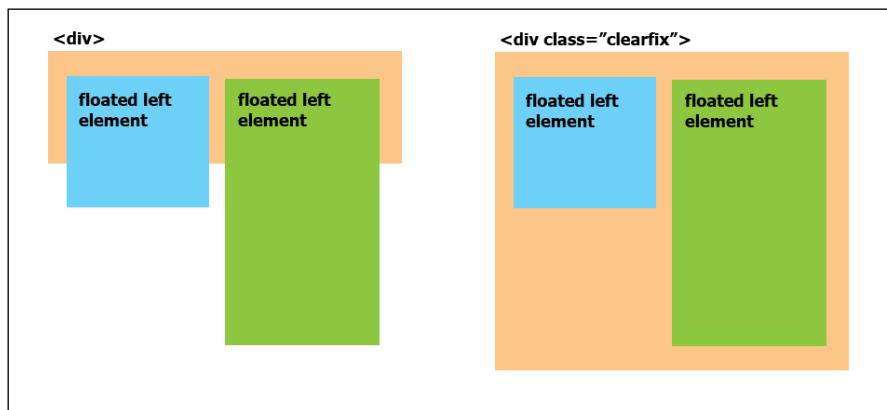
```
<div class="pull-left"></div>
<div class="pull-right"></div>
```



In the next chapters, we will present the navbar components. Remember that if you want to align an element inside a navbar, you should use `.navbar-left` and `.navbar-right` instead of `.pull-left` and `.pull-right`.

Clearfix

Clearfix is a way of clearing the floating of an element related to its child element. The Bootstrap columns are all floated left, and the parent row has a clearfix. This makes every column appear right next to each other, and the row does not overlap with other rows. This figure exemplifies how clearfix works:



So, if you want to add the clearfix to a new element or pseudo-element, add the `.clearfix` class and you can get the hard work quickly done.

Font definitions for typography

In the following table, the font sizes for default text and heading are presented. It describes the heading, font family, and line height. It is important to make it explicit for you to deeply understand the Bootstrap default configuration if you want a different customization.

CSS option	Default value
font-family	"Helvetica Neue", Helvetica, Arial, and sans-serif
line-height	1.42857143 (almost 20 px)
font-size	14 px (0.875 em)
h1 font-size	36 px (2.25 em)
h2 font-size	30 px (1.875 em)
h3 font-size	24 px (1.5 em)
h4 font-size	18 px (1.125 em)
h5 font-size	14 px (0.875 em)
h6 font-size	12 px (0.75 em)
Heading line-height	16 px (1 em)

Summary

In this chapter, we started our example of creating a landing page. By now, we can create a beautiful page without a single line of CSS or JavaScript!

First, we were able to reveal the secrets behind Bootstrap's scaffolding and understand its proper usage. You learned how to set rows inside a container, columns inside rows, and settings classes for a specific viewport. We played some tricks on columns as well, making nested rows, offsetting columns, and using multiple containers.

The basis of scaffolding will be important throughout the book, and it is in fact the basis of the Bootstrap framework. The power to manipulate it is a key factor in understanding the framework. I advise you to try out some new combinations of rows in your landing page and see what happens. Further in this book, we will show you some other grid combinations and custom layouts.

Furthermore, we played with buttons, which is another key factor in Bootstrap. We presented some of the basis of button configurations and customizations. During the rest of this book, some more options will be presented in many different ways, but respecting the basis that you have just learned.

Tables are also a very common element in web pages, and Bootstrap offers a wide variety of customizations for them. We showed an example with all the main table features that you can for sure use in your daily tasks.

Finally, we saw some tricks of the framework. As I already said, you must understand the roots of Bootstrap to understand the magic. In an easy way, Bootstrap offers helpers to make our work as fast as it can get.

In the next chapter, we will dive into mobile-first development and different viewport configurations, making our landing page best fit for any device. We will also show a nice way to debug our page for any virtual devices.

3

Yes, You Should Go Mobile First

You should be asking yourself, "I thought that we should first do the layout in mobile and then go to the desktop version. Why are we in the opposing way?"

Sorry, you are right! We should always go mobile-first. We went the opposite direction just for learning purposes and now we are going to fix it.

In the current chapter, we will focus on mobile design and site responsiveness with the help of the Bootstrap framework by learning how to change the page layout for different viewports, changing the content, and more. The key points of the chapter are as follows:

- Mobile-first development
- Debugging for any device
- Bootstrap grid system for different resolutions

To figure out what, we will continue with the landing page that we developed in the last chapter.

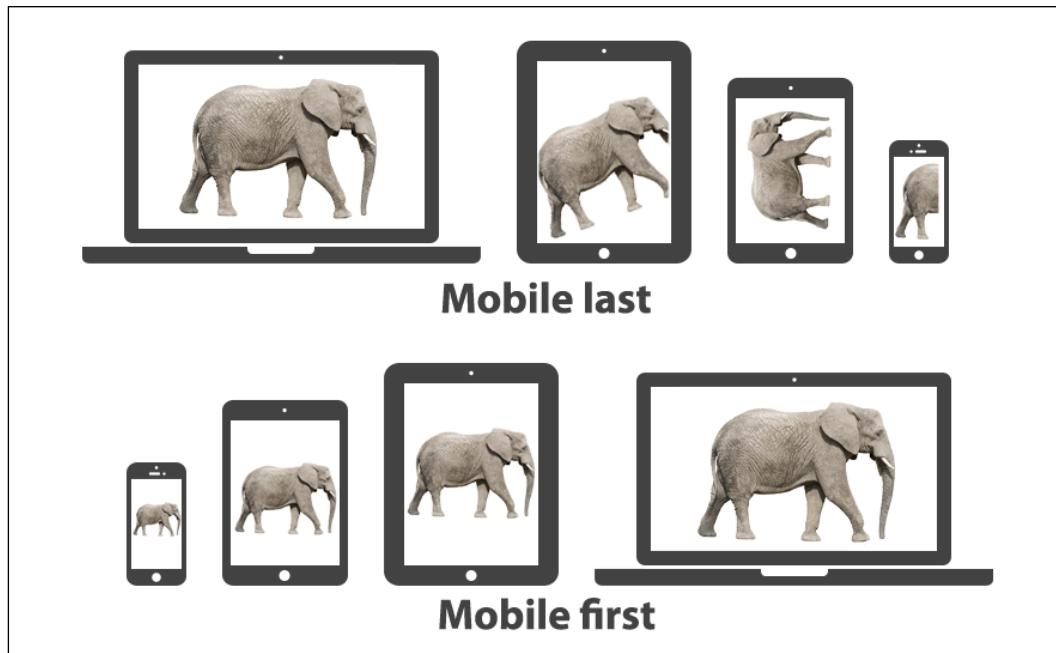
Making it greater

Maybe you have asked yourself (or even searched for) the reason for the mobile-first paradigm trend. It is simple and makes complete sense for speeding up your development.

The main argument for the mobile-first paradigm is that it is easier to make it than to shrink it. In other words, if you make a desktop version of the web page (known as responsive design or mobile last) first and then adjust the website for mobile, it has a 99 percent probability of breaking the layout at some point and you will have to fix a lot of things.

On the other hand, if you create the mobile version first, naturally the website will use (or show) less content than the desktop version. So, it will be easier to just add the content, place the things in the right places, and create the fully responsiveness stack.

The following figure tries to illustrate this concept. Going mobile last, you will get a degraded, sharped, and crappy layout and you will get a progressively enhanced, future-friendly, awesome web page if you go mobile first. You can see what happens to the poor elephant... Mobile-first naturally grows the elephant instead of adjusting it:



Bootstrap and the mobile-first design

In the beginning of Bootstrap, there was no concept of mobile-first. It was first used for responsive design web pages. With the Version 3 of the framework, the concept of mobile-first became very solid in the community.

The whole code of the scaffolding system was rewritten to become mobile-first from the start. They decided to reformulate how to set the grid instead of just adding mobile styles. This made a great impact in compatibility between versions older than v3, but was crucial for making the framework even more popular.

As we saw in the first chapter, to ensure the proper rendering of the page, set the correct viewport at the `<head>` tag:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

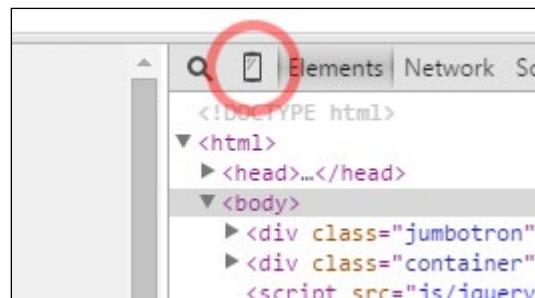
How to debug different viewports at the browser

Let's see how to debug different viewports using the Google Chrome web browser. Even if you already know that you can skip this section, it is important to refresh the steps for doing that.

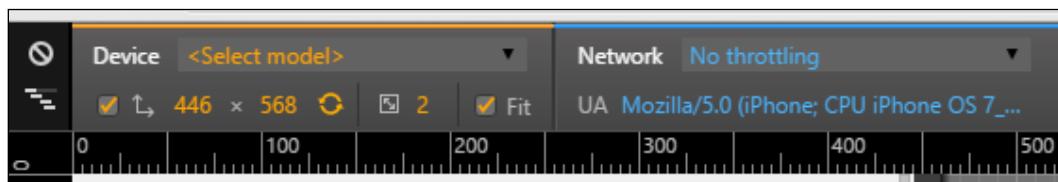
First of all, open the current landing page project that we will continue working with in this chapter in the Google Chrome browser. In the page, you need to select the **Developer tools** option. There are many ways to open this menu:

- Right-click at any place on the page and click on the last option **Element inspector**
- Go to the setting (the sandwich button at the top-right of the address bar), click on **More tools**, and select **Developer tools**
- The shortcut to open it is *Ctrl + Shift + I* (*cmd* for OS X users)
- *F12* in Windows also works (this is an Internet Explorer legacy)

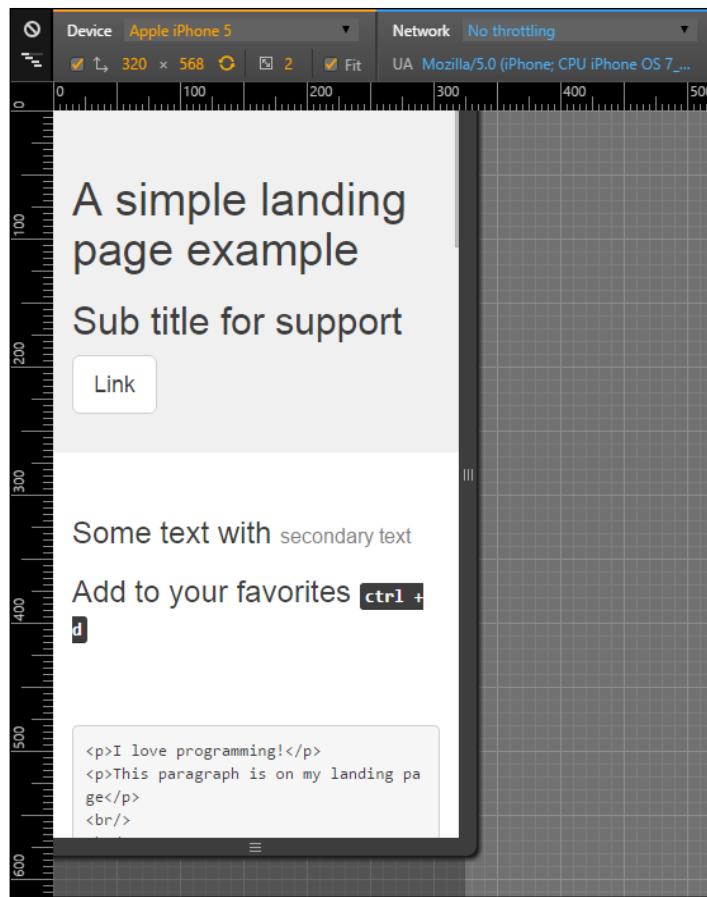
In the **Developer tools**, click in the mobile phone on the left of a magnifier, as shown in the following screenshot:



It will change the display of the viewport to a certain device and you can also set a specific network usage to limit the data bandwidth. Chrome will show a message telling you that for proper visualization you may need to reload the page to get the correct rendering.



As shown in the next screenshot, we have activated the **Device** mode for an iPhone 5 device. When we set this viewport, problems started appearing because we did not make the landing page with mobile-first methodology.



The first problem is in the second row of our layout. See how *Ctrl + D* breaks to a new line. That is not supposed to happen.

Another problem is that we have a horizontal scroll for this device due to some unknown reason. That sucks! We will have more work than with the opposite direction that starts with the mobile page. Keep it as a lesson for not repeating the same mistake.

Now, we can debug our website in different devices with different resolutions. You may see that the mouse cursor has changed to a gray circle. Also, the click actions have changed to tap actions. With that, you can fully test the website without the physical device.

Let's first clean out the messy parts in the layout before playing some tricks with the mobile version.

Cleaning up the mess

First, we will stop the line from breaking in the *Ctrl + D* text in the second row of our design. For fixing this issue, we will create our first line of CSS code. Add the `<head>` tag to a custom CSS file. Remember to place it below the `bootstrap.css` import line:

```
<link rel="stylesheet" href="css/base.css">
```

In the `base.css` file, create a helper class rule for `.nowrap`:

```
.nowrap {  
    white-space: nowrap;  
}
```

In the HTML file, add the created class to the `<kbd>` element (line 43):

```
<kbd class="nowrap"><kbd>ctrl</kbd> + <kbd>d</kbd></kbd>
```

Reload the page and you'll see that one problem is solved. Now, let's fix the horizontal scroll. Can you figure out what is making the unintended horizontal scroll? A tip, the problem is in the table!

What is the meaning of the `white-space` CSS property?

The `white-space` property specifies how whitespace is handled inside an element. The default is `normal`, where the line breaks will occur when needed. `nowrap` will prevent line break by creating a new line and `pre` will only wrap on line breaks.

The problem is caused by the buttons with the `display: block` that make the content of each column larger than intended. To fix this, we will create our first CSS media query:

```
@media (max-width: 48em) {  
    table .btn {  
        font-size: 0.75rem;  
        font-size: 3.5vw;  
    }  
}
```

Breaking down each line, first we see the current media query. The rule is that for `max-width` of `48em` (defined in Bootstrap as small devices), apply the following rules. If the view port is greater than `48em`, the rule will not be applied.

For the `.btn` elements inside the `table` element, we changed the font size (this was causing the horizontal overflow). We used a new way to set the font size based on the viewport with the `3.5vw` value. Each `1vw` corresponds to 1 percent of the total viewport. If we change the viewport, we will change the font size dynamically without breaking the layout.

Since it is a new property, nowadays just Chrome 20-34 and Safari 6 or higher have this rendering feature. For this reason, we added the other line with `font-size: 0.75rem` as a fallback case. If the browser can't handle the viewport font size, it will already had decreased the font to `12px`, which is a font that does not break the layout.

Creating the landing page for different devices

Now that we have fixed everything and learned some things about media queries and CSS3 properties, let's play with our layout and change it a bit for different devices. We will be starting with mobile and go further until we reach large desktops.

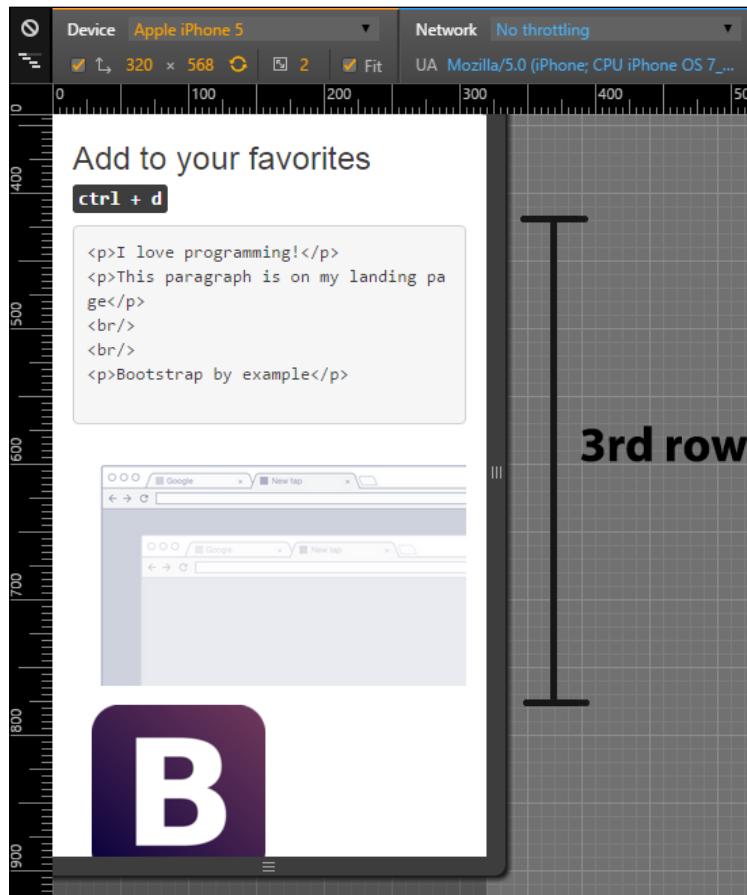
To do so, we must apply the column class for the specific viewport, as we did for medium displays using the `.col-md-*` class. The following table was presented in the previous chapter to show the different classes and the resolutions applicable for specific classes:

	Extra small devices (phones < 544px / 34em)	Small devices (tablets ≥ 544px / 34em and < 768px / 48em)	Medium devices (desktops ≥ 768px / 48em < 900px / 62em)	Large devices (desktops ≥ 900px / 62em < 1200px / 75em)	Extra-large devices (Desktops ≥ 1200px / 75em)
Grid behavior	Horizontal lines at all times	Collapse at start and fit column grid			
Container fixed width	Auto	544px or 34rem	750px or 45rem	970px or 60rem	1170px or 72.25rem
Class prefix	<code>.col-xs-*</code>	<code>.col-sm-*</code>	<code>.col-md-*</code>	<code>.col-lg-*</code>	<code>.col-xl-*</code>
Number of columns	12 columns				
Column fixed width	Auto	~ 44px or 2.75rem	~ 62px or 3.86rem	~ 81px or 5.06rem	~ 97px or 6.06rem

Mobile and extra small devices

To adapt our landing page to mobile devices, we will be using the Chrome mobile debug tool with the device iPhone 5 set and no network throttling.

You might have noticed that for small devices, Bootstrap just stacks each column without the referring for different rows. Some of our rows seem fine in this new grid, like the header and the second one. In the third row, it is a bit strange that the portion of code and the image are not in the same line, as shown in the following screenshot:



For doing that, we need to add the class columns prefix for extra small devices, which is `.col-xs-*`, where * is the size of the row from 1 to 12. Add the class `.col-xs-5` and `.col-xs-7` for the columns of the respective row (near line 49). Refresh the page and you will see now how the columns are side-by-side. The code is as follows:

```
<div class="row">
  <!-- row 3 -->
  <div class="col-md-3 col-xs-5">
    <pre>&lt;p&gt;I love programming!&lt;/p&gt;
&lt;p&gt;This paragraph is on my landing page&lt;/p&gt;
&lt;br/&gt;
&lt;br/&gt;
&lt;p&gt;Bootstrap by example&lt;/p&gt;
  </pre>
</div>
<div class="col-md-9 col-xs-7">
  
</div>
</div>
```

Although the image of the web browser is too small on the right, it would be better if it was a more vertical stretched image, such a mobile phone (what a coincidence!). To make it, we need to hide the browser image in extra small devices and display an image of a mobile device. Add the new mobile image below the old one, as shown in the code:

```

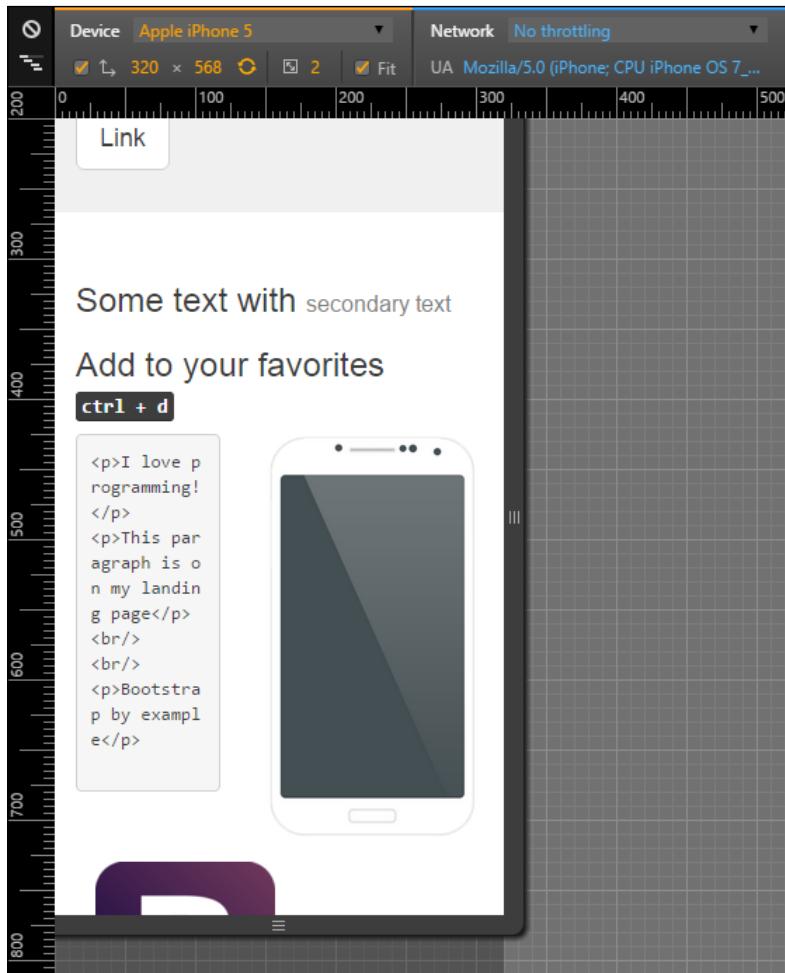
```

You will see both images stacked up vertically in the right column. Then, we need to use a new concept of availability classes. We need to hide the browser image and display the mobile image just for this kind of viewport, which is extra small. For that, add the class `.hidden-xs` in the browser image and add the class `.visible-xs` in the mobile image:

```
<div class="row">
  <!-- row 3 -->
  <div class="col-md-3 col-xs-5">
    <pre>&lt;p&gt;I love programming!&lt;/p&gt;
&lt;p&gt;This paragraph is on my landing page&lt;/p&gt;
&lt;br/&gt;
&lt;br/&gt;
&lt;p&gt;Bootstrap by example&lt;/p&gt;
  </pre>
```

```
</div>
<div class="col-md-9 col-xs-7">
  
  
</div>
</div>
```

Now this row seems nice! The browser image was hidden in extra small devices and the mobile image is shown only for this viewport in question. The following screenshot shows the final display of this row:



Moving on to the next row, the fourth one, it is the testimonial row surrounded by two images. It would be nicer if the testimonial appeared first and both of the images were displayed after it, splitting the same row. For this, we will repeat almost the same techniques presented in the previous row. Let's do it again for practice.

The first change is to hide the Bootstrap image with the class `.hidden-xs`. After that, create another image tag with the Bootstrap image in the same column of the PACKT image. The final code of the row should be like this:

```
<div class="row">
    <!-- row 4 -->
    <div class="col-md-3 hidden-xs">
        
    </div>
    <div class="col-md-6 col-xs-offset-1 col-xs-11">
        <blockquote>
            <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
            Integer posuere erat a ante.</p>
            <footer>Testimonial from someone at <cite title="Source
            Title">Source Title</cite></footer>
        </blockquote>
    </div>
    <div class="col-md-3 col-xs-7">
        
    </div>
    <div class="col-xs-5 visible-xs">
        
    </div>
</div>
```

We made plenty of things now and they are highlighted in bold. First is the `.hidden-xs` in the first column of Bootstrap image, which hid the column for this viewport.

Afterwards, in the testimonial, we changed the grid for mobile, adding a column offset with size 1 and making the testimonial fill the rest of the row with the class `.col-xs-11`.

Finally, as we said, we want to split in the same row both images from PACKT and Bootstrap. For that, make the first image column fill seven columns with the class `.col-xs-7`.

The other image column is a little more complicated. Since it is just visible for mobile devices, we add the class `.col-xs-5`. This will make the element span five columns in extra small devices. Moreover, we hide the column for other viewports with the class `.visible-xs`.

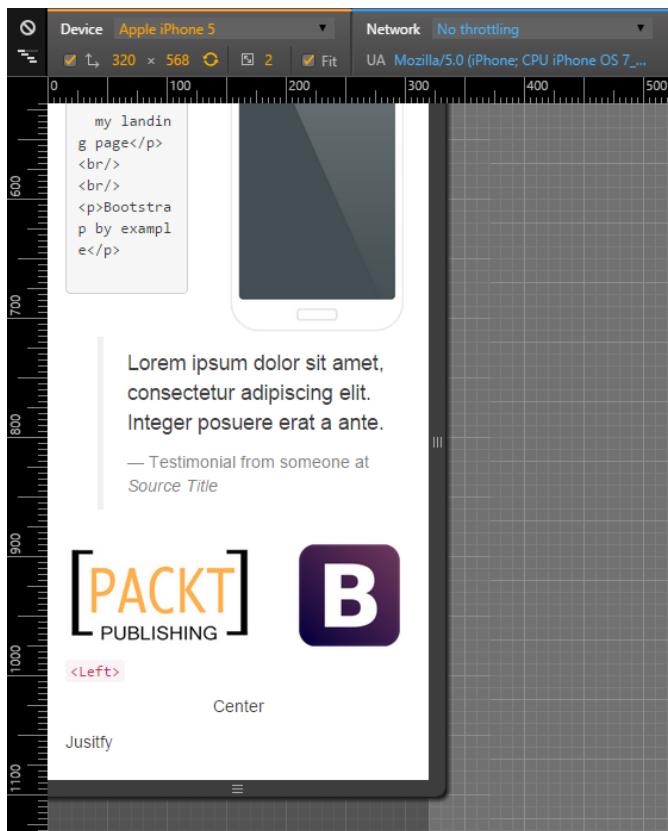
As we can see, this row has more than 12 columns (1 offset, 11 testimonial, 7 PACKT image, 5 Bootstrap image). This process is called column wrapping, and it happens when you put more than 12 columns in the same row so the groups of extra columns will wrap to the next lines.

Availability classes



Just like the `.hidden-*`, there are the `.visible-*-*` classes for each variation of display and column from 1 to 12. There is also a way to change the display CSS property using the class `.visible-*-*`, where the last * means block, inline, or inline-block. Use this to set the properly visualization for different visualizations.

The following screenshot shows the result of the changes. As you can see, we made the testimonial appears first, with one column of offset and both images appearing below it:



Tablets and small devices

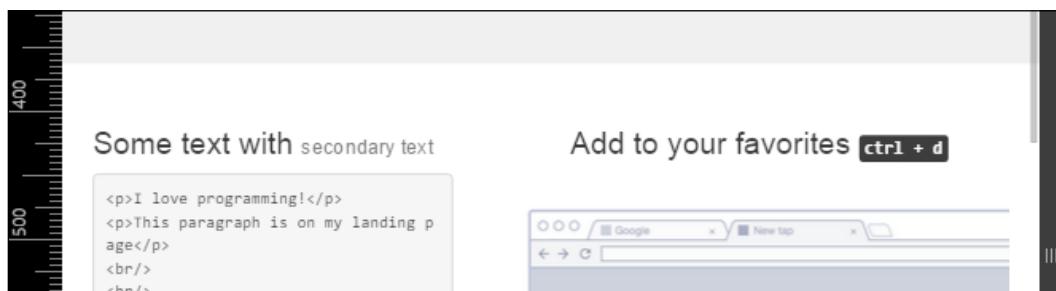
After completing the mobile visualizations, let's go further to tablets and small devices, which are devices from 48em to 62em. Most of these devices are tablets or old desktop monitors. For this example, we are using the iPad Mini in the portrait position with a resolution of 768 x 1024 pixels.

For this resolution, Bootstrap handles the rows just like extra small devices by just stacking up each one of the columns, making them fill the total width of the page. So if we do not want that to happen, we have to manually set the column fill for each element with the class `.col-sm-*`.

If you see how our example is presented now, there are two main problems. The first one is the second row, where the headings are in separated lines when they could be in the same. So, we just need to apply the grid classes for small devices with the class `.col-sm-6` for each column, splitting them into equal sizes:

```
<div class="row">
  <div class="col-md-offset-4 col-md-4 col-sm-6">
    <h3>
      Some text with <small>secondary text</small>
    </h3>
  </div>
  <div class="col-md-4 col-sm-6">
    <h3>
      Add to your favorites
    <small>
      <kbd class="nowrap"><kbd>ctrl</kbd> + <kbd>d</kbd></kbd>
    </small>
    </h3>
  </div>
</div>
```

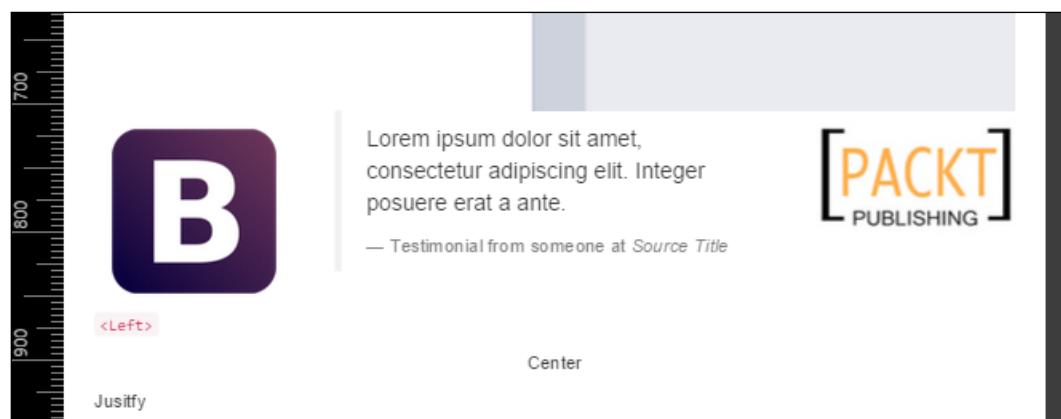
The result should be as shown in the following screenshot:



The second problem in this viewport is again the testimonial row! Because of the classes that we have added for mobile viewport, now the testimonial has an offset column and different column span. We must add the classes for small devices and make this row with the Bootstrap image on the left, the testimonial in the middle, and the PACKT image at the right position:

```
<div class="row">
  <div class="col-md-3 hidden-xs col-sm-3">
    
  </div>
  <div class="col-md-6 col-xs-offset-1 col-xs-11 col-sm-6 col-sm-offset-0">
    <blockquote>
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.</p>
      <footer>Testimonial from someone at <cite title="Source Title">Source Title</cite></footer>
    </blockquote>
  </div>
  <div class="col-md-3 col-xs-7 col-sm-3">
    
  </div>
  <div class="col-xs-5 hidden-sm hidden-md hidden-lg">
    
  </div>
</div>
```

As you can see, we had to reset the column offset in the testimonial column. It happened because it kept the offset that we added for extra small devices. Moreover, we are just ensuring that the images columns had to fill just three columns. Here's the result:



Everything else seems fine! These viewport was easier to setup. See how Bootstrap helps us a lot? Let's move to the final viewport: desktop or large devices.

Desktop and large devices

Last but not least, we enter the grid layout for desktop and large devices. We skipped medium devices, because we first coded for that viewport.

Deactivate the *device mode* in Chrome and put your page in a viewport with a width larger or equal to 1200 pixels or 75em.

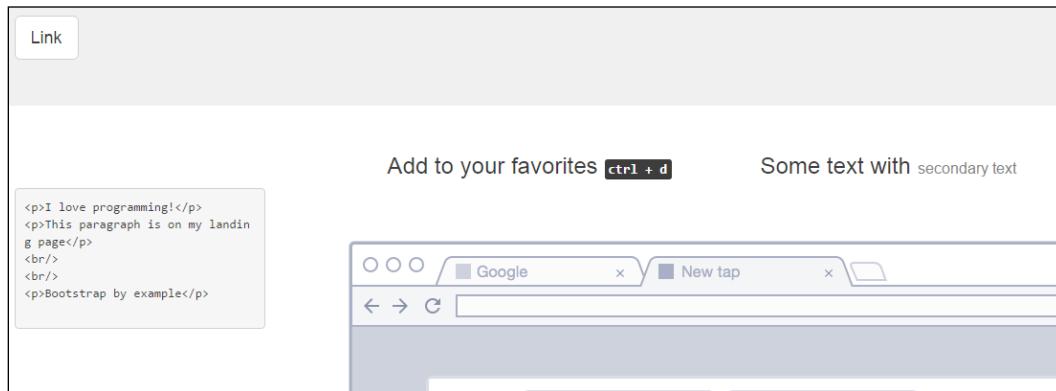
The grid prefix that we will be using is `.col-lg-*`. If you take a look at the landing page, you will see that everything is well placed and we don't need to make changes! However, we would like to make some tweaks to make our layout fancier and learn some stuffs of Bootstrap grid.

We want to touch upon column ordering. It is possible to change the order of column in the same row by applying the classes `.col-lg-push-*` and `.col-lg-pull-*` (note that we are using the large devices prefix, but any other grid class prefix can be used).

The `.col-lg-push-*` means that the column will be pushed to the right by the `*` columns, where `*` is the number of columns pushed. On the other hand, `.col-lg-pull-*` will pull the column in the left direction by `*`, where `*` is the number of columns as well. Let's test this trick in the second row by changing the order of both columns:

```
<div class="row">
  <div class="col-md-offset-4 col-md-4 col-sm-6 col-lg-push-4">
    <h3>
      Some text with <small>secondary text</small>
    </h3>
  </div>
  <div class="col-md-4 col-sm-6 col-lg-pull-4">
    <h3>
      Add to your favorites
    <small>
      <kbd class="nowrap"><kbd>ctrl</kbd> + <kbd>d</kbd></kbd>
    </small>
    </h3>
  </div>
</div>
```

We just added the class `.col-lg-push-4` to the first column and `.col-lg-pull-4` to the other one to get this result. By doing this, we changed the order of both columns in second row, as shown in the following screenshot:



Summary

We have completed another chapter. Here, we discussed why we should always go mobile-first if we want to make a web page for every viewport, from mobile to large desktop. Making things bigger is always easier and causes less issues, so start small with mobile devices and evolve the web page until it reaches large desktop resolutions.

We saw how to debug different devices using our browser and set the right classes for each viewport. We now have our example of a landing page with full stack responsiveness, working well in any device.

We covered the grid options for various devices resolutions using the mobile-first methodology – starting with mobile and going further until the large desktop version.

The main lesson of this chapter was that we always should go mobile-first. We did not follow this approach at first and because of that, we faced some problems that we could have eliminated if we had started mobile-first.

It was not mentioned before, but going mobile-first helps the whole team. The designer will have a bigger picture of what he or she needs to reach and what information is important from the beginning. The backend developer can focus on the main features and optimize them for mobile before moving on to the rest of the page content delivery. Mobile-first is also part of the development strategy.

At this point, we have our landing page fully set at all resolutions. Using Bootstrap, we took a shortcut towards responsivity, doing all the groundwork with a few lines of code in HTML and some more in CSS.

In the next chapter, we will apply some customizable styles to make the page a little less like a Bootstrap page. We will also see how to create landing pages for different uses by customizing the components.

4

Applying the Bootstrap Style

After making our landing page mobile-first and fully responsive for any device, it is time to go further into the Bootstrap framework, adding more and more components along with the style improvement.

This is the main objective of this chapter. We will take a step forward in terms of layout improvement, taking in regards the use of Bootstrap components. The chapter's key points are:

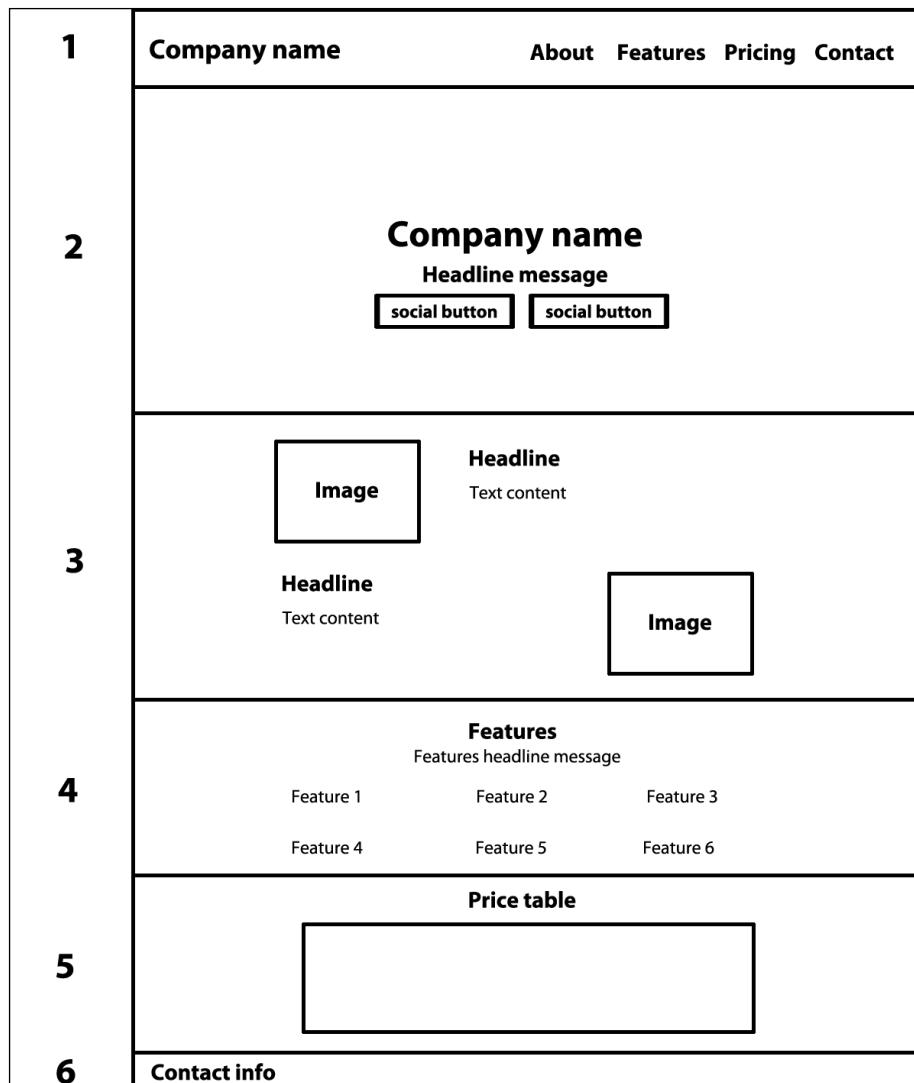
- Layout improvement
- Bootstrap forms
- Using images in Bootstrap
- Bootstrap helpers

By the end of this chapter, our landing page will almost be done, and you will be able to handle every HTML component customized by Bootstrap.

Changing our grid layout

First of all, the grid that we used for the current landing page is just a showcase of Bootstrap's potential and possibilities of customization. In this chapter, our objective is to make the grid fancier and more beautiful. To do this, we will change the grid to be like the one presented in the next figure.

We will go a little faster this time, since you already know how to create the grid using Bootstrap. Also, we will go mobile-first, as we discussed in the last chapter, but the screenshots will be taken from larger viewports just to improve the understandability.



Starting over the grid system

As you can see in the grid image, we split the grid into six parts. This time, each part will be a section that we will present step by step. If you are starting the example from scratch, don't forget to keep the boilerplate that we presented previously.

The header

So, we will start with the header. The code for representing the grid presented should be this one, to be placed right after the `<body>` tag:

```
<header>
  <div class="container">
    <!-- row 1 -->
    <div class="row">
      <a class="brand pull-left" href="#">Company name</a>
      <ul class="list-inline list-unstyled pull-right">
        <li><a href="#about">About</a></li>
        <li><a href="#features">Features</a></li>
        <li><a href="#pricing">Pricing</a></li>
        <li><a href="#contact">Contact</a></li>
      </ul>
    </div>
  </div>
</header>
```

As you can see, the `<header>` tag is wrapping all of our `.container`, making it similar to a section. Just for the note, to have the brand link placed on the left-hand side and the list on the right-hand side, we added the `.pull-left` and `.pull-right` classes to it, respectively. These are two Bootstrap helpers.

Now, let's modify our CSS to change the header style. Remember to import the custom CSS file at `<header>`:

```
<link rel="stylesheet" href="css/base.css">
```

For that part, we will change the background color and the alignment to a better placement of the link and list elements, so let's customize and override some styles from Bootstrap:

```
header {
  background-color: #F8F8F8;
}

header ul {
```

```
    margin: 0;  
}  
  
header a,  
header li {  
    padding: 1.4rem 0;  
    color: #777;  
    font-weight: bold;  
}
```

The header will look like what is shown in the following screenshot:



The introduction header

We have called the *introduction* header *section 2* of our grid. In this section, we have a big name of the company followed by the tagline and some buttons. The code for this row should be as follows:

```
<section id="intro-header">  
    <div class="container">  
        <!-- row 2 -->  
        <div class="row">  
            <div class="wrap-headline">  
                <h1 class="text-center">Company name</h1>  
                <h2 class="text-center">Tagline message</h2>  
                <hr>  
                <ul class="list-inline list-unstyled text-center">  
                    <li>  
                        <a class="btn btn-default btn-lg" href="#"  
role="button">Sign in</a>  
                    </li>  
                    <li>  
                        <a class="btn btn-primary btn-lg" href="#"  
role="button">Sign up</a>  
                    </li>  
                </ul>  
            </div>  
        </div>  
    </div>
```

So, we have wrapped the entire container in a section, just as we said we would. There is no secret here; we used `<h1>` for the company name and `<h2>` for the tagline. We placed the buttons in a centered list, just like the headlines, using the `.text-center` helper class, and the buttons are all set as before.

We will place a big image as the background for the `#intro-header` section. To do this, we edit the CSS file as follows:

```
section#intro-header {  
    background-image: url(..../imgs/landscape.jpg);  
    background-size: cover;  
}
```

The background set as `cover` will do the trick for us to make the image cover full width, although the size of the section is too small right now. For that, we will use our `.wrap-headline` element to do the trick and make it bigger:

```
section#intro-header .wrap-headline {  
    position: relative;  
    padding-top: 20%;  
    padding-bottom: 20%;  
}
```

As you may notice, we let a 20% padding at the top and bottom relative to our current position. With this, the height of the section becomes responsive to any viewport.

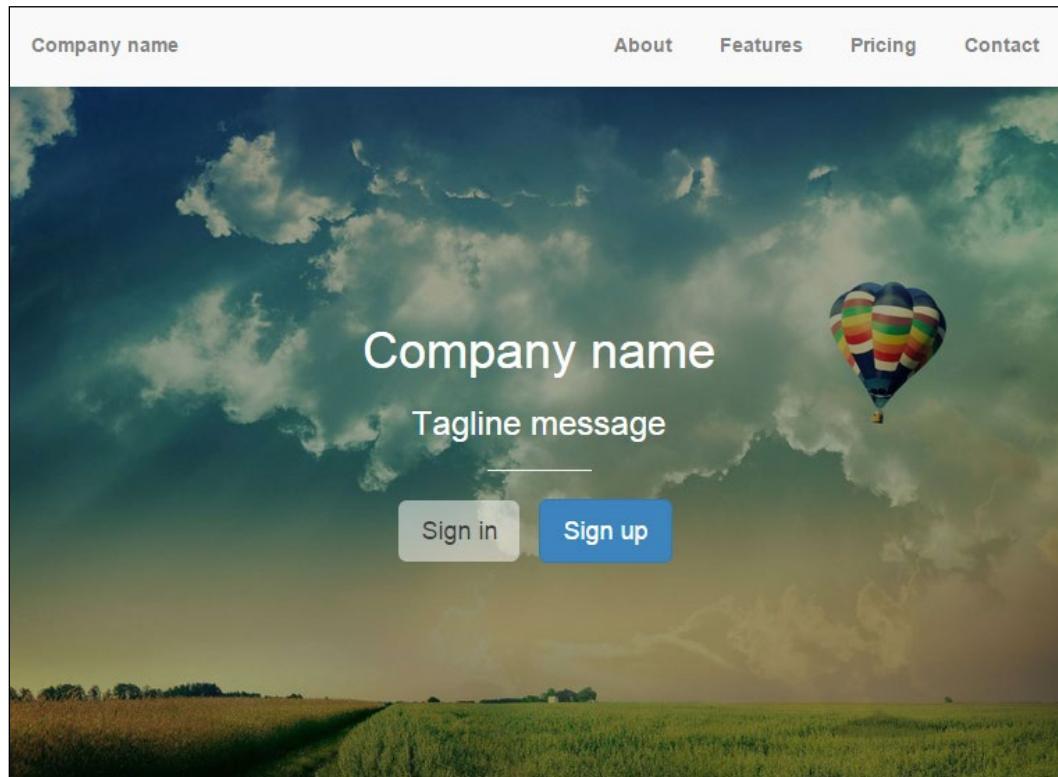
Moving on, we add some more CSS rules just for formatting, as follows:

```
section#intro-header {  
    background-image: url(..../imgs/landscape.jpg);  
    background-size: cover;  
}  
  
section#intro-header .wrap-headline {  
    position: relative;  
    padding-top: 20%;  
    padding-bottom: 20%;  
}  
  
section#intro-header h1,  
section#intro-header h2 {  
    color: #FFF;  
}
```

Applying the Bootstrap Style

```
section#intro-header h2 {  
    font-size: 1.5rem;  
}  
  
section#intro-header hr {  
    width: 10%;  
}  
  
section#intro-header .btn-default {  
    background-color: rgba(255, 255, 255, 0.5);  
    border: none;  
}
```

The final output of those two sections should be like the one shown in the following screenshot. Pretty fancy, isn't it?



The about section

So, for the *about* section, we will place a container that wraps all of the section as well. We will play with two rows equally divided, in which we will display an image and text alternated side by side. The code for this section should be as follows:

```
<section id="about">
  <div class="container">
    <!-- row 3 -->
    <div class="row">
      <div class="col-sm-6">
        
      </div>
      <div class="col-sm-6">
        <h3>Lorem ipsum dolor sit amet</h3>
        <p>
          Lorem ipsum dolor...
        </p>
      </div>
    </div>
    <hr>

    <!-- row 4 -->
    <div class="row">
      <div class="col-sm-6">
        <h3>Lorem ipsum dolor sit amet</h3>
        <p>
          Lorem ipsum dolor...
        </p>
      </div>
      <div class="col-sm-6">
        
      </div>
    </div>
  </div>
</section>
```

At this section, we just created two rows with two columns in each one. Since the columns are equally divided, they receive the `.col-sm-6` class. We added the `.img-responsive` class to the images to keep the ratio over the viewport and placed some text content on the image side of the column.

For the CSS, we add some rules to increase the margin between the content and the top portion of the page:

```
section#about img {  
    margin-top: 6.5rem;  
    margin-bottom: 5rem;  
}  
  
section#about h3 {  
    margin-top: 10rem;  
}
```

The following screenshot shows the resultant output of this section. Check whether the result of your code is similar to the following screenshot, and then let's move on to the *features* section:

Lorem ipsum dolor sit amet

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

Lorem ipsum dolor sit amet

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

The features section

The *features* section is composed of two lines of three columns, although we will create only one `.row` element and use the column wrapper technique. Do you remember it?

The column wrapper technique uses more than 12 parts of columns in a single row. The columns that overflow the `.row` will then be placed in the line below, creating the effect similar to having two `.row` elements:

```
<section id="features">
  <div class="container">

    <!-- row 5 -->
    <div class="row">
      <div class="col-sm-12">
        <h3 class="text-center">Features</h3>
        <p class="text-center">Features headline message</p>
      </div>
    </div>

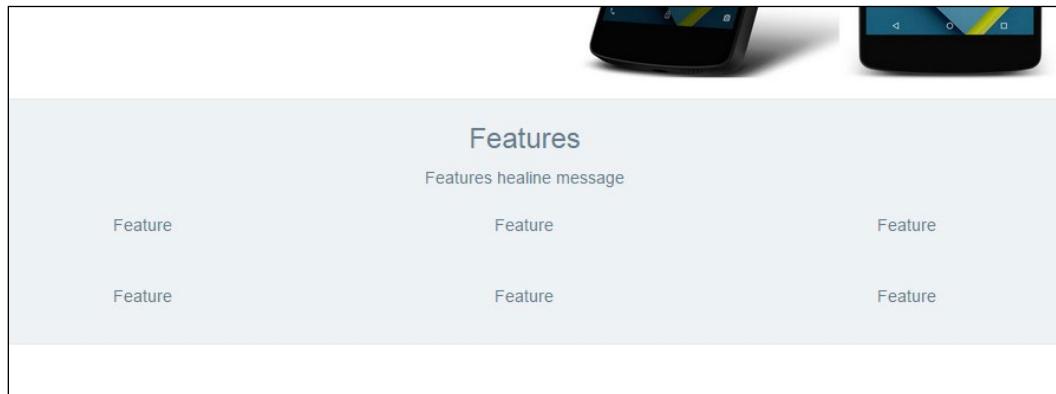
    <!-- row 6 -->
    <div class="row">
      <div class="col-sm-2 col-md-4">
        <div class="feature">Feature</div>
      </div>
      <div class="col-sm-2 col-md-4">
        <div class="feature">Feature</div>
      </div>
    </div>
  </div>
</section>
```

In this section, we created two rows. The first one holds the title and headline of the section with the `<h3>` and `<p>` tags, respectively. The second row is just composed of six equal columns with the `.col-sm-2` and `.col-md-4` classes. The use of `.col-sm-2` will place the `.feature` elements in a single line when using a small viewport.

Now, edit the CSS, and let's change the text color and add some padding between the features columns list:

```
section#features {  
    background-color: #eef2f5;  
    border-top: 0.1rem solid #e9e9e9;  
    border-bottom: 0.1rem solid #e9e9e9;  
}  
  
section#features * {  
    color: #657C8E;  
}  
  
section#features .feature {  
    padding-top: 2rem;  
    padding-bottom: 4rem;  
    text-align: center;  
}
```

The following screenshot presents the final output for the features section. Then it is time for us to start modifying the price table. It should be easy since we have already done the groundwork for it.



The price table section

For the *price table* section, we will use the same table from the *Manipulating tables* section in *Chapter 2, Creating a Solid Scaffolding*, but with a few modifications to make it prettier. We will make some small changes, as presented in the following code:

```
<section id="pricing">
  <div class="container">

    <!-- row 7 -->
    <div class="row">
      <div class="col-sm-12">
        <h3 class="text-center price-headline">Price table</h3>
      </div>
    </div>

    <!-- row 8 -->
    <div class="row">
      <div class="col-sm-10 col-sm-offset-1">
        <table class="table table-striped table-hover">
          <thead>
            <tr>
              <th class="success">
                <h4 class="text-center white-text">Free plan</h4>
              </th>
              <th class="info">
                <h4 class="text-center white-text">Standard
plan</h4>
              </th>
              <th class="danger">
                <h4 class="text-center white-text">Premium
plan</h4>
              </th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td class="success">
                <h3 class="text-center white-text">$ 0</h3>
              </td>
              <td class="info">
                <h3 class="text-center white-text">$ 99</h3>
              </td>
            </tr>
          </tbody>
        </table>
      </div>
    </div>
```

```
<td class="danger">
    <h3 class="text-center white-text">$ 999</h3>
</td>
</tr>
<tr>
    <td>Lorem ipsum</td>
    <td>Lorem ipsum</td>
    <td>Lorem ipsum</td>
</tr>
<tr>
    <td>Lorem ipsum</td>
    <td>Lorem ipsum</td>
    <td>Lorem ipsum</td>
</tr>
<tr>
    <td>Dolor sit amet</td>
    <td>Lorem ipsum</td>
    <td>Lorem ipsum</td>
</tr>
<tr>
    <td>-</td>
    <td>Dolor sit amet</td>
    <td>Lorem ipsum</td>
</tr>
<tr>
    <td>-</td>
    <td>-</td>
    <td>Lorem ipsum</td>
</tr>
<tr>
    <td><a href="#" class="btn btn-success btn-block">Purchase</a></td>
    <td><a href="#" class="btn btn-info btn-block">Purchase</a></td>
    <td><a href="#" class="btn btn-danger btn-block">Purchase</a></td>
</tr>
</tbody>
</table>
</div>
</div>
</div>
</section>
```

The first change is that we added a header, `<h3>`, in this section in the first row. Furthermore, we added the `.success`, `.info`, and `.danger` classes to the first `<tr>` in `<tbody>` (they are highlighted in bold).

Also in `<table>`, we removed the `.table-bordered` class to take out the border from it. Finally, we changed some colors in the CSS file and created the `.white-text` class, which is highlighted in the code as well:

```
section#pricing h3.price-headline {  
    margin-top: 5rem;  
    margin-bottom: 3rem;  
}  
  
section#pricing .white-text {  
    color: #FFF;  
}  
  
section#pricing thead .success {  
    background-color: #78CFBF;  
}  
  
section#pricing thead .info {  
    background-color: #3EC6E0;  
}  
  
section#pricing thead .danger {  
    background-color: #E3536C;  
}  
  
section#pricing tbody .success {  
    background-color: #82DACA;  
}  
  
section#pricing tbody .info {  
    background-color: #53CFE9;  
}  
  
section#pricing tbody .danger {  
    background-color: #EB6379;  
}
```

The following screenshot presents the result of the price table. Finally, to sum it up, we will advance to the footer, which contains the contact information:

The footer

For the footer, we will have five columns, the first one being the logo with a `.col-sm-2`. This will be followed by three info columns, each one with a `.col-sm-2` as well. The last column is the address column, with the `.col-sm-4` class. The HTML code is as follows:

```
<footer>
  <div class="container">
    <div class="col-sm-2">
      
    </div>
    <div class="col-sm-2">
      <h5>The company</h5>
      <ul class="list-unstyled">
        <li><a href="#">Documentation</a></li>
        <li><a href="#">Packt publisher</a></li>
        <li><a href="#">About us</a></li>
        <li><a href="#">Contact</a></li>
      </ul>
    </div>
```

```
<div class="col-sm-2">
  <h5>Social</h5>
  <ul class="list-unstyled">
    <li><a href="#">Facebook</a></li>
    <li><a href="#">Twitter</a></li>
    <li><a href="#">Blog</a></li>
  </ul>
</div>
<div class="col-sm-2">
  <h5>Support</h5>
  <ul class="list-unstyled">
    <li><a href="#">Contact</a></li>
    <li><a href="#">Privacy policy</a></li>
    <li><a href="#">Terms & conditions</a></li>
    <li><a href="#">Help desk</a></li>
  </ul>
</div>
<div class="col-sm-4">
  <address>
    <strong>Name, Inc.</strong>
    Address line 1<br>
    Address line 2<br>
    <abbr title="Phone">P:</abbr> (123) 456-7890
  </address>
</div>
</div>
</footer>
```

Now, let's prettify the footer with some CSS rules:

```
footer {
  background-color: #191919;
  color: #ADADAD;
  margin-top: 3em;
}

footer h5,
footer img {
  margin-top: 5em;
  font-weight: bold;
}

footer address {
  margin-top: 5em;
```

```
margin-bottom: 5em;
color: #5A5A5A;
}

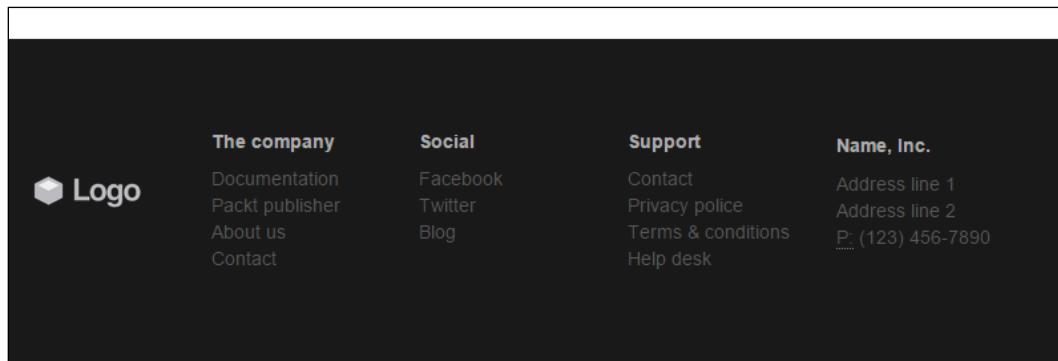
footer ul {
margin-bottom: 5em;
}

footer address strong {
color: #ADADAD;
display: block;
padding-bottom: 0.62em;
}

footer a {
font-weight: 300;
color: #5A5A5A;
}

footer a:hover {
text-decoration: none;
color: #FFF;
}
```

So, we basically changed the background to a shaded one, added some margins to make the footer larger, and modified the links' colors. And we are done with the new layout! See in the following screenshot how the final layout for the footer looks:



Resize the viewport and you will see how the page correctly adapts to any kind of resolution. So we have made the page again, this time with the mobile-first perspective in mind, adding more content and using Bootstrap as our backup. Nicely done!

Forming the forms

The Web would not be the same without forms. They are one of the major methods of interacting with a web page and sending data to consolidate. Since the beginning of the Web, the style and rendering of forms were a source of trouble, because they were displayed differently for each browser and there were placement problems.

This is one of the reasons Bootstrap appeared to make all web pages follow the same rendering pattern for any browser and device. For forms, this is no different. There are styles for almost every kind of element. We will start talking about forms in this section, although we will keep discussing them in later chapters as well, since they are an important element in frontend web development.

Newsletter form

To start easy, we will use an inline form in our landing page. Let's add a new row between the price table row and the footer with the following HTML code:

```
<section id="newsletter" class="text-center">
  <h4>Stay connected with us. Join the newsletter to receive fresh
  info.</h4>
  <form class="form-inline" method="POST">
    <div class="form-group">
      <input class="form-control" placeholder="Your name">
    </div>
    <div class="form-group">
      <input class="form-control" placeholder="Your email">
    </div>
    <button type="submit" class="btn btn-primary">Join
    now!</button>
  </form>
</section>
```

OK, we're starting to break down every part of the code. The first part—we just created a new `<section>` and centralized it using the `.text-center` class from the Bootstrap helpers.

The first form type that you will learn about is `.form-inline`, which simply makes all the controls inside it the inline-block kind. Because of that, we are able to centralize the form using the `.text-center` helper. Also, this form will make the controls inline until the small viewport, when it changes the controls to become blocks, each one filling one line.

Inside `.form-inline`, we have two `div.form-group`. Every element inside a `<form>` that contains the `.form-group` class will automatically be displayed as a block element. In almost every form element, we will have a `.form-group`, since it is an important wrapper for labels and controls for the purpose of optimizing spacing in Bootstrap.

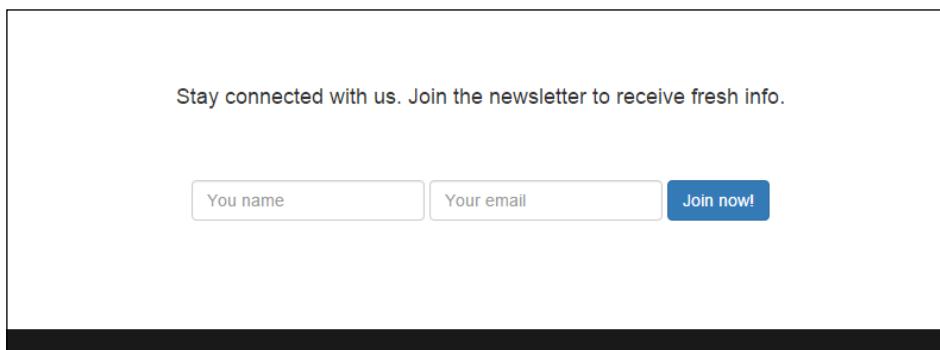
In our case, since we set the form to be the inline kind (because of the `.form-inline` class), the `.form-group` elements will be inline elements as well.

The two `<input>` are not magical; just place it in your code as shown. The same applies to the button, by using the `.btn-primary` class to make it blue.

The CSS for this section is quite simple. We have just made some tweaks for better rendering:

```
section#newsletter {  
    border-top: 1px solid #E0E0E0;  
    padding-top: 3.2em;  
    margin-top: 2em;  
}  
  
section#newsletter h4 {  
    padding: 1em;  
}  
  
section#newsletter form {  
    padding: 1em;  
    margin-top: 2em;  
    margin-bottom: 5.5em;  
}
```

Our first form is complete! The following screenshot shows the final output of the form:



This one was the simplest form. Now let's crack some other forms to nail it in Bootstrap.

Contact form

To make a *contact* form, we need to create another HTML file. Name it `contact.html` and use the same header and footer that you used earlier in the landing page. The final output is shown in the next image. Let's break down each part of the form to achieve the final result:

The form consists of the following elements:

- Name:** Input field placeholder: Full name
- Email:** Input field placeholder: Contact email
- Message:** Text area placeholder: Type your message
- Subscription:** A checkbox labeled "I want to subscribe to receive updates from the company."
- Submit:** A green button labeled "Submit".

First of all, we need to create the grid for this form. As you can see, the form is in the center of the page, so to do that, create this HTML code:

```
<section id="contact" class="container">
  <div class="row">
    <div class="col-sm-offset-2 col-sm-8">
      ...
    </div>
  </div>
</section>
```

We just created the grid for this container. Inside this column, we need to create a `<form>` element with the following code:

```
<form class="form-horizontal">
  <div class="form-group">
```

```
<label class="col-sm-2 control-label" for="contact-name">Name</label>
<div class="col-sm-10">
  <input class="form-control" type="text" id="contact-name"
placeholder="Full name">
</div>
</div>
<div class="form-group">
  <label class="col-sm-2 control-label" for="contact-email">Email</label>
  <div class="col-sm-10">
    <input class="form-control" type="text" id="contact-email"
placeholder="Contact email">
  </div>
</div>
<div class="form-group">
  <label class="col-sm-2 control-label" for="contact-email">Message</label>
  <div class="col-sm-10">
    <textarea class="form-control" rows="3" placeholder="Type
your message"></textarea>
  </div>
</div>
<div class="form-group">
  <div class="col-sm-offset-2 col-sm-10">
    <label class="checkbox">
      <input type="checkbox" value="">
      I want to subscribe to receive updates from the company.
    </label>
  </div>
</div>
<div class="form-group">
  <div class="col-sm-offset-2 col-sm-10">
    <button class="btn btn-success btn-lg"
type="submit">Submit</button>
  </div>
</div>
</form>
```

At first sight, it looks like a common form, with two input fields, a text area, a checkbox, and a *submit* button. The `.form-horizontal` class is responsible for aligning the labels and the inputs side by side horizontally. Note that we are using the `.col-sm-*` grid classes in both labels and inputs in a grid of 12 parts inside `.form-group`, just like the column nesting technique.

In the `.form-group` checkbox, we created a `<div>` with an offset of 2 to fill the part that is not needed in this case. Note that we are able to use the same grid classes to acquire the same results inside forms. To place the Bootstrap theme in the checkbox, just add the `.checkbox` class to the label wrapping the input.

We don't need much CSS in this section; just add some padding to give some space to the form:

```
section#contact form {
    padding-top: 9rem;
    padding-bottom: 3rem;
}
```

Let's start with some JavaScript

It's time to start playing with some JavaScript! Create a file named `main.js` inside the `js` folder, which also contains the Bootstrap JavaScript file and jQuery library. To do the groundwork in the JavaScript file, we need to load it after `document` is ready:

```
$(document).ready(function() {
    // document ready, place you code
});
```

Then, we will validate the form before sending it. To do that, attach an event handler to the form submission, like this:

```
$(document).ready(function() {
    $('#contact form').on('submit', function(e) {
        e.preventDefault();
    });
});
```

You may know this, but the `e.preventDefault()` code line is a method that prevents the default action from being triggered, the form submission in this case.

Moving on, we create the variables that we will use and the validation code:

```
$(document).ready(function() {
    $('#contact form').on('submit', function(e) {
        e.preventDefault();
        var $form = $(e.currentTarget),
            $email = $form.find('#contact-email'),
            $button = $form.find('button[type=submit]');

        if ($email.val().indexOf('@') == -1) {
            vaca = $email.closest('form-group')
```

```
        $email.closest('.form-group').addClass('has-error');
    } else {
        $form.find('.form-group').addClass('has-success').
removeClass('has-error');
        $button.attr('disabled', 'disabled');
        $button.after('<span>Message sent. We will contact you
soon.</span>');
    }
});
```

So, we first created our variables for the form, the `email` field, and the `button` element. After that, we performed a naïve validation on the `email` field, where if the `@` character is present in the field, it is valid. If it is not present, we add the `.has-error` class to the parent `.form-group` of the field. It will produce the elements inside the form group in red, as presented in the following screenshot:

The screenshot shows a contact form with three fields: Name, Email, and Message. The Name field contains 'Full name' and is grayed out. The Email field contains 'notpresent@email.com' and has a red border, indicating an error. The Message field contains 'Type your message' and is grayed out. Below the fields is a checkbox labeled 'I want to subscribe to receive updates from the company.' followed by a green 'Submit' button.

Load the JavaScript file in the HTML of `contact.html` just below where `bootstrap.js` loads:

```
<script src="js/bootstrap.js"></script>
<script src="js/main.js"></script>
```

If the `@` sign is present in the field, we simply pass the validation by fake-sending it. When this happens, we add the `.has-success` class to each `.form-group`, making them green. We also add the attribute `disabled` to the button, changing its behavior and theme as Bootstrap does it.

Finally, we add after the button a simple feedback message for the user, saying that the contact message was sent. The following screenshot shows the case where the contact message is successfully sent:

The screenshot shows a contact form with three input fields: Name (full name), Email (present@email.com), and Message (test message). Below the fields is a checkbox labeled "I want to subscribe to receive updates from the company." A green "Submit" button is present. To the right of the button, a message says "Message sent. We will contact you soon." The entire form is enclosed in a light gray border.

The sign-in form

Now that you have learned some more form styles in the contact file, we will play with another kind of form: the sign-in form.

Go back to the landing page HTML file, and in the sign in .btn located inside the introduction header, add the #signin-btn identifier:

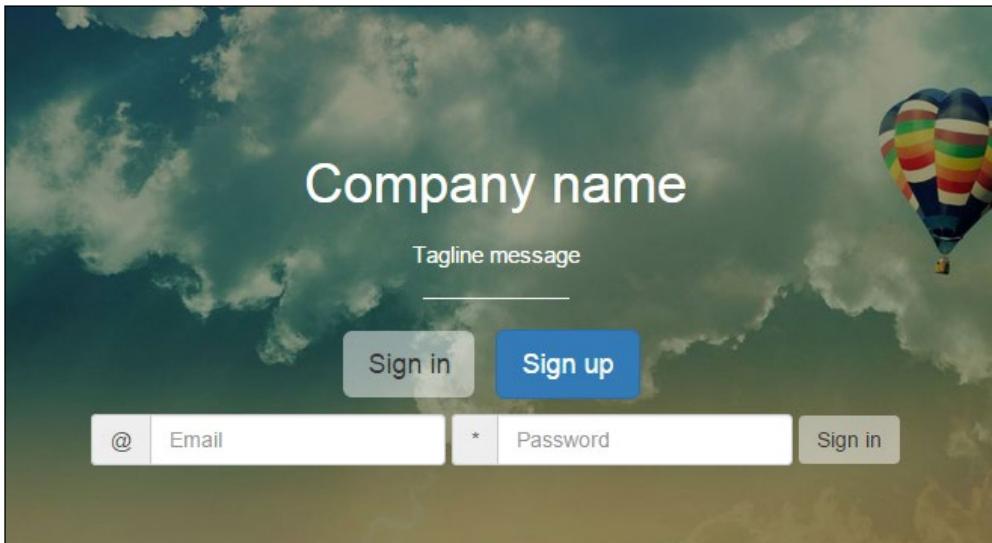
```
<a id="signin-btn" class="btn btn-default btn-lg" href="#" role="button">Sign in</a>
```

After the `` that wraps the sign buttons, place the sign-in form code:

```
<form id="signin" class="form-inline text-center hidden-element">
  <div class="form-group">
    <div class="input-group">
      <div class="input-group-addon">@</div>
      <input type="text" class="form-control" id="signin-email" placeholder="Email">
    </div>
  </div>
  <div class="form-group">
    <div class="input-group">
      <div class="input-group-addon">*</div>
```

```
<input type="password" class="form-control" id="signin-password"
placeholder="Password">
</div>
</div>
<button type="submit" class="btn btn-default">Sign in</button>
</form>
```

The result should be like what is shown in the following screenshot, where the new form appears after the buttons:



Before moving on to fixing the layout, let's explain `.input-group`. Bootstrap offers this option to prepend or append things to an input using `.input-group-addon`. In this case, we prepend @ and * to each input. We could also have appended this to the inputs by placing `.input-group-addon` after the input instead of before.

For the CSS, we just added the `.hidden-element` rule. We could not use the `.hidden` Bootstrap helper because it applies the `!important` option, and we would not have been able to make it visible again without removing the class:

```
.hidden-element {
  display: none;
}
```

Let's animate it a little! Go to the JavaScript file and add the event listener to the click on the sign-in button:

```
$(document).ready(function() {
    ... // rest of the JavaScript code
    $('#sign-btn').on('click', function(e) {
        $(e.currentTarget).closest('ul').hide();
        $('form#signin').fadeIn('fast');
    });
});
```

By doing this, we hide the `` element that contains the *sign* buttons and show the sign in form. That was just the cherry on our pie, and we are done with forms by now! Refresh the web page in the browser, click on the **Sign in** button, and see the new form appearing. Moving forward, we will use some images and see how Bootstrap can help us with that.

Images

For images, Bootstrap offers some classes to make your day better. We have already discussed the use of the `.img-responsive` class, on which the image becomes scalable by setting `max-width: 100%` and `height: auto`.

The framework also offers three convenient classes to style your image. To make use of that, place the following code after the price table in the landing:

```
<section id="team">
    <div class="container">
        <div class="row">
            <div class="col-sm-12">
                <ul class="list-inline list-unstyled text-center">
                    <li>
                        
                        <h5>Jonny Doo</h5>
                        <p>CEO</p>
                    </li>
                    <li>
                        
                        <h5>Jonny Doo</h5>
                        <p>CTO</p>
                    </li>
                    <li>
```

```

<h5>Jonny Doo</h5>
<p>CEO</p>
</li>
</ul>
</div>
</div>
</section>
```

As you can notice, we simply created another container and row with a single column, .col-sm-12. Inside the column, we added an inline list with the elements, each one having one image with a different class. The .img-rounded class makes the corners rounded, .img-circle turns the image into a circular shape, and .img-thumbnail adds a nice rounded border to the image, like this:



The preceding screenshot shows how this section is displayed. We also had to add some CSS code to increase margins and paddings, along with font customization:

```
section#team ul {
    margin: 5rem 0;
}

section#team li {
    margin: 0 5rem;
}

section#team h5 {
    font-size: 1.5rem;
    font-weight: bold;
}
```

So, it's nice to have a backup of Bootstrap, even with the images, making our work easier and pacing up the development. By the way, Bootstraps offers tons of helpers with the same objective. We have already used some of them; now let's use even more.

Helpers

Helpers are Bootstrap classes that help us achieve certain customizations. They are planned to offer a single purpose and reduce CSS frequency of repeated rules. The goal is always the same: increase the pace of development.

Floating and centering blocks

We have talked previously about the `.pull-left` and `.pull-right` classes, which make the HTML element float to the left or right. To center the block, you can use the `.center-block` class.

To make use of this, go to the column that wraps the pricing table, and replace the `col-sm-10 col-sm-offset-1` classes with `.center-block`. In the CSS, add the following rule:

```
section#pricing .center-block {  
    width: 90%;  
}
```

Refresh the web page and you will see that the table stays centered, but now using a different approach.

Context colors

You can apply the same colors that we used in buttons and the price table to every element in the page. To do that, use these classes: `.text-primary`, `.text-success`, `.text-warning`, `.text-info`, `.text-danger`, and `.text-muted`.

In the images section that we have just made, apply the `.text-info` class to the `<h5>` elements and apply `.text-muted` in `<p>`:

```
<section id="team">  
    <div class="container">  
        <div class="row">  
            <ul class="list-inline list-unstyled text-center">  
                <li>  
                      
                    <h5 class="text-info">Jonny Doo</h5>
```

```
<p class="text-muted">CEO</p>
</li>
<li>
  
  <h5 class="text-info">Jonny Doo</h5>
  <p class="text-muted">CTO</p>
</li>
<li>
  
  <h5 class="text-info">Jonny Doo</h5>
  <p class="text-muted">CIO</p>
</li>
</ul>
</div>
</section>
```

Refresh the web page, and the headline element will be light blue and the paragraph text grey.

To make the opposite operation—changing the background to the context color—apply the `.bg-*` class, where you can pass one of the color options (primary, info, warning, or danger).

Spacing

In Bootstrap 4, they added new helpers for margins and padding spacing. If you are using Sass, you can set a default \$spacer and every margin will work like a charm by using these classes, although the default value for the spacer is `1rem`.

Next, we will present a table with the classes for margin usage. In summary, you will use the `.m-*-*` regex, where the first option is the location, such as top, bottom, and so on. The second option is the size of the margin. Refer to this table for a better understanding of the usage:

	Remove margin	Default margin	Medium margin (1.5 times)	Large margin (3 times)
All	<code>.m-a-0</code>	<code>.m-a</code>	<code>.m-a-md</code>	<code>.m-a-lg</code>
Top	<code>.m-t-0</code>	<code>.m-t</code>	<code>.m-t-md</code>	<code>.m-t-lg</code>
Right	<code>.m-r-0</code>	<code>.m-r</code>	<code>.m-r-md</code>	<code>.m-r-lg</code>
Bottom	<code>.m-b-0</code>	<code>.m-b</code>	<code>.m-b-md</code>	<code>.m-b-lg</code>
Left	<code>.m-l-0</code>	<code>.m-l</code>	<code>.m-l-md</code>	<code>.m-l-lg</code>
Horizontal	<code>.m-x-0</code>	<code>.m-x</code>	<code>.m-x-md</code>	<code>.m-x-lg</code>
Vertical	<code>.m-y-0</code>	<code>.m-y</code>	<code>.m-y-md</code>	<code>.m-y-lg</code>

For the padding, the classes are almost the same; just use the `.p-*-*` prefix to get the expected result. Remember that the default spacer is `1rem`, so the medium is `1.5rem` and large is `3rem`:

	Remove margin	Default padding	Medium padding (1.5 times)	Large padding (3 times)
All	<code>.p-a-0</code>	<code>.p-a</code>	<code>.p-a-md</code>	<code>.p-a-lg</code>
Top	<code>.p-t-0</code>	<code>.p-t</code>	<code>.p-t-md</code>	<code>.p-t-lg</code>
Right	<code>.p-r-0</code>	<code>.p-r</code>	<code>.p-r-md</code>	<code>.p-r-lg</code>
Bottom	<code>.p-b-0</code>	<code>.p-b</code>	<code>.p-b-md</code>	<code>.p-b-lg</code>
Left	<code>.p-l-0</code>	<code>.p-l</code>	<code>.p-l-md</code>	<code>.p-l-lg</code>
Horizontal	<code>.p-x-0</code>	<code>.p-x</code>	<code>.p-x-md</code>	<code>.p-x-lg</code>
Vertical	<code>.p-y-0</code>	<code>.p-y</code>	<code>.p-y-md</code>	<code>.p-y-lg</code>

Responsive embeds

The new version of Bootstrap 4 also allows us to make embeds responsive. So, there are classes for the `<iframe>`, `<embed>`, `<video>`, and `<object>` elements. To get the expected result, add the `.embed-responsive` class to your element:

```
<div class="embed-responsive embed-responsive-16by9">
  <iframe class="embed-responsive-item"
  src="//www.youtube.com/embed/dQw4w9WgXcQ"
  allowfullscreen></iframe>
</div>
```

We added the `.embed-responsive-16by9` class to make the aspect ratio of the video 16:9. You can also use the aspect ratios 21:9 and 4:3 with the `.embed-responsive-21by9` and `.embed-responsive-4by3` classes respectively.

Summary

In this chapter, we remade our landing page by applying the Bootstrap theme and customizing it, getting a much better result in the end. Right now, we have a clean web page, developed quickly using the mobile-first paradigm.

You also started to learn the use of some forms by going through three examples, one of these being a complementary contact page. Along with forms, we started using JavaScript! We performed form validation and some simple animations on our page, with regard to the template.

Finally, we presented the Bootstrap image options and a bunch of helpers. Remember that there are more helpers than the ones shown in this chapter, but don't worry, because we will see them in the upcoming chapters.

If you think you already have a fancy landing page, we will prove to you that we can improve it even more! We will talk about it again in the next chapter, reaching icons, more forms, buttons, and navigation bars.

Congratulations! You have reached this point of the book. Brace yourself, because the next level is coming. We will take a step forward by using more complex elements and components of Bootstrap.

5

Making It Fancy

It is finally time to take our last step through the landing page example. After learning all the basics of Bootstrap, passing from side to side of the grid system, mobile-first development, and using Bootstrap HTML elements, the landing page example has come to an end. Now it is time to go a little deeper and acquire more knowledge of this beautiful framework—Bootstrap.

In this chapter, we will focus on adding components all over the landing page. We will also touch upon the flexbox option, present in version 4. After all has been said, our landing page will be ready for the production stage. Get ready for the key points that we will cover in this chapter:

- Glyphicon icons
- Navigation bars
- The Drop-down component
- Input grouping
- Flexbox Bootstrap usage

Using Bootstrap icons

Bootstrap is such a nice thing! It provides for us more than 250 icons ready for use and fully resizable. The icons were created from the Glyphicon Halflings set (<http://glyphicons.com/>). They are fully rendered as fonts, so you can customize both size and color for each one of them. To make use of that, let's see the features section on the landing page. As you can see, we let this section be a little simpler. By adding some fonts, we will get a nicer result:

```
<section id="features">
  <div class="container">
```

```
<!-- row 5 -->
<div class="row">
    <div class="col-sm-12">
        <h3 class="text-center">Features</h3>
        <p class="text-center">Features headline message</p>
    </div>
</div>

<!-- row 6 -->
<div class="row">
    <div class="col-sm-2 col-md-4">
        <div class="feature">
            <span class="glyphicon glyphicon-screenshot" aria-hidden="true"></span>
            <span class="feature-tag">Product focus</span>
        </div>
    </div>
    <div class="col-sm-2 col-md-4">
        <div class="feature">
            <span class="glyphicon glyphicon-education" aria-hidden="true"></span>
            <span class="feature-tag">Teaching as a passion</span>
        </div>
    </div>
    <div class="col-sm-2 col-md-4">
        <div class="feature">
            <span class="glyphicon glyphicon-send" aria-hidden="true"></span>
            <span class="feature-tag">Spreading knowledge</span>
        </div>
    </div>
    <div class="col-sm-2 col-md-4">
        <div class="feature">
            <span class="glyphicon glyphicon-hourglass" aria-hidden="true"></span>
            <span class="feature-tag">Save your day time</span>
        </div>
    </div>
    <div class="col-sm-2 col-md-4">
        <div class="feature">
            <span class="glyphicon glyphicon-sunglasses" aria-hidden="true"></span>
            <span class="feature-tag">Make it fancy</span>
        </div>
    </div>
</div>
```

```

<div class="col-sm-2 col-md-4">
    <div class="feature">
        <span class="glyphicon glyphicon-heart" aria-
hidden="true"></span>
            <span class="feature-tag">Made with love</span>
        </div>
    </div>
    </div>
</div>
</section>

```

So, from the beginning, here is the code of the modified features section. The bold text corresponds to the icon additions. It is pretty simple to add an icon. Just check out the options at <http://getbootstrap.com/components/#glyphicons>, copy the class code, and use it in an element. Note that you must add both classes, .glyphicon and .glyphicon-*.

The aria-hidden property

You may have noticed that there is a property called `aria-hidden="true"` present in all the icons. The reason for this is that the fonts are represented as Unicode characters, meaning they may represent words. Therefore, to prevent that accessibility, screen readers start reading those characters such as they are words, the `aria-hidden` attribute prevents that.

Even more, we made some changes to the CSS file, adding more rules for the current working section. Add the following style to the `base.css` file, located in the `css` folder:

```

section#features .feature {
    padding-top: 2rem;
    padding-bottom: 4rem;
    text-align: center;
}

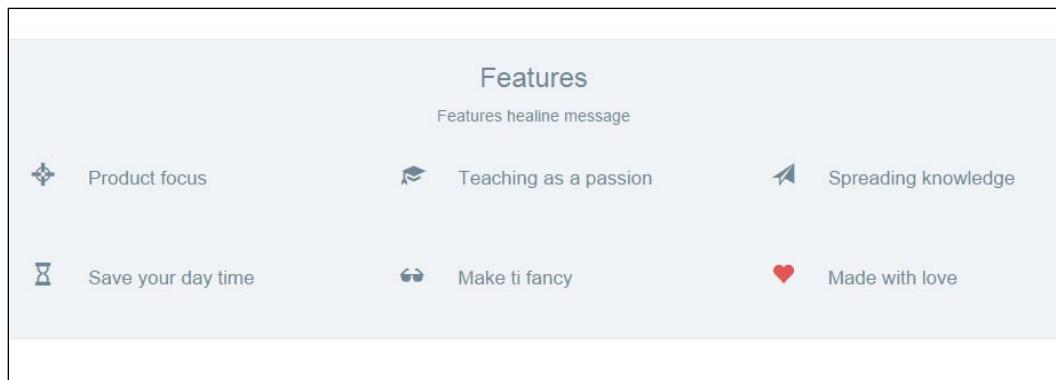
section#features .glyphicon {
    font-size: 2rem;
}

section#features .glyphicon-heart {
    color: #E04C4C;
}

```

```
section#features .feature-tag {  
    max-width: 10.7em;  
    display: inline-block;  
    text-align: left;  
    margin-left: 1.5em;  
    font-size: 1.7rem;  
}
```

With this, we want to show some nice options that you can use with icons. The first one is that you can change the size of the icon by changing its font size. In our case, we set it to `font-size: 2rem`. The second one is that icons provide the option to change their color by just adding the CSS color rule. We applied it to the heart icon, because the heart must be red – `color: #E04C4C`.



The preceding screenshot shows the final result of the **Features** section. As you can see, it is pretty simple to use icons in Bootstrap. Also, the possibilities that the framework offers are very suitable for daily adjustments, such changing icons' colors and sizes.



Using other icons sets

There are plenty of other icon sets out there that can be used just like glyphicons for Bootstrap. Among all of them, it is worth mentioning Font Awesome (<https://fontawesome.github.io/Font-Awesome/>). It stands out from others, since it was the first icon set to use font encoding, together with a wide variety of icons.

Paying attention to your navigation

Bootstrap offers a very nice navigation bar to be placed at the top of website, or even in places where you want that behavior. Let's change our header section to make it our navigation bar. It will stick to the top of the web page, working as a navigation menu.

First of all, let's use a `<nav>` element and add to it the `.navbar` and `.navbar-default` classes, which are required for the component, and the `.navbar-fixed-top` class to fix the element at the top. Replace the `<header>` HTML section with the following code:

```
<nav class="navbar navbar-default navbar-fixed-top">
  <div class="navbar-header">
    <a class="navbar-brand" href="landing_page.html">Company name</a>
  </div>
  <div class="navbar-right">
    <ul class="nav navbar-nav">
      <li><a href="#about">About</a></li>
      <li><a href="#features">Features</a></li>
      <li><a href="#pricing">Pricing</a></li>
      <li><a href="contact.html">Contact</a></li>
    </ul>
  </div>
</nav>
```

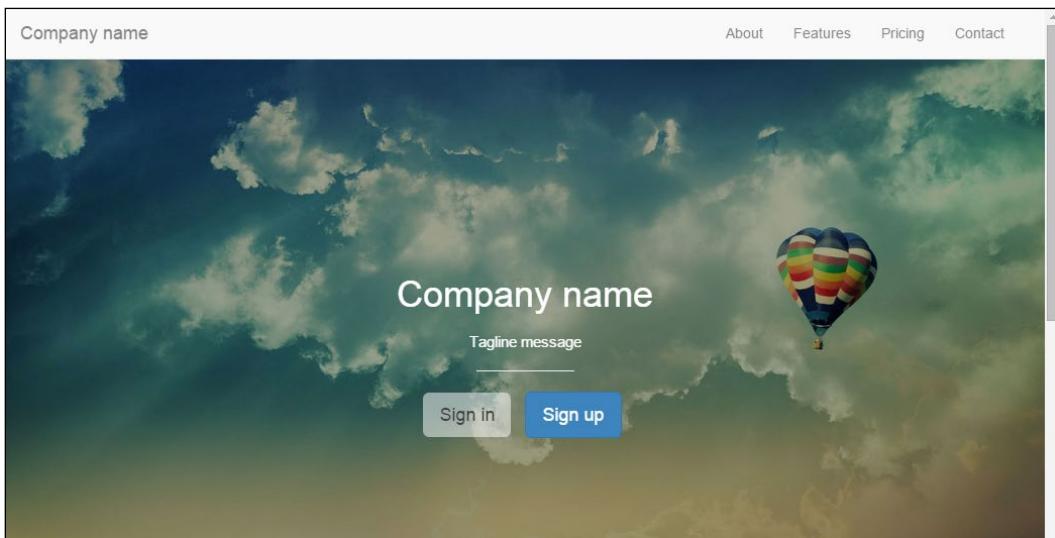
As was mentioned, the `.navbar` and `.navbar-default` classes are required for the navigation component. For the **Company name** link, we added a class, `.navbar-brand`, which has the purpose of branding the heading with an appropriate font size and padding.

Then, we created a `<div>` tag using the `.navbar-right` class to provide a set padding CSS rules and place the list to the right to appear the same way as was before. For the CSS, just add the following rule to create a padding to the `<body>` of your page:

```
body {
  padding-top: 3.6em;
}

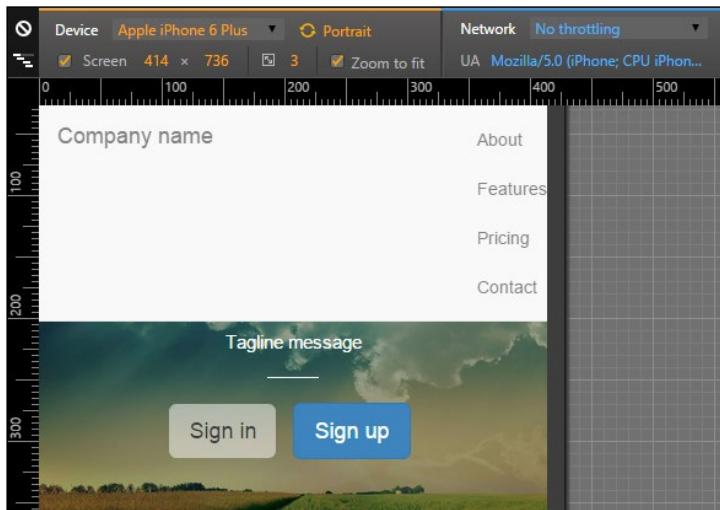
#nav-menu {
  margin-right: 1rem;
}
```

The result of the navigation bar should be like what is presented in the following screenshot:



Until the navigation collapse

Try to resize the web page and you will see that for small viewports, the horizontal list placed in the navigation bar will stack vertically, as illustrated in the next screenshot. Fortunately, Bootstrap has the option to collapse the lists at the navigation bar. The procedure for doing this is pretty simple and we will do it now.



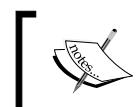
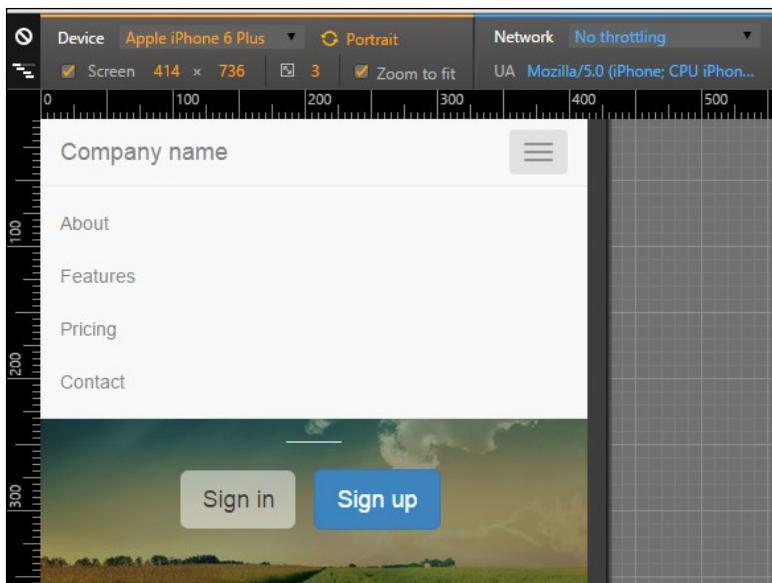
So, let's make the .nav-header collapses and create a toggle button to show or hide the list menu. Change the HTML to this:

```
<nav class="navbar navbar-default navbar-fixed-top">
  <div class="navbar-header">
    <a class="navbar-brand" href="landing_page.html">Company
    name</a>
    <button type="button" class="navbar-toggle collapsed" data-
    toggle="collapse" data-target="#nav-menu" aria-expanded="false">
      <span class="sr-only">Toggle navigation</span>
      <span class="icon-bar"></span>
      <span class="icon-bar"></span>
      <span class="icon-bar"></span>
    </button>
  </div>

  <div id="nav-menu" class="collapse navbar-collapse navbar-right">
    <ul class="nav navbar-nav">
      <li><a href="#about">About</a></li>
      <li><a href="#features">Features</a></li>
      <li><a href="#pricing">Pricing</a></li>
      <li><a href="contact.html">Contact</a></li>
    </ul>
  </div>
</nav>
```

The new code is in bold. First, we added a `<button>` element to create our *sandwich button* (the one with three dashes). On `data-target`, we must add the element that is the collapse target, `#nav-menu` in our case.

Then, we must say which is the element to be collapsed in the small viewport, so for `.navbar-right`, we added the `.collapse navbar-collapse` classes. The navigation bar should then appear like the one shown in the following screenshot. Hooray! Bootstrap has saved the day again!



In order to use `.navbar-collapse`, you should remember to load the Bootstrap JavaScript library as well.

Using different attachments

For this example, we fixed the navigation bar to the top of our web page, although we can use different attachments. For instance, we can attach the navigation bar to the bottom using the `.navbar-fixed-bottom` class.



If it is attached in the bottom, do not forget to change the `<body>` padding from `top` to `bottom` in the CSS code.

The bar can also be static. For that, use the `.navbar-static-*` class, where the asterisk can mean `top` or `bottom`. If you are using the static navigation bars, you must place a container (static or fluid) right in the next level of the component:

```
<nav class="navbar navbar-default navbar-static-top">
  <div class="container">
    ...
  </div>
</nav>
```

Coloring the bar

You can also change the color of the navigation bar. Bootstrap version 3 offers an inverted set of colors. To do so, add the `.navbar-inverse` class to the `<nav>` element, as follows:

```
<nav class="navbar navbar-inverse">
  ...
</nav>
```

In version 4, they added other color options. So if the background of your navigation bar has a dark color, add the `.navbar-dark` class to make the text and other elements white. If the background has a light color, use the `.navbar-light` class to get the opposite result.

For the background color, you can pass a class called `.bg-*`, where the asterisk means a set of colors from Bootstrap. These are `default`, `primary`, `info`, `success`, `warning`, or `danger`:

```
<nav class="navbar navbar-dark bg-danger">
  ...
</nav>
```

Dropping it down

It is time to go back to the buttons once more. Now we will use the buttons dropdown. Button dropdowns are great for grouping a set of options in a single button. It can be used in several situations.



Remember that it is necessary to use Bootstrap JavaScript for buttons drop-downs as well.

To make use of these, you just need to make some small markups and class usages. We will also go a little further and add a button dropdown to our new navigation bar. The complete HTML code of the `<nav>` tag is this one:

```
<nav class="navbar navbar-default navbar-fixed-top">
  <div class="navbar-header">
    <a class="navbar-brand" href="landing_page.html">Company
    name</a>
    <button type="button" class="navbar-toggle collapsed" data-
    toggle="collapse" data-target="#nav-menu" aria-expanded="false">
      <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="btn btn-primary navbar-btn pull-right" href="#" 
    role="button">Sign up</a> -->
  </div>

  <div class="btn-group pull-right">
    <button type="button" class="btn btn-primary dropdown-toggle"
    data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
      Customer area <span class="caret"></span>
    </button>
    <ul class="dropdown-menu">
      <li><a href="#">Action</a></li>
      <li><a href="#">Another action</a></li>
      <li><a href="#">Something else here</a></li>
      <li role="separator" class="divider"></li>
      <li><a href="#">Separated link</a></li>
    </ul>
  </div>

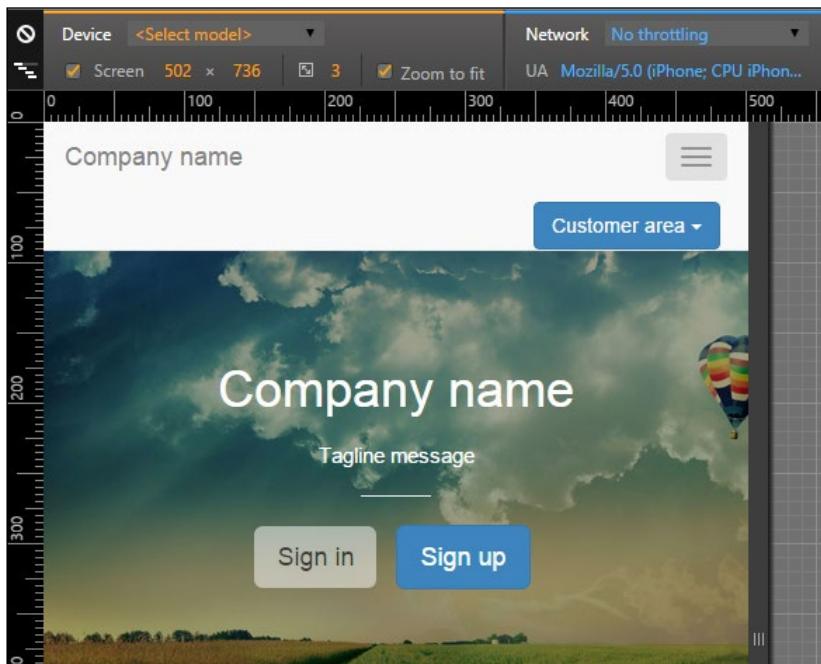
  <div id="nav-menu" class="collapse navbar-collapse navbar-right">
    <ul class="nav navbar-nav">
      <li><a href="#about">About</a></li>
      <li><a href="#features">Features</a></li>
      <li><a href="#pricing">Pricing</a></li>
      <li><a href="contact.html">Contact</a></li>
    </ul>
  </div>
</nav>
```

The highlighted code is the new one for the drop-down button. We have to create a `<button>`, followed by a list ``, all of that wrapped by a `div.btn-group`. It is a pretty strict piece of code that should be used for these components.

Regarding the CSS, we must add some spacing between the button and the list. So, the CSS for the button drop-down is as follows:

```
nav .btn-group {  
    margin: 0.8rem 2rem 0 0;  
}
```

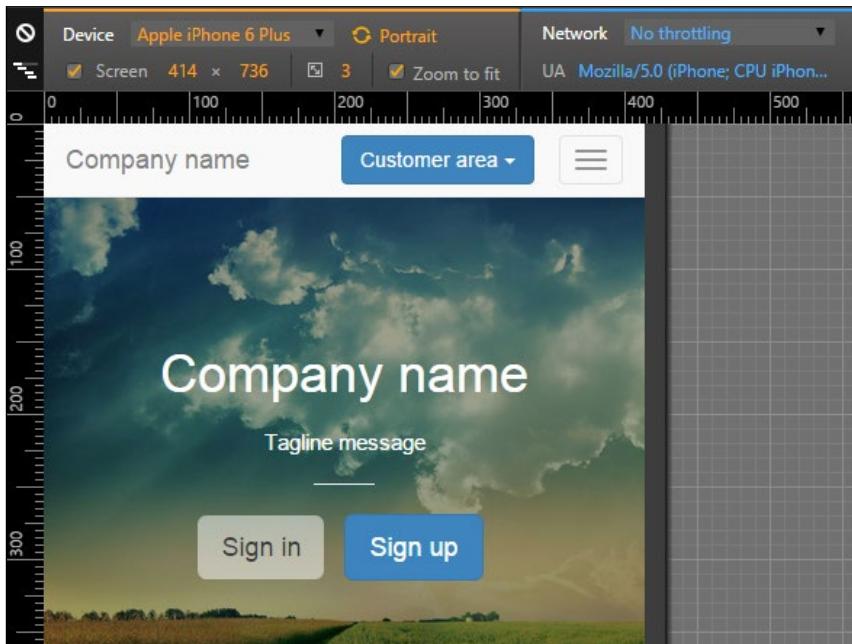
The result for the button is presented in the following screenshot:



Oops! If you see the example for large devices, the new button looks pretty good, although it looks badly placed for small devices. Let's fix this with a `media` query!

```
@media (max-width: 48em) {  
    nav .btn-group {  
        position: absolute;  
        top: 0;  
        right: 4em;  
    }  
}
```

After the fix, the output that you get should be as shown in this screenshot:

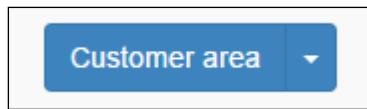


Customizing buttons dropdown

The Bootstrap buttons dropdown offers some custom options. The first one that we will discuss is the split option. To do this, you need to change your HTML a bit:

```
<div class="btn-group pull-right">
  <button type="button" class="btn btn-primary">Customer
area</button>
  <button type="button" class="btn btn-primary dropdown-toggle" data-
toggle="dropdown" aria-haspopup="true" aria-expanded="false">
    <span class="caret"></span>
    <span class="sr-only">Toggle Dropdown</span>
  </button>
  <ul class="dropdown-menu">
    <li><a href="#">Action</a></li>
    <li><a href="#">Another action</a></li>
    <li><a href="#">Something else here</a></li>
    <li role="separator" class="divider"></li>
    <li><a href="#">Separated link</a></li>
  </ul>
</div>
```

The main difference is the bold text, where we create another button, which will be responsible for the split effect, as shown in the following screenshot:



Moving on, you can make the drop-down a "drop-up". To do that, simply add the class to `div.btn-group`:

```
<div class="btn-group dropdown">  
    ...  
</div>
```

Making an input grouping

As we discussed in the last chapter, it is possible to group components together with inputs, as we did to the sign form in the home page. However, it is possible to add even more things to inputs. We will talk about some group options that can be useful.

First of all, let's exemplify the usage of grouping inputs and buttons. The main idea is almost the same—creating a `div.input-group`, and creating an input and a button inside this element, as shown in this HTML code:

```
<div class="input-group">  
    <input type="text" class="form-control" placeholder="Type the  
    page title...">  
    <span class="input-group-btn">  
        <button class="btn btn-success" type="button">Search</button>  
    </span>  
</div>
```

The output of the preceding code is shown in the following screenshot:



The only trick here is to add a `` element wrapping the button. If you invert the input order with the button, you will prepend the button to the input:

```
<div class="input-group">
  <span class="input-group-btn">
    <button class="btn btn-success" type="button">Search</button>
  </span>
  <input type="text" class="form-control" placeholder="Type the page
title...">
</div>
```

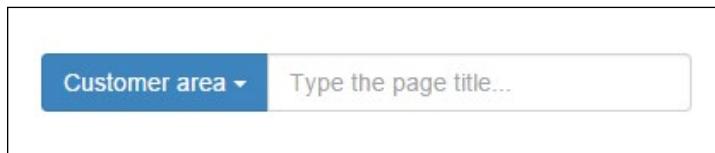
The output of the preceding code is shown in this screenshot:



Bootstrap also gives us the possibility to add any other kind of button. To exemplify this, let's now add a button dropdown grouped with an input. Replace `<button>` with the button dropdown that we just used in the previous example:

```
<div class="input-group">
  <span class="input-group-btn">
    <div class="btn-group pull-right">
      <button type="button" class="btn btn-primary dropdown-toggle"
data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
        Customer area <span class="caret"></span>
      </button>
      <ul class="dropdown-menu">
        <li><a href="#">Action</a></li>
        <li><a href="#">Another action</a></li>
        <li><a href="#">Something else here</a></li>
        <li role="separator" class="divider"></li>
        <li><a href="#">Separated link</a></li>
      </ul>
    </div>
  </span>
  <input type="text" class="form-control" placeholder="Type the
page title...">
</div>
```

It is pretty simple; you can add almost any kind of button, prepended or appended in an input. The following screenshot shows the result of the previous HTML code:



Can you append two buttons?

This is a small challenge for you. Can you append two buttons to the same input? Try to append some more buttons to `.input-group` and see what happens!

Getting ready for flexbox!

In version 4 of Bootstrap, flexbox support has finally arrived! However, it is an opt-in that can be used. The first step is to understand a little bit of flexbox, just in case you don't know, and then start using it.

We will not add any other element to our landing page example, since support for flexbox just began with Bootstrap 4. We will cover it only to clarify this new option.

Understanding flexbox

The definition of flexbox came out with the CSS3 specifications. Its main purpose is to better organize elements in a web page in a predictable manner. It can be seen as an option similar to `float` but one that offers a lot more choices, such as reordering elements and avoiding known issues of `float`, for example, the `clearfix` workaround.

For a hierarchical organization, first of all, you need to wrap the element of all the flex items (such as the columns inside a `.row`). It is also possible to play with the direction and axis from the wrapping element.

To exemplify the usage, let's create an HTML example. Create another file, named `flexbox.html`, use your base template and place the HTML code inside the `<body>` tag:

```
<body>
  <div class="wrapping-flex">
    <div class="item1">Item 1</div>
    <div class="item2">Item 2</div>
    <div class="item3">Item 3</div>
  </div>
</body>
```

So, in this case, we must make the `div.wrapping-flex` the flex wrapping element. Apply the following CSS and you will get the child elements placed inline:

```
.wrapping-flex {
  display: -webkit-flex;
  display: flex;
  background-color: #CCC;
}

.wrapping-flex > div {
  background-color: #ECA45A;
  margin: 1rem;
  padding: 1.5rem;
}
```

Create this is a sample HTML page and you will get the following output:



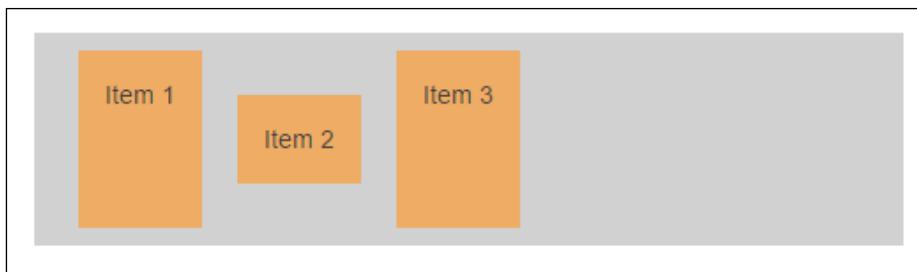
There are a plenty of options for flexbox. I do recommend the guide at <https://css-tricks.com/snippets/css/a-guide-to-flexbox/> for you to learn more about flexbox, since it is not our focus.

However, let's show a very powerful use case of flexbox. Have you ever faced a problem with aligning one `div` inside another vertically? I hope not, because it can be a pain in the neck, even more if you made it for older browsers.

With flexbox, we just have to apply the following CSS:

```
.wrapping-flex {  
    display: -webkit-flex;  
    display: flex;  
    background-color: #CCC;  
    height: 12rem;  
    width: 50%;  
    margin-left: 20%;  
}  
  
.wrapping-flex > div {  
    background-color: #ECA45A;  
    margin: 1rem;  
    padding: 1.5rem;  
}  
  
.wrapping-flex .item2 {  
    align-self: center;  
    height: 5rem;  
}
```

We added a `height: 12rem` to the wrapping element and set `align-self: center` and `height: 5rem` to the `.item2` element. With that, we align the second flex child `<div>` in the center, while the other two children continue to occupy the full height, as shown in the following screenshot:



Playing with Bootstrap and flexbox

Version 4 of Bootstrap provides two ways to use flexbox. The first one is with Sass, where you need to set the `$enable-flex` variable to the `true` state.

The other option is to download the compiled CSS version using the flex opt-in. The compiled version can be found in the Bootstrap repository (<https://github.com/twbs/bootstrap/releases>).

With regard to using flexbox, you will have limited browser support, since not all browsers are ready for this property. Consider using it if you will have access only from new browsers, such as Internet Explorer versions newer than v10.

Check out the currently available support for the flexbox property here:

<http://caniuse.com/#feat=flexbox>.

Summary

In this chapter, we took a big step towards more complex elements and theory. You deserve congratulations for nailing the first example of this book!

First, we presented icons in Bootstrap! It is a very handy tool to place the perfect icon in a perfect way on your page, by customizing the icon size and color. In version 4 of Bootstrap, they dropped native support for Glyphicons, even though you can still use it as a third-party library.

Then we touched the navigations bar of Bootstrap and presented a bunch of options to customize it for our case. We played with some tricks to collapse the menu in the navigation bar, and added more components to it, such as the button dropdown.

Moreover, we again talked about input grouping by showing some more examples of its usage, such as a group of an input and a button.

Finally, we added some theory to the soup, by introducing flexbox and showing that you can use it in conjunction with Bootstrap in the new version 4.

In the next chapters, we will dive into another example. We will start creating a web app! For that, expect the use of even more Bootstrap elements and components. By the end of the explanation, you will be qualified to create any web application!

6

Can You Build a Web App?

Among all kinds of web pages, the web application is the one with the fastest growth in usage. So, we will take a deep dive into this area by developing a really nice web application. Actually, Bootstrap was mainly designed for this type of application, since it was developed at first for the Twitter web application.

Therefore, in this chapter and in the next ones, we will take the reverse path. Instead of developing Bootstrap for Twitter, we will develop an app like Twitter with Bootstrap. With that, we will touch upon even more components and elements of Bootstrap, as follows:

- Web application definitions
- A fully customized navigation bar
- Cards
- Thumbnails
- Pagination
- Breadcrumbs

This chapter will be a bit more difficult, but I believe you are ready for this. So, can you build a web app?

Understanding web applications

Web applications came from the mix of an application and a browser, of course! Basically, a web application is a client application that runs on a web browser. Thus most of the processes are done on the client machine and the server is just responsible for the data processing.

This is interesting, since you can always deliver to the client the most updated version of the application, while the client does not need to upgrade the software. This leads to fast-paced and continuous development of the app.

Creating the code structure

Just as we always say when starting a new example, let's use the HTML boilerplate that we always use, keeping the same folder structure and so on:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1">
    <title>Web App</title>

    <link rel="stylesheet" href="css/bootstrap.css">
    <link rel="stylesheet" href="css/base.css">

    <!-- [if lt IE 9 ]>
        <script
src="https://oss.maxcdn.com/html5shiv/3.7.2/html5shiv.min.js">
    </script>
        <script
src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js">
    </script>
    <![endif] -->
    </head>
    <body>
        <script src="js/jquery-1.11.3.js"></script>
        <script src="js/bootstrap.js"></script>
        <script src="js/main.js"></script>
    </body>
</html>
```

Adding the navigation

First of all, we will add the navigation bar to our web application. Before the start of the `<body>` tag, add the navigation bar, just as we did in the last chapter:

```
<nav class="navbar navbar-default navbar-fixed-top">
  <div class="container">
```

```

<div class="navbar-header">
    <a class="navbar-brand" href="webapp.html">
        
    </a>
    <button type="button" class="navbar-toggle collapsed" data-
        toggle="collapse" data-target="#nav-menu" aria-expanded="false">
        <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="btn btn-primary navbar-btn pull-left" href="#" role="button">Sign up</a> -->
</div>

<div id="nav-menu" class="collapse navbar-collapse">
    <ul class="nav navbar-nav">
        </ul>
    </div>
</div>
</nav>

```

First, we created a simple navigation bar with the collapse option, just as we did in the last chapter. The major difference this time is the addition of the image `` logo. The CSS for adjusting the logo is as follows:

```

.navbar-brand img {
    height: 100%
}

```

So, we need to create the items inside the list `ul.nav.navbar-nav` tag. Append the following code inside the list:

```

<ul class="nav navbar-nav">
    <li>
        <a href="#">
            Home
        </a>
    </li>
    <li>
        <a href="#">
            Notifications
        </a>
    </li>
    <li>

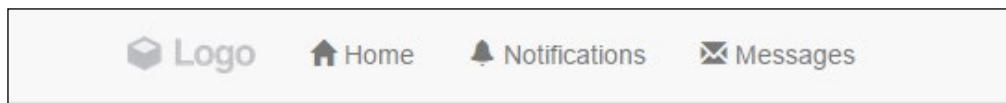
```

```
<a href="#">  
    Messages  
</a>  
</li>  
</ul>
```

Therefore, we should add some icons to each menu. Do you remember how to do this? We need to use the Bootstrap Glyphicons. Add the icons, as highlighted in this HTML code:

```
<ul class="nav navbar-nav">  
    <li>  
        <a href="#">  
            <span class="glyphicon glyphicon-home" aria-hidden="true"></span>  
                Home  
        </a>  
    </li>  
    <li>  
        <a href="#">  
            <span class="glyphicon glyphicon-bell" aria-hidden="true"></span>  
                Notifications  
        </a>  
    </li>  
    <li>  
        <a href="#">  
            <span class="glyphicon glyphicon-envelope" aria-hidden="true"></span>  
                Messages  
        </a>  
    </li>  
</ul>
```

The result right now should look like what is shown in the following screenshot:



Adding the search input

In our navigation bar, we will add a search input. There are two tricks for this. The first is the input must be like an input group to have a magnifier icon on the right-hand-side part. The second is that the input must be aligned to the right and not to the left in the `<nav>`. In the HTML, let's create a form after `ul.nav.navbar-nav`:

```
<div id="nav-menu" class="collapse navbar-collapse">
  <ul class="nav navbar-nav">
    ...
  </ul>

  <form id="search" role="search">
    <div class="input-group">
      <input type="text" class="form-control"
placeholder="Search...">
      <span class="glyphicon glyphicon-search" aria-hidden="true"></
span>
    </div>
  </form>
</div>
```

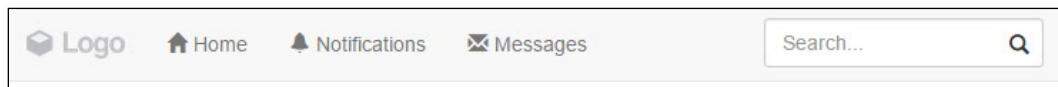
In the CSS, move the form to the right and add some padding:

```
nav form#search {
  float: right;
  padding: 0.5em;
}

nav form#search .glyphicon-search {
  z-index: 99;
  position: absolute;
  right: 0.7em;
  top: 50%;
  margin-top: -0.44em;
}

nav form#search .input-group .form-control {
  border-radius: 0.25em;
}
```

Refresh the web page and check out the input. It should appear as shown in this screenshot:



Time for the menu options!

Our navigation bar is starting to appear like the navigation bar of a web application, but not close enough! Now, it's the turn of the menu options.

The option at the thumbnail

We will now do some crazy stuff: add a thumbnail together with a Bootstrap button dropdown. Just before `form#search`, add the button HTML:

```
<div id="nav-options" class="btn-group pull-right">
  <button type="button" class="btn btn-default dropdown-toggle
thumbnail" data-toggle="dropdown" aria-haspopup="true" aria-
expanded="false">
    
  </button>
  <ul class="dropdown-menu">
    <li><a href="#">Profile</a></li>
    <li><a href="#">Setting</a></li>
    <li role="separator" class="divider"></li>
    <li><a href="#">Logout</a></li>
  </ul>
</div>
```

Basically, we used the template for a button dropdown (which you learned about in the previous chapter) and just removed the `.caret` component present on it. Instead of adding some text, we added an image, that is, the profile image. In `.btn-group`, we applied the helper class from Bootstrap, `.pull-right`. Since it was placed before the form, the button will appear after the form.

Then, it's time for the CSS. We need to resize the image and properly set the margins and paddings:

```
#nav-options {
  margin: 0.5em;
}

#nav-options button.thumbnail {
```

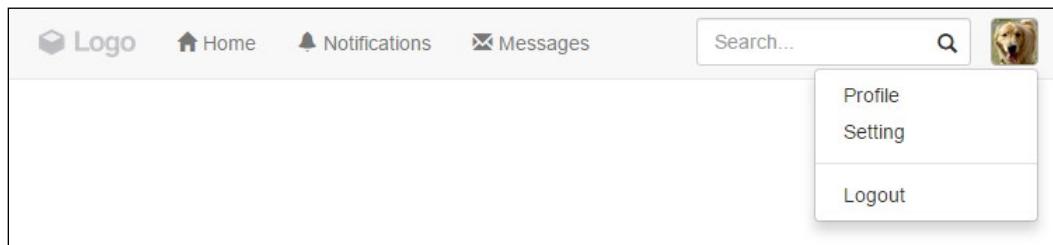
```

margin: 0;
padding: 0;
}

#nav-options img {
  max-height: 2.3em;
  border-radius: 0.3em;
}

```

The result of the addition of the button should be like what is shown in the following screenshot:



Adding the Tweet button

The last element present in the navigation bar is the **Tweet** button. To add it, we set following the HTML right before the button group option that we just added:

```

<button id="tweet" class="btn btn-default pull-right">
  <span class="glyphicon glyphicon-pencil" aria-hidden="true"></span>
  Tweet
</button>

```

For the CSS, we just need to add some margin:

```

#tweet {
  margin: 0.5em;
}

```

Finally, we have all the elements and components in our navigation bar, and it should look like this:



Customizing the navigation bar

Now that we have our navigation bar done, it's time to customize the Bootstrap theme, add some tweaks, and fix viewport issues.

Setting up the custom theme

To be a little different, we will use a blue background color for our navigation bar. First, we need to add some simple CSS rules:

```
.navbar-default {  
    background-color: #2F92CA;  
}  
  
.navbar-default .navbar-nav > li > a {  
    color: #FFF;  
}
```

Afterwards, let's add the active option to the list on the navigation. Add the `.active` class to the first element of the nav list (the **Home** one), presented in bold in the following code:

```
<ul class="nav navbar-nav">  
    <li class="active">  
        <a href="#">  
            <span class="glyphicon glyphicon-home" aria-hidden="true"></span>  
            Home  
        </a>  
    </li>  
    ... <!--others li and the rest of the code -->  
</ul>
```

Then, go to the CSS and set the following:

```
.navbar-default {  
    background-color: #2F92CA;  
}  
  
.navbar-default .navbar-nav > li > a {  
    color: #FFF;  
}  
  
.navbar-default .navbar-nav > .active > a {  
    background-color: transparent;  
    color: #FFF;
```

```

padding-bottom: 10px;
border-bottom: 5px solid #FFF;
}

```

The result of this should be like the one presented in the following screenshot. You can see that **Home** is in the active state. To mark that, we've added a border below it for denotation:



Fixing the list navigation bar pseudo-classes

If you hover over any element in the navigation list, you will see that it has the wrong color. We will use some style to fix that—by using CSS3 transitions! The complete CSS for the customization should be like the following:

```

.navbar-default {
  background-color: #2F92CA;
}

.navbar-default .navbar-nav > li > a,
.navbar-default .navbar-nav > li > a:hover {
  color: #FFF;
  -webkit-transition: all 150ms ease-in-out;
  -moz-transition: all 150ms ease-in-out;
  -ms-transition: all 150ms ease-in-out;
  -o-transition: all 150ms ease-in-out;
  transition: all 150ms ease-in-out;
}

.navbar-default .navbar-nav > .active > a {
  background-color: transparent;
  color: #FFF;
  padding-bottom: 0.62em;
  border-bottom: 0.45em solid #FFF; }

.navbar-default .navbar-nav > .active > a:hover,
.navbar-default .navbar-nav > li > a:hover {
  background-color: transparent;
  color: #F3F3F3;
  padding-bottom: 0.62em;
  border-bottom: 0.45em solid #F3F3F3;
}

```



CSS3 transitions

Transitions are an addition of CSS3 that allow us to change a property smoothly. We can pass in order the property (in our case, we used `all`), the time to complete the transition, and the animation function (we used `ease-in-out`).

Here, we had to change the default colors from the default Bootstrap navigation list. Also, by adding the transitions, we got a nice effect; when the user hovers over the menu, a border appears at the bottom of the item list.

You deserve a badge!

To finish the navigation bar, it would be nice to add some badges to the notifications item in the up list to show the number of new notifications, just as Twitter has on its website. For that, you will learn to use Bootstrap badges.

So, in the notifications item in the list, add the following highlighted HTML line:

```
<ul class="nav navbar-nav">
  <li class="active">
    <a href="#">
      <span class="glyphicon glyphicon-home" aria-hidden="true"></span>
      Home
    </a>
  </li>
  <li>
    <a href="#">
      <span class="badge">5</span>
      <span class="glyphicon glyphicon-bell" aria-hidden="true"></span>
      Notifications
    </a>
  </li>
  <li>
    <a href="#">
      <span class="glyphicon glyphicon-envelope" aria-hidden="true"></span>
      Messages
    </a>
  </li>
</ul>
```

For the CSS, set some positions, paddings, and borders:

```
.navbar-nav .badge {
    color: #2F92CA;
    background-color: #FFF;
    font-size: 0.7em;
    padding: 0.27rem 0.55rem 0.2rem 0.4rem;
    position: absolute;
    left: 0.37rem;
    top: 0.7rem;
    z-index: 99;
    border: 0.2rem solid #2F92CA;
}
```

Nicely done! Refresh the browser and you will see this pretty, beautiful badge:



Fixing some issues with the navigation bar

We now have three issues with the navigation bar. Can you guess them?

They are the **Tweet** button at the small viewport, the collapsed navigation menu collapse, and the color of the collapse *hamburger* button.

Well, first we will handle the easiest one—fix the **Tweet** button! For that, we will create another element to be placed at the left-hand side of the collapse button and just display it when they are in extra small resolution. First, add the `.hidden-xs` class to the current Tweet button:

```
<button id="tweet" class="btn btn-default pull-right hidden-xs">
    <span class="glyphicon glyphicon-pencil" aria-hidden="true"></span>
    Tweet
</button>
```

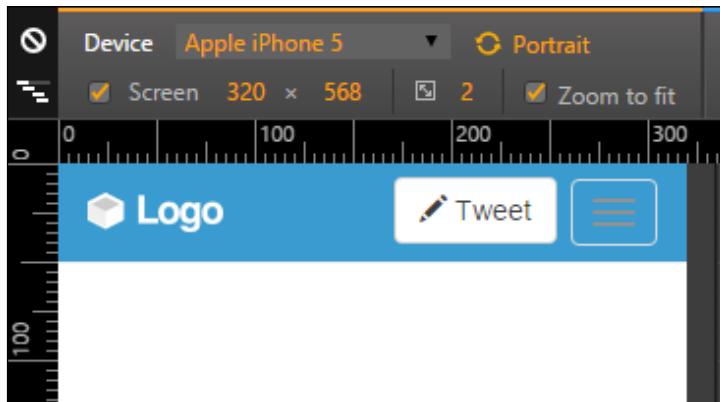
Secondly, at `.navbar-header`, after `button.navbar-toggle`, add the following highlighted button:

```
<div class="navbar-header">
    <a class="navbar-brand" href="webapp.html">
        
    </a>
```

```
<button type="button" class="navbar-toggle collapsed" data-
toggle="collapse" data-target="#nav-menu" aria-expanded="false">
  <span class="sr-only">Toggle navigation</span>
  <span class="icon-bar"></span>
  <span class="icon-bar"></span>
  <span class="icon-bar"></span>
</button>

<button id="tweet" class="btn btn-default pull-right visible-xs-
block">
  <span class="glyphicon glyphicon-pencil" aria-hidden="true"></
span>
  Tweet
</button>
</div>
```

So, what we did is hide the **Tweet** button for extra small devices and show a new one in a different element. Set a mobile viewport and you can see the button's position fixed, as follows:



Next, let's fix the color of the collapse *hamburger* button. Just apply the next CSS to change its color:

```
.navbar-header .navbar-toggle,
.navbar-default .navbar-toggle:focus {
  background-color: #57A5D2;
}

.navbar-default .navbar-toggle:hover {
  background-color: #3986B3;
}
```

```
.navbar-default .navbar-toggle .icon-bar {
    background-color: #FFF;
}
```

Finally, let's customize the collapsed navigation bar using Bootstrap helpers. We add the `.hidden-xs` class to `.nav-options` and the `.hidden-sm` class to the `form#search` element, making them invisible for extra small and small devices respectively, just as we did to the **Tweet** button:

```
<div id="nav-options" class="btn-group pull-right hidden-xs">
    ...
</div>

<form id="search" role="search" class="hidden-sm">
    ...
</form>
```

Then, in the `ul.nav.navbar-nav` navigation list, create two items that will replace the ones hidden at the current viewport:

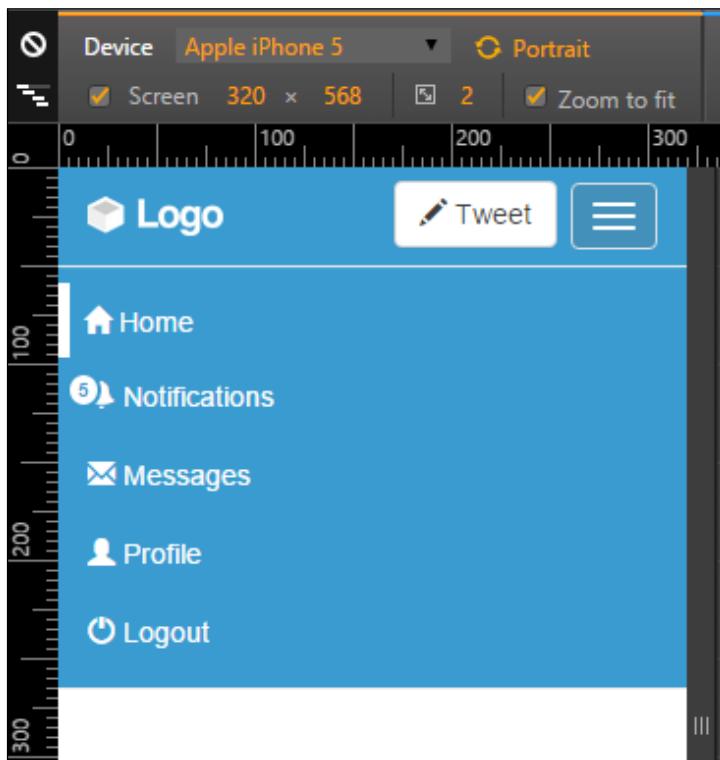
```
<ul class="nav navbar-nav">
    ...
    <!-- others elements list were hidden -->
    <li class="visible-xs-inline">
        <a href="#">
            <span class="glyphicon glyphicon-user" aria-hidden="true"></span>
            Profile
        </a>
    </li>
    <li class="visible-xs-inline">
        <a href="#">
            <span class="glyphicon glyphicon-off" aria-hidden="true"></span>
            Logout
        </a>
    </li>
</ul>
```

Thus we are making them visible for extra small resolution with the `.visible-xs-inline` class, as long they are from an inline list.

To wrap it up, let's remove the border in the active list item, since it does not seem nice at the bottom in the layout. Let's change it to a right border instead of bottom with the following CSS using a media query:

```
@media (max-width:34em) {  
    .navbar-default .navbar-nav > .active > a {  
        border-bottom: none;  
        border-left: 0.45em solid #FFF;  
        padding-left: 0.5em;  
    }  
}
```

And we are done! Refresh the web page and see the final result of the navigation bar. It is awesome!



Do a grid again

We have finally finished the navigation bar. Now it's time to the page main content. For that, we must create a page grid. Following how Twitter uses a three-column-based layout, we will do the same. The HTML code for the scaffolding is the one that should be placed after the `<nav>` element:

```
<div class="container">
  <div class="row">
    <div id="profile" class="col-md-3 hidden-sm hidden-xs"></div>
    <div id="main" class="col-sm-12 col-md-6"> </div>
    <div id="right-content" class="col-md-3 hidden-sm hidden-xs"> </div>
  </div>
</div>
</div>
```

To understand it, we just created a `.container` with a single `.row`. The `.row` contains three columns, the first and the last being visible only for medium and larger devices. This is because of the `.hidden-sm` and `.hidden-xs` classes. When both columns are hidden, the middle column fills the row completely. This is because of the `.col-sm-12` class.

To finish that, add a `padding-top` to `<body>` in order to correct the page's position with respect to the navigation bar:

```
body {
  padding-top: 4em;
  background-color: #F5F8FA;
}
```

Playing the cards

Moving on, in our web application, we will create a new component containing the about information, named *Card*. We will take a break from page development to discuss this section in depth.

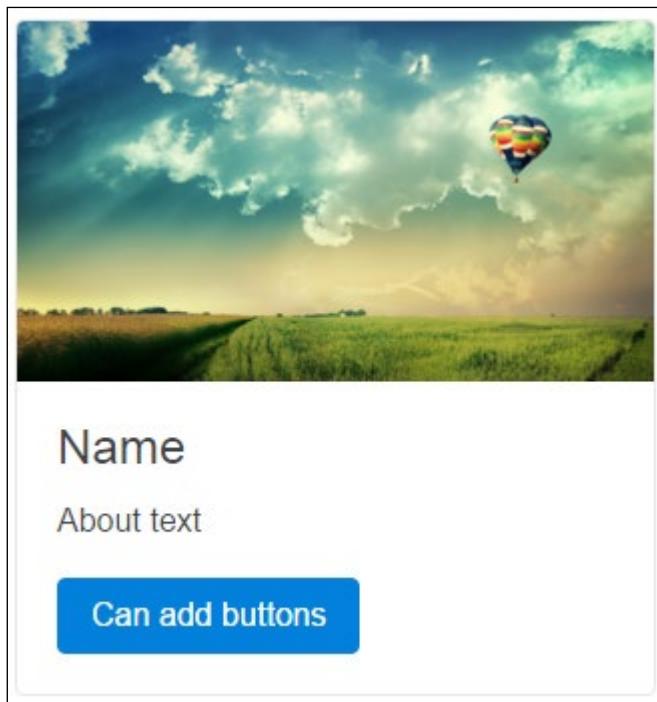
Cards are flexible container extensions that include internal options, such as header, footer, and other display options. In Bootstrap 4, there is a component called Card, but since we are supporting versions 3 and 4 in this book, we will teach both ways.

Learning cards in Bootstrap 4

As was mentioned before, Bootstrap 4 provides Cards components. To make use of them, create a `div.card` element and start adding elements such as `.card-block` and `.card-img-top`:

```
<div class="card">
  
  <div class="card-block">
    <h4 class="card-title">Name</h4>
    <p class="card-text">About text</p>
    <a href="#" class="btn btn-primary">Can add buttons</a>
  </div>
</div>
```

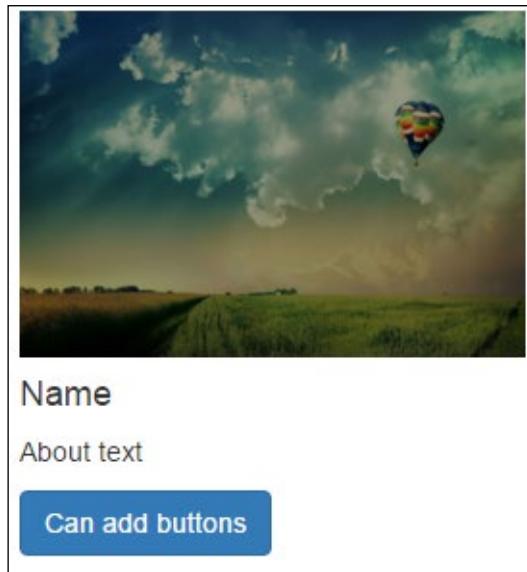
For the preceding code, the output will look like what is shown in the following screenshot. As we can see, the Card component is pretty simple and straightforward. The component offers some other options as well, but we will talk about that when needed.



Creating your own cards

Like the famous quote, *if you have lemons, make lemonade*, in Bootstrap version 3, we do not have the Card component. However, we have the tools needed to make our own Card component for sure! So let's have some lemonade!

We will use the same classes and structures of Bootstrap 4, playing with only the CSS. Therefore, if you are using version 3, you will see the page render like this for the use of the same HTML from version 4:



To squeeze the first lemon, let's create the CSS for the `.card` class:

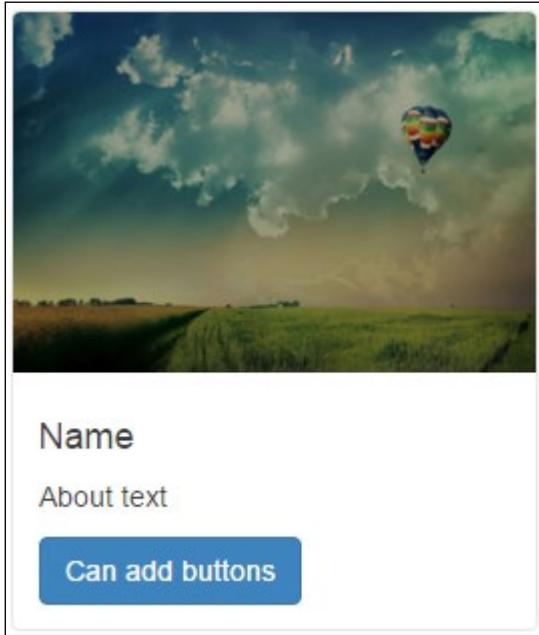
```
.card {  
  position: relative;  
  border: 0.1rem solid #e5e5e5;  
  border-radius: 0.4rem;  
  background-color: #FFF;  
}
```

Following this, just add two CSS rules for `img.card-img-top` and `.card-block`, as shown here:

```
.card-img-top {  
  border-radius: 0.4rem 0.4rem 0 0;  
}
```

```
.card-block {  
    padding: 1.25rem;  
}
```

Done! We have our own card component ready for Bootstrap 3. The next screenshot presents the final result. Of course, there are some differences of typography and button color, but these are the differences because of the version; the component is perfectly done.



Can you finish the Card component?

[] We presented just a few options for the Card component in Bootstrap version 3. Can you do the rest? Try making some CSS rules for classes such as `.card-img-bottom`, `.card-header`, and `.card-footer`.

Adding Cards to our web application

Getting back to the web application, let's add the Card components inside `div#profile`, at the main container. The HTML code for this section will be as follows:

```
<div id="profile-resume" class="card">  
    
```

```

<div class="card-block">
  
  <h4 class="card-title">Jonny Doo <small>@jonnydoo</small></h4>
  <p class="card-text">Dog goes woofy. Did you said squitly?</p>
  <ul class="list-inline list-unstyled">
    <li id="card-tweets">
      <a href="#">
        <span class="profile-stats">Tweets</span>
        <span class="profile-value">99k</span>
      </a>
    </li>
    <li class="card-following">
      <a href="#">
        <span class="profile-stats">Following</span>
        <span class="profile-value">7</span>
      </a>
    </li>
    <li class="card-followers">
      <a href="#">
        <span class="profile-stats">Followers</span>
        <span class="profile-value">132k</span>
      </a>
    </li>
  </ul>
</div>
</div>

```

Breaking down the code, we added some components to `.card-block`. First of all is the `.card-img` element, which will represent the profile photography. Following this, we changed `.card-title` by adding a `<small>` tag inside `<h4>`. The last change is the addition of the `` list, representing some stats for the profile.

There is no secret in this HTML piece; we just added some elements in a straightforward way. Now it's time for the CSS rules. First, change the position and size of the `img.card-img` element:

```

.card-block img.card-img {
  top: 50%;
  margin-top: -36px;
  width: 72px;
  border: 3px solid #FFF;
  border-radius: 0.4rem;
  float: left;
  position: relative;
  z-index: 99;
}

```

Since it is in the right place, let's correctly align .card-title and add some padding to .card-text:

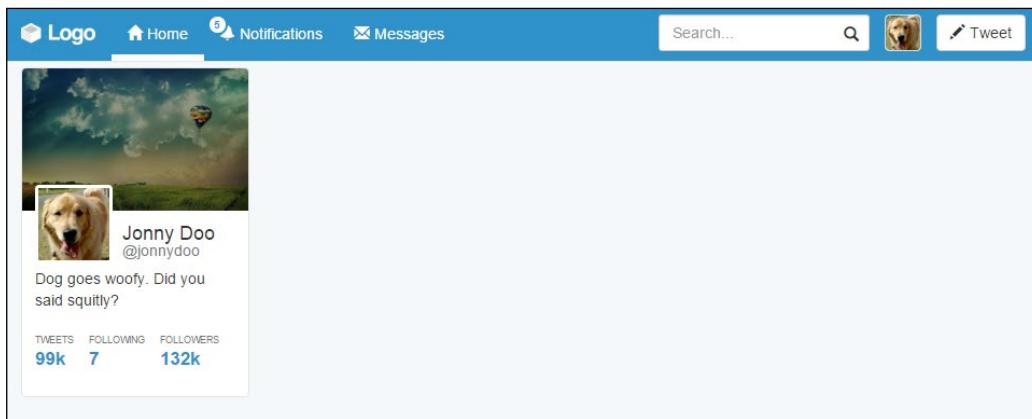
```
.card-block .card-title {  
    float: left;  
    margin: 0;  
    margin-left: 0.5em;  
}  
  
.card-block .card-title small {  
    display: block;  
}  
  
.card-block .card-text {  
    clear: both;  
    padding-top: 0.25em;  
    margin-bottom: 1.5em;  
}
```

 **Can you change the card block to use flexbox?**
Another challenge appears here. Since you have already learned about the usage of flexbox, try to replace the floats in the previous code with some flexbox CSS rules. Just keep in mind that it is recommended for Bootstrap 4 and works only on new browsers.

It is almost looking like the Twitter card on the left; we just need to change the list style inside the profile card. Add this CSS:

```
.card-block ul a:hover {  
    text-decoration: none;  
}  
  
.card-block ul .profile-stats {  
    color: #777;  
    display: block;  
    text-transform: uppercase;  
    font-size: 0.63em;  
}  
  
.card-block ul .profile-value {  
    font-size: 1.2em;  
    font-weight: bold;  
    color: #2F92CA;  
}
```

Well done! It looks prettier than the Twitter component. In the following screenshot, we present the expected result:



Another card using thumbnails

After the `#profile-resume` card, we will create another one named `#profile-photo`, which will contain photos of the user. Use the same cards methodology to place this new one after `#profile-resume` with the following HTML code:

```
<div id="profile-photo" class="card">
  <div class="card-header">Photos</div>
  <div class="card-block">
    <ul class="list-inline list-unstyled">
      <li>
        <a href="#" class="thumbnail"></a>
      </li>
      <li>
        <a href="#" class="thumbnail"></a>
      </li>
      <li>
        <a href="#" class="thumbnail"></a>
      </li>
      <li>
        <a href="#" class="thumbnail"></a>
      </li>
    </ul>
  </div>
</div>
```

In this card we will create a new card element, `.card-header`. In Bootstrap 4, you can use the regarding class, but in version 3, you will need this CSS rule:

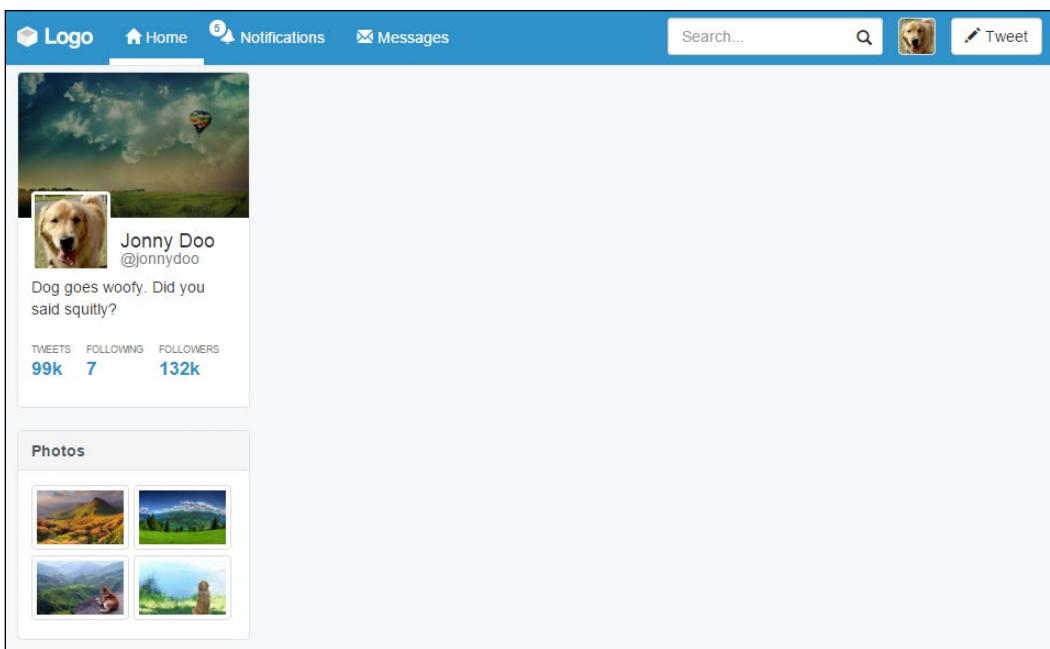
```
.card .card-header {  
    border-radius: 0.4rem 0.4rem 0 0;  
    padding: .75rem 1.25rem;  
    background-color: #f5f5f5;  
    border-bottom: 0.1em solid #e5e5e5;  
    color: #4e5665;  
    font-weight: bold;  
}
```

Moving on, the rest of CSS for this card is simple. Just change the image's width and adjust some margins and paddings:

```
#profile-photo {  
    margin-top: 2rem;  
}  
  
#profile-photo ul {  
    margin: 0;  
}  
  
#profile-photo li {  
    width: 48%;  
    padding: 0;  
}
```

Also note that we are using the `.thumbnail` class in the `<a>` tag that wraps the images. This class is useful for nicely styled thumbnail images. It can also be used to wrap text along with an image.

The photo card should look like what is shown in the following screenshot. Again, we will use some more cards in this web application, although we'll talk about that later, when needed.



Implementing the main content

Moving on, we will implement the main content in the middle of the page. This content will hold the feeds while allowing new tweets.

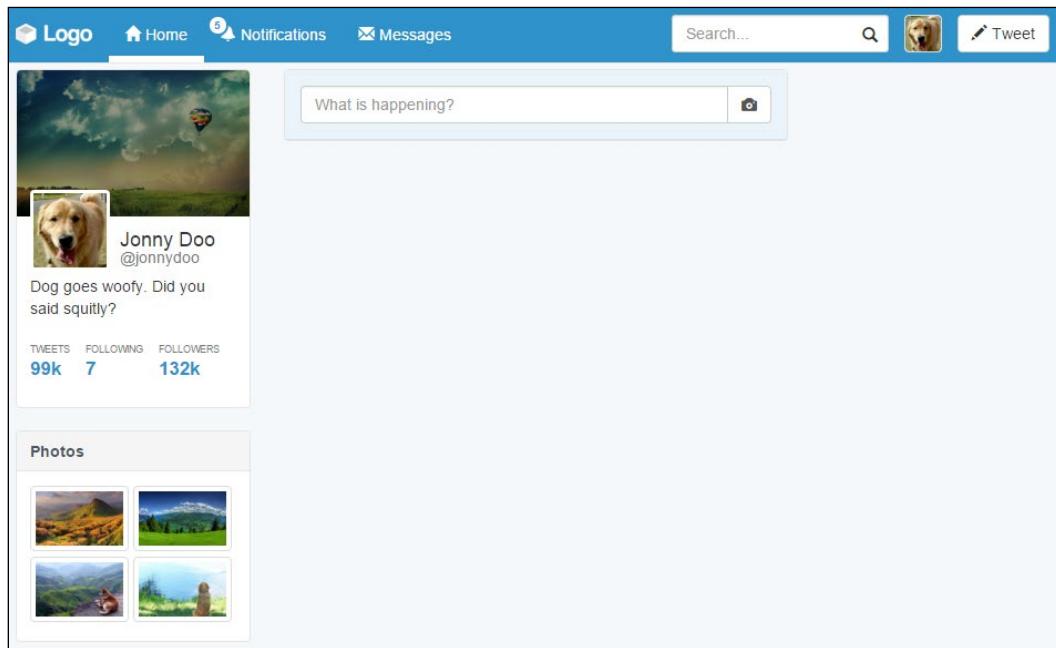
We need to create the input to send a new message. To do this, create the following HTML code at the `div#main` element:

```
<div id="main" class="col-sm-12 col-md-6">
  <div id="main-card" class="card">
    <form id="new-message">
      <div class="input-group">
        <input type="text" class="form-control" placeholder="What
is happening?">
        <span class="input-group-addon">
          <span class="glyphicon glyphicon-camera" aria-
hidden="true"></span>
        </span>
      </div>
    </form>
  </div>
</div>
```

For that, we created a form, again making use of input groups, icons, and cards. Can you see the ease provided by Bootstrap again? We just placed the elements with the right classes and everything went perfect. The next CSS takes place with some rules regarding the color and padding of the form:

```
form#new-message {  
    border-radius: 0.4rem 0.4rem 0 0;  
    padding: 1em;  
    border-bottom: 0.1em solid #CEE4F5;  
    background-color: #EBF4FB;  
}  
  
form#new-message .input-group-addon {  
    background-color: #FFF;  
}
```

At this point, the result should be as shown in the following screenshot. Next up, we will create the other elements.



Making your feed

We have made cool things so far, but the feed is the core of the page. We will create a nice and friendly feed for our web app.

As usual, let's create the HTML code first. The feed will work inside a stacked list. With that in mind, let's create the first element in the list:

```
<div id="main" class="col-sm-12 col-md-6">
  <div id="main-card" class="card">
    <form id="new-message">
      <div class="input-group">
        <input type="text" class="form-control" placeholder="What
is happening?">
        <span class="input-group-addon">
          <span class="glyphicon glyphicon-camera" aria-
hidden="true"></span>
        </span>
      </div>
    </form>
    <ul id="feed" class="list-unstyled">
      <li>
        
        <div class="feed-post">
          <h5>Name <small>@namex - 3h</small></h5>
          <p> You can't hold a dog down without staying down with
him!</p>
        </div>
        <div class="action-list">
          <a href="#">
            <span class="glyphicon glyphicon-share-alt" aria-
hidden="true"></span>
          </a>
          <a href="#">
            <span class="glyphicon glyphicon-refresh " aria-
hidden="true"></span>
            <span class="retweet-count">6</span>
          </a>
          <a href="#">
            <span class="glyphicon glyphicon-star" aria-
hidden="true"></span>
          </a>
        </div>
      </li>
    </ul>
  </div>
</div>
```

The highlighted code is the code added for the list. To understand it, we created an element in the list containing the common stuff inside a post, such as an image, a name, text, and options. Add the `.img-circle` class to the image in the list to style it using Bootstrap image styles.

With the CSS, we will correctly style the page. For the list and the image, apply the following rules:

```
ul#feed {  
    margin: 0;  
}  
  
ul#feed li {  
    padding: 1em 1em;  
}  
  
ul#feed .feed-avatar {  
    width: 13%;  
    display: inline-block;  
    vertical-align: top;  
}
```

By doing this, you will be correcting the margins and padding while adjusting the size of the image avatar. For the post section, use this CSS:

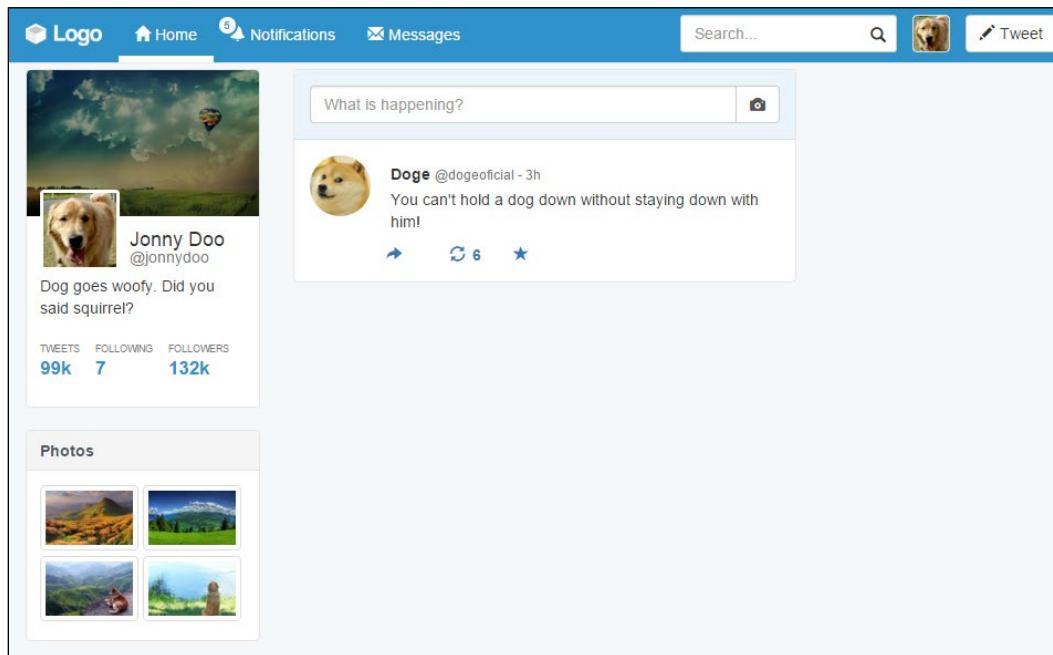
```
ul#feed .feed-post {  
    width: 80%;  
    display: inline-block;  
    margin-left: 2%;  
}  
  
ul#feed .feed-post h5 {  
    font-weight: bold;  
    margin-bottom: 0.5rem;  
}  
  
ul#feed .feed-post h5 > small {  
    font-size: 1.2rem;  
}
```

Finally, with regard to `.action-list`, set the following styles:

```
ul#feed .action-list {  
    margin-left: 13%;  
    padding-left: 1em;  
}
```

```
ul#feed .action-list a {  
    width: 15%;  
    display: inline-block;  
}  
  
ul#feed .action-list a:hover {  
    text-decoration: none;  
}  
  
ul#feed .action-list .retweet-count {  
    padding-left: 0.2em;  
    font-weight: bold;  
}
```

Refresh your browser and you will get this result:



Awesome! Note that for the post, we did all the spacing using percentage values. This is also a great option because the page will resize with respect to the user's resolution very smoothly.

We have only one problem now. Add another post and you will see that there is no divisor between the posts. To illustrate this, add a second post in the HTML code:

```
<ul id="feed" class="list-unstyled">
  <li>
    
    <div class="feed-post">
      <h5>Doge <small>@dogeoficial - 3h</small></h5>
      <p>You can't hold a dog down without staying down with
him!</p>
    </div>
    <div class="action-list">
      <a href="#">
        <span class="glyphicon glyphicon-share-alt" aria-
hidden="true"></span>
      </a>
      <a href="#">
        <span class="glyphicon glyphicon-refresh" aria-
hidden="true"></span>
        <span class="retweet-count">6</span>
      </a>
      <a href="#">
        <span class="glyphicon glyphicon-star" aria-hidden="true"></
span>
      </a>
    </div>
  </li>

  <li>
    
    <div class="feed-post">
      <h5>Laika <small>@spacesog - 4h</small></h5>
      <p>That's one small step for a dog, one giant leap for
giant</p>
    </div>
    <div class="action-list">
      <a href="#">
        <span class="glyphicon glyphicon-share-alt" aria-
hidden="true"></span>
      </a>
      <a href="#">
        <span class="glyphicon glyphicon-refresh" aria-
hidden="true"></span>
        <span class="retweet-count">6</span>
      </a>
    </div>
  </li>

```

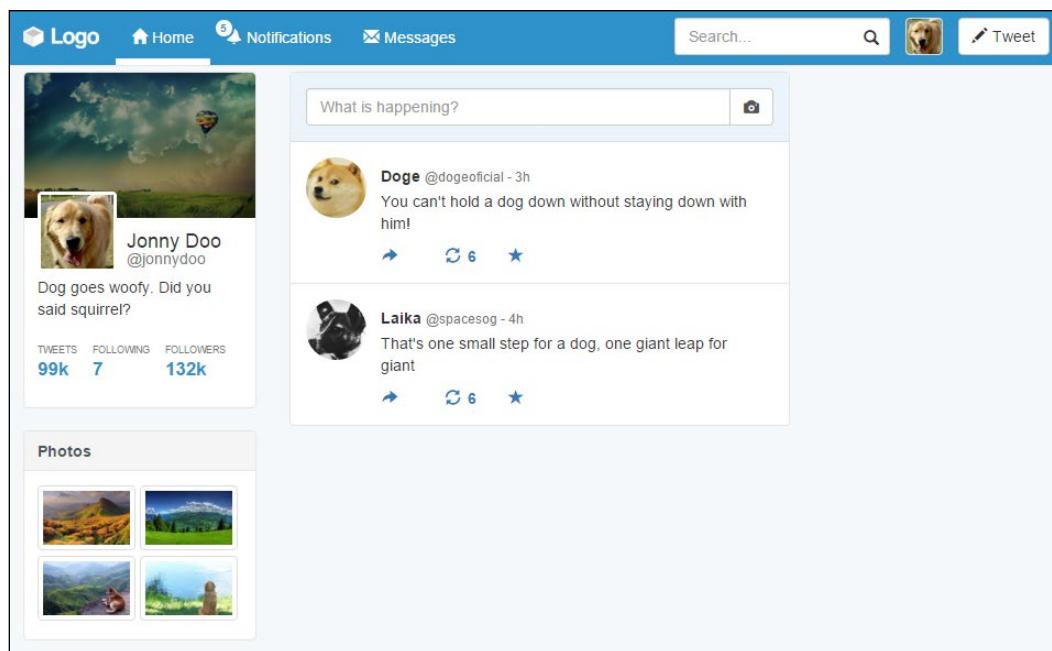
```
<a href="#">
    <span class="glyphicon glyphicon-star" aria-hidden="true"></span>
</a>
</div>
</li>
</ul>
```

At the CSS, change it a little to correct the padding and add a border between the items, as follows:

```
ul#feed li {
    padding: 1em 1em;
    border-bottom: 0.1rem solid #e5e5e5;
}

ul#feed li:last-child {
    border-bottom: none;
}
```

The output results will be like this:



Doing some pagination

Okay, I know that web applications such as Twitter usually use infinite loading and not pagination, but we need to learn that! Bootstrap offers incredible options for pagination, so let's use some of them right now.

From the start, create the HTML code for the component and insert it just after div.
main-card:

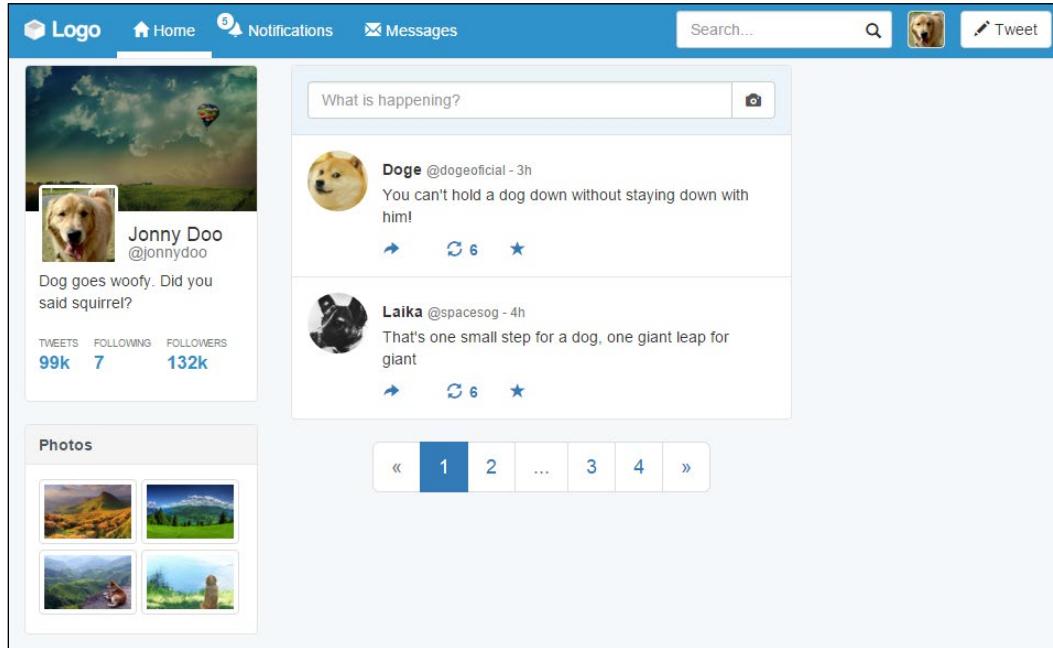
```
<nav class="text-center">
  <ul class="pagination pagination-lg">
    <li class="disabled"><a href="#" aria-label="Previous"><span aria-hidden="true">&laquo;</span></a></li>
    <li class="active"><a href="#">1 <span class="sr-only">(current)</span></a></li>
      <li><a href="#">2</a></li>
      <li class="disabled"><a href="#">...</a></li>
      <li><a href="#">3</a></li>
      <li><a href="#">4</a></li>
      <li><a href="#" aria-label="Next"><span aria-hidden="true">&raquo;</span></a></li>
    </ul>
  </nav>
```

Thus, we must consider a few things here. Firstly, to center the pagination, we used the helper class from Bootstrap, .text-center. This is because ul.pagination does apply the display: inline-block style.

Secondly, we created a `` with the .pagination class, determined by the framework. We also added the .pagination-lg class, which is an option of pagination for making it bigger.

Lastly, the .disabled class is present in two items of the list, the previous link and the ellipsis ... one. Also, the list on page 1 is marked as active, changing its background color.

Check out the result of adding pagination in this screenshot:



Creating breadcrumbs

To make use of Bootstrap breadcrumbs, we will add it to our web app. Note that we will do this step for learning purposes so that you will be able to create it when you need it.

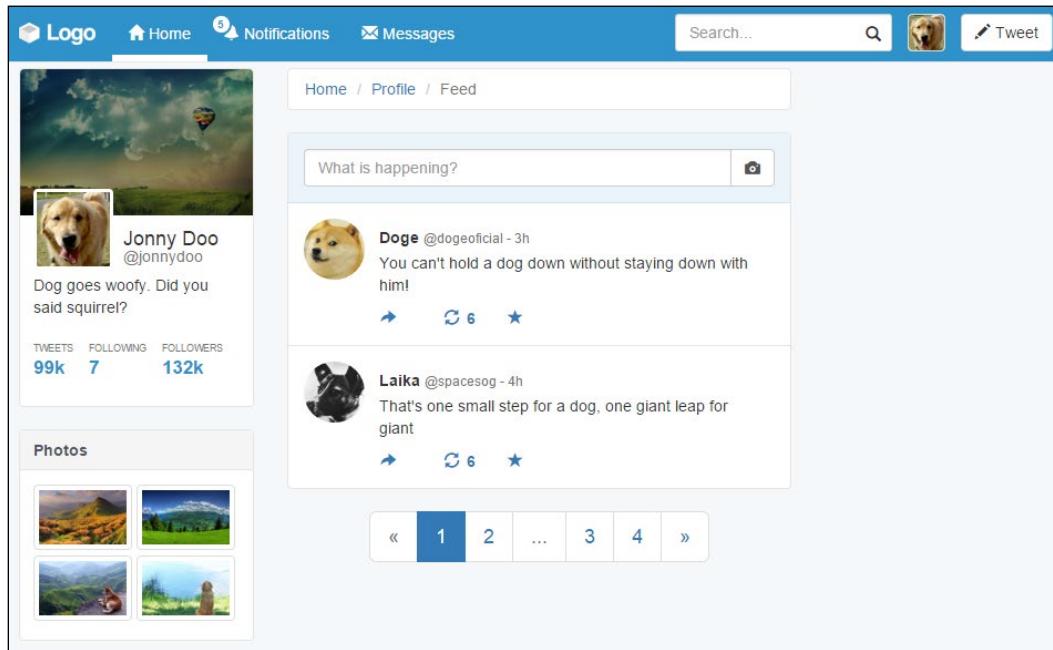
Like pagination, Bootstrap offers a component for breadcrumbs as well. For that, create an ordered list just after the open tag `div#main`:

```
<div id="main" class="col-sm-12 col-md-6">

<ol class="breadcrumb card">
  <li><a href="#">Home</a></li>
  <li><a href="#">Profile</a></li>
  <li class="active">Feed</li>
</ol>
...
</div>
```

The cool thing about Bootstrap breadcrumbs is that the separator bars are automatically added through :before and the content CSS option, so you do not need to worry about them.

Note that the .card class was added to the breadcrumbs component to keep the web app style. The following screenshot presents the result of breadcrumbs:



Finishing with the right-hand-side content

Well, we are almost done. It is time to create the right-hand-side content of our web app. The right-hand-side content contains information such as *Whom to follow* and the about page. Let's create it!

Coming to the HTML, let's create another Card component inside `div.right-content`, as follows:

```
<div id="right-content" class="col-md-3 hidden-sm hidden-xs">
  <div id="who-follow" class="card">
    <div class="card-header">
      Who to follow
    </div>
```

```
<div class="card-block">  
    </div>  
  </div>  
</div>
```

Inside .card-block, create a vertical list:

```
<div id="right-content" class="col-md-3 hidden-sm hidden-xs">  
  <div id="who-follow" class="card">  
    <div class="card-header">  
      Who to follow  
    </div>  
    <div class="card-block">  
      <ul class="list-unstyled">  
        <li>  
            
          <div class="info">  
            <strong>Crazy cats</strong>  
            <button class="btn btn-default">  
              <span class="glyphicon glyphicon-plus" aria-hidden="true"></span> Follow  
            </button>  
          </div>  
        </li>  
        <li>  
            
          <div class="info">  
            <strong>Free ration alert</strong>  
            <button class="btn btn-default">  
              <span class="glyphicon glyphicon-plus" aria-hidden="true"></span> Follow  
            </button>  
          </div>  
        </li>  
      </ul>  
    </div>  
  </div>  
</div>
```

So, the result without the CSS is not good, as shown in the following screenshot. We need to fix it.



First, we add margins for the items in the list:

```
div#who-follow li {  
    margin-bottom: 2em;  
}  
  
div#who-follow li:last-child {  
    margin-bottom: 0;  
}
```

Then, we adjust the size of the image and the following text:

```
div#who-follow li img {  
    width: 26%;  
    display: inline-block;  
    vertical-align: top;
```

```

    margin-right: 2%;
}

div#who-follow li .info {
    width: 68%;
    display: inline-block;
}

```

To finish this, we adjust the content inside the `.info` element:

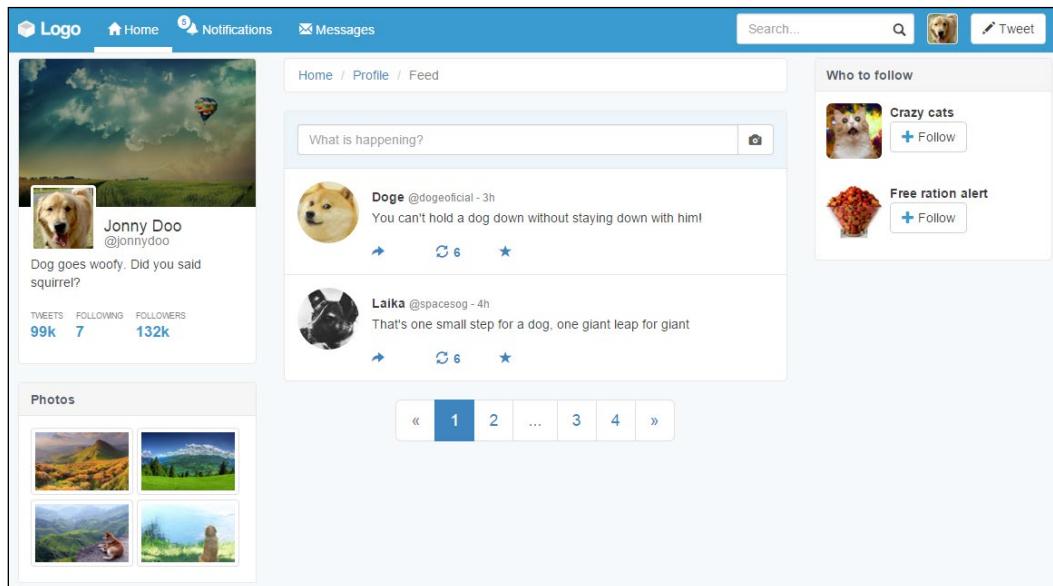
```

div#who-follow li .info strong {
    display: block;
    overflow:hidden;
    text-overflow: ellipsis;
}

div#who-follow li .info .glyphicon {
    color: #2F92CA;
}

```

The result should look like what is shown here:



To end the main web page content, let's create another card that has content about the web app, such as help, privacy, and so on. After the `div#who-follow`, create another card:

```
<div id="app-info" class="card">
  <div class="card-block">
    © 2015 SampleApp
    <ul class="list-unstyled list-inline">
      <li><a href="#">About</a></li>
      <li><a href="#">Terms and Privacy</a></li>
      <li><a href="#">Help</a></li>
      <li><a href="#">Status</a></li>
      <li><a href="#">Contact</a></li>
    </ul>
  </div>
  <div class="card-footer">
    <a href="#">Connect other address book</a>
  </div>
</div>
```

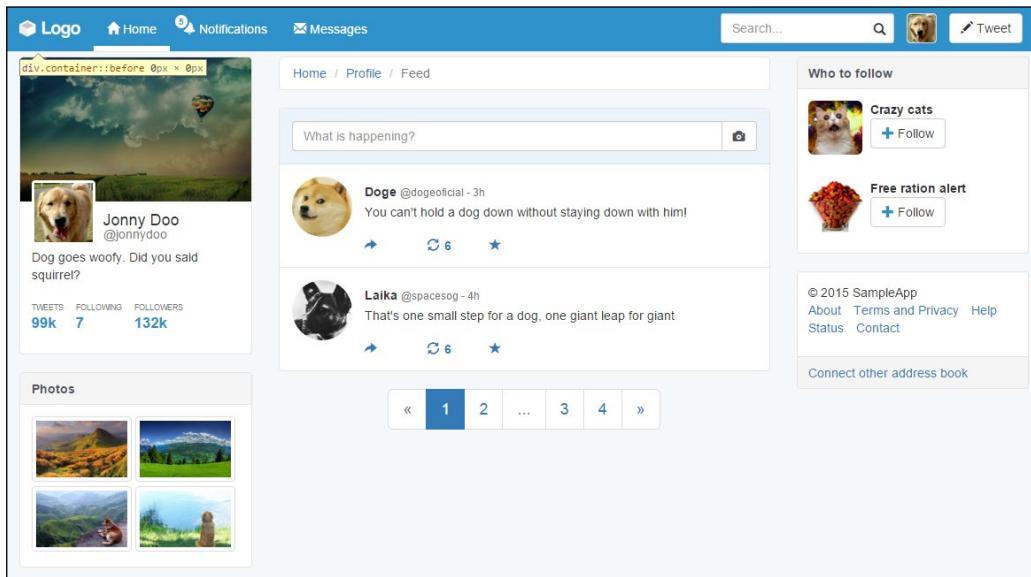
First of all, note that we have just used `.card-footer` for this card. If you are using Bootstrap 3, add the next CSS:

```
.card .card-footer {
  border-radius: 0 0 0.4rem 0.4rem;
  padding: .75rem 1.25rem;
  background-color: #f5f5f5;
  border-top: 0.1em solid #e5e5e5;
  color: #4e5665;
}
```

For this card, we also need to add some margin within the card above:

```
div#app-info {
  margin-top: 2rem;
}
```

That looks great! We have finished the majority of the components in our web application! The next image shows the final result that we will cover at this chapter. Great work!



Summary

In this chapter, we started the development of another example – an awesome web application like Twitter. We started creating every component with the help of Bootstrap, while also customizing each one. By the end of the chapter, we were done with the major part of the components to be added.

First, we created a fully customized navigation bar that works on any device. Just like at every component, we took special care with different visualizations for mobiles and desktops.

We talked a lot about cards. This is a new component in Bootstrap 4, but we created our own for version 3, so we nailed it all. Cards are present in every column, having different content and placements of items.

We also discussed the use of other Bootstrap components by making use of breadcrumbs, pagination, and thumbnails.

I hope now you feel confident about web application development, because in the next chapter, we will take a step further in this kind of development by using other Bootstrap components and more customization.

7

Of Course, You Can Build a Web App!

In this chapter, we will complete the elements of our web app with the use of other Bootstrap elements and components. By the end of this chapter, we will have covered the majority of elements present in Bootstrap, making you almost an expert as well as answering this question from the last chapter: can you build a web app? Of course you can!

We will cover some more complex Bootstrap components and elements. These are the key points of this chapter, and you will learn how to:

- Use Bootstrap alerts
- Customize alerts
- Progress bars
- CSS key frames
- Navigation components
- Tabs
- Labels and badges
- List groups

Even though these seem to be a lot of key points, they are easy to learn and master. So, I am sure you will be able to nail all of them.

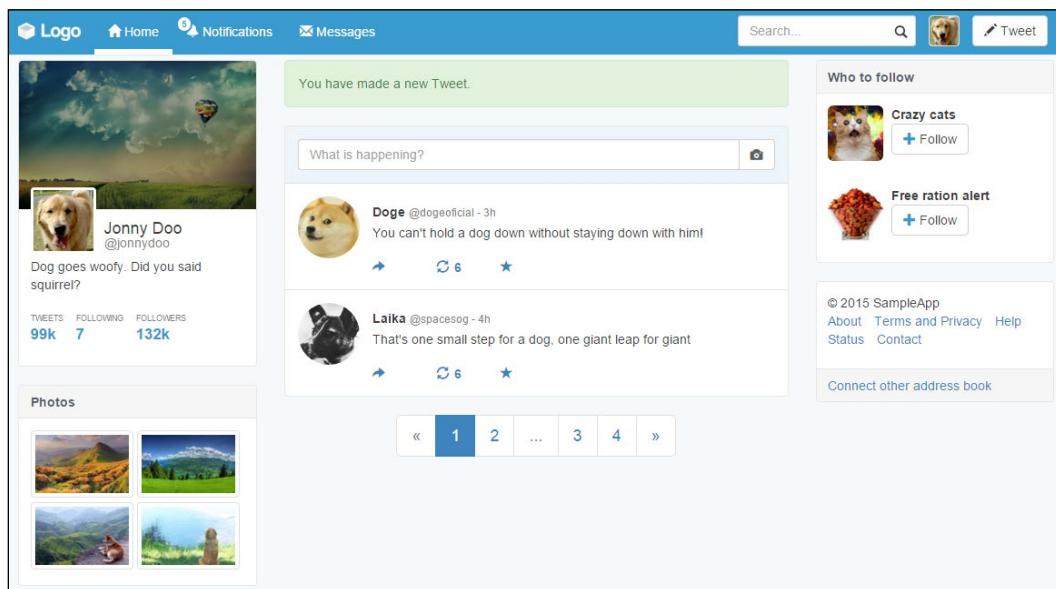
Alerts in our web app

In the last chapter, we did almost everything related to page components. Now we will create some components that interact with the user. To start this, we will introduce alerts, which are very common components of every web app.

In order to learn about alerts, we should create some of them. The pattern for creation is pretty simple; just remember to import Bootstrap JavaScript as we have been doing all throughout the book.

The main class needed to create alerts is `.alert`. You can just follow this class with some other, regarding the type of alert, such as `.alert-success` for a success message. There are other classes available as well, such as `.alert-info` and `.alert-danger`. Just replace the suffix of `.alert` with the one that you want to use.

It's time to create our first alert! Keeping the same code of the web app from the last chapter, right before `div#main`, you must have your `ol.breadcrumb`. Replace `ol.breadcrumb` with your `.alert`, like what is shown in this screenshot:



The HTML code for creating this alert is really simple:

```
<div class="alert alert-success" role="alert">  
  You have made a new Tweet.  
</div>
```

As mentioned before, just create an element with the `.alert` class in combination with the state of the alert, `.alert-success` in this case.



Why do we use the `role` attribute?

In the preceding example, we made use of the `role="alert"` attribute in our `.alert` component. The role attribute was incorporated into HTML 5, coming from the ARIA 1.0 specification. The reason for using that is to keep the semantics for different items, for example, in this case, where we used a common `<div>` to describe a more semantic element that is an alert.

Dismissing alerts

Bootstrap is incredible! Did you realize that? We created an alert with just three lines of code!

Well, another reason to think about that is to create dismissible alerts. Just add the highlighted line to the alert component and you will get the expected result:

```
<div class="alert alert-success" role="alert">
  <button type="button" class="close" data-dismiss="alert" aria-
  label="Close"><span aria-hidden="true">&times;</span></button>
  You have made a new Tweet.
</div>
```

This will create a close button that will dismiss the component using the `data-dismiss="alert"` attribute. Refresh the web page and you will see the alert like this:

You have made a new Tweet.

X

Customizing alerts

Now, it's time for us to create our recipe for the alert. We have two tasks: add a title to `.alert` and use the links inside it.

First, create a heading element inside the alert:

```
<div class="alert alert-success" role="alert">
  <button type="button" class="close" data-dismiss="alert" aria-
  label="Close"><span aria-hidden="true">&times;</span></button>
  <h3>Tweet alert</h3>
  You have made a new Tweet.
</div>
```

Then, adjust the CSS for the heading inside the alert:

```
.alert h3 {  
  margin: 0 0 1rem;  
  font-size: 1.4em;  
}
```

The final result of adding the title must be like what is shown in this screenshot:



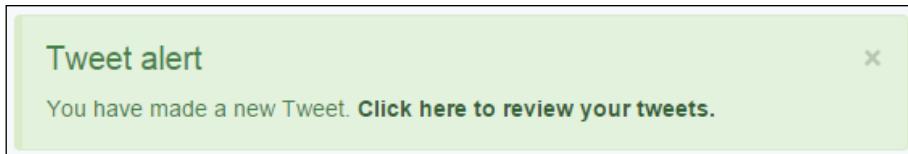
For the second task, we have to add some links inside the component. Bootstrap can give us a little shortcut for this using the `.alert-link` class in the link. The class will give the correctly matching color for the link in response to the kind of the alert shown.

Therefore, the HTML code is simple:

```
<div class="alert alert-success" role="alert">  
  <button type="button" class="close" data-dismiss="alert" aria-  
  label="Close"><span aria-hidden="true">&times;</span></button>  
  <h3>Tweet alert</h3>  
  You have made a new Tweet.  
  <a href="#" class="alert-link">Click here to review your  
  tweets.</a>  
</div>
```

To finish our first alert usage, let's just add one last fancy thing in the CSS, refresh the browser after that, and check the final result, as shown in the next screenshot:

```
.alert {  
  border-left-width: 0.5rem;  
}
```



Waiting for the progress bar

Progress bars are very useful in web applications in cases where, for example, you need to wait for an action to be sent to the server while maintaining a feedback for the user that something is being done in the background.

For instance, we can create a progress bar to present the user that a new tweet is being posted. Likewise, other scenarios can suit well for a progress bar, for example, when you are uploading a file on the server or when the web client is loading some information.

To exemplify this, we will create another alert that will contain a progress bar inside for a new tweet post feedback, subliminally saying "Hey, wait until I finish my task!"

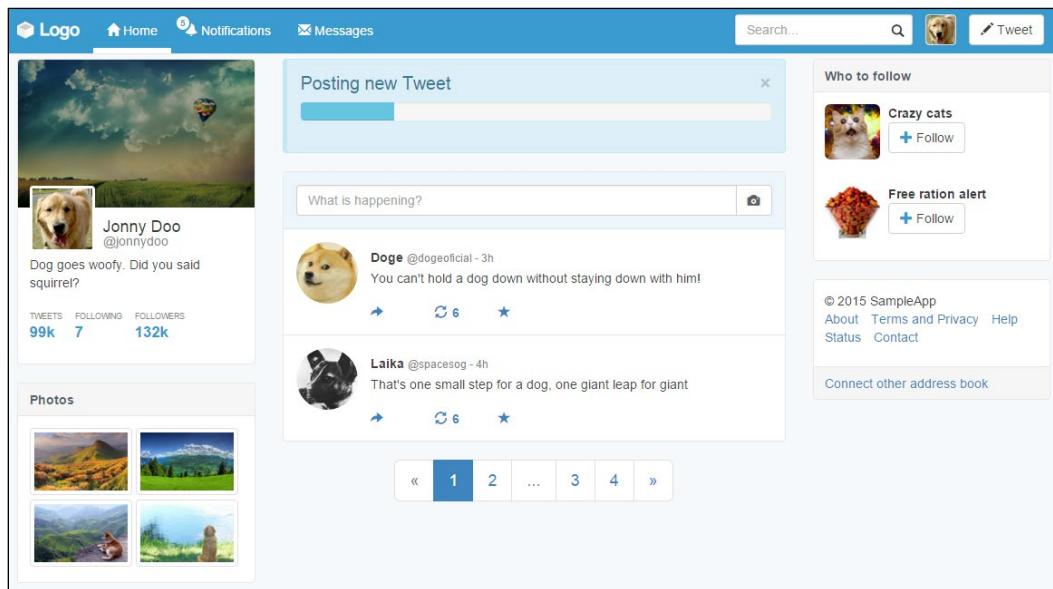
We replace the `.alert` code that we just created with the new one presented here:

```
<div class="alert alert-info" role="alert">
  <button type="button" class="close" data-dismiss="alert" aria-
label="Close"><span aria-hidden="true">&times;</span></button>
  <h3>Posting new Tweet</h3>
</div>
```

This will produce a blue alert using the colors from `.alert-info`. In the new element, create the following code for the progress bar:

```
<div class="alert alert-info" role="alert">
  <button type="button" class="close" data-dismiss="alert" aria-
label="Close"><span aria-hidden="true">&times;</span></button>
  <h3>Posting new Tweet</h3>
  <div class="progress">
    <div class="progress-bar progress-bar-info" role="progressbar"
aria-valuenow="25" aria-valuemin="0" aria-valuemax="100"
style="width: 25%">
      </div>
    </div>
  </div>
```

The result of the progress bar is shown in the next screenshot. Let's understand each part of the new component:



We created a `div.progress` inside our alert component, which is the gray rectangle to be filled during the progress. Inside it, we have another tag, `div.progress-bar`, to create the inside filler that contains the `.progress-bar-info` class to make the bar blue, following the `.info` contextual color.

The progress bar also has some `aria-*` attributes and its size is set by the `width: 25%` style. To change its size, just change the `width` style.

Progress bar options

Just like alerts, progress bars have the same contextual colors of Bootstrap. Just use the `.progress-bar-*` prefix while using the suffix of one of the contextual colors. It is also possible to apply stripe marks to the progress bar with the `.progress-bar-striped` class:

```
<div class="alert alert-info" role="alert">
  <button type="button" class="close" data-dismiss="alert" aria-
  label="Close"><span aria-hidden="true">&times;</span></button>
  <h3>Posting new Tweet</h3>
  <div class="progress">
```

```
<div class="progress-bar progress-bar-info progress-bar-striped
active" role="progressbar" aria-valuenow="25" aria-valuemin="0"
aria-valuemax="100" style="width: 25%">
</div>
</div>
</div>
```

Finally, you can also animate the strip using the `.active` class in conjunction with the `.progress-bar-striped` class. Check out the next screenshot to see the result of addition of the classes:



Animating the progress bar

Now we have a good opportunity to use the CSS `@keyframes` animations.

If you check out the CSS code when you add the `.progress-bar-striped` and `.active` classes, Bootstrap will load the following animation:

```
.progress-bar.active {
    animation: progress-bar-stripes 2s linear infinite;
}
```

This animation applies to the CSS selector, the `@keyframe` defined at `progress-bar-stripes`:

```
@keyframes progress-bar-stripes {
    from {
        background-position: 40px 0;
    }
    to {
        background-position: 0 0;
    }
}
```

This means that the striped background will move from a position of 40 pixels to 0, repeating it every 2 seconds.

Well, nice to meet you `progress-bar-stripes`, but I can do another animation! Create the following `@keyframe`:

```
@keyframes w70 {
  from { width: 0%; }
  to { width: 70%; }
}
```

The goal of this key frame is to change the width of our progress bar from 0% to 70% of the maximum when the page loads. Now apply the new animation to `.progress-bar.active`:

```
.progress-bar.active {
  animation: w70 1s ease forwards,
    progress-bar-stripes 2s linear infinite;
}
```

So, our animation will last 1 second and execute just once, since we defined it to be just `forwards`. Note that we must override the animations, so after the new animation, which is `w70`, add the current animation, which is `progress-bar-stripes`. Refresh the page and see this fancy effect.

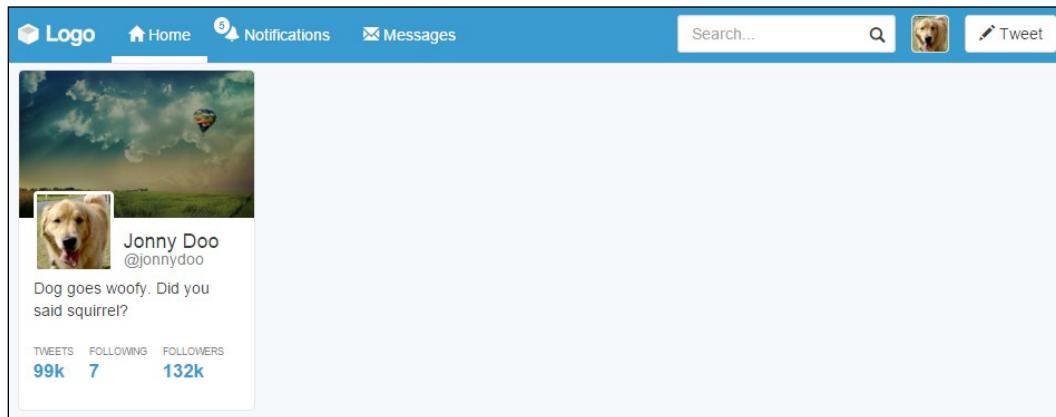
Creating a settings page

Moving on with our web app example, it is time to create a settings page for the application. We have already created a link for this in the navigation bar, inside the button group. Do you remember?

So, in the same folder as that of the web app HTML file, create another one named `settings.html` and update the link at the navigation bar:

```
<div id="nav-options" class="btn-group pull-right hidden-xs">
  <button type="button" class="btn btn-default dropdown-toggle thumbnail" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
    
  </button>
  <ul class="dropdown-menu">
    <li><a href="#">Profile</a></li>
    <li><a href="settings.html">Setting</a></li>
    <li role="separator" class="divider"></li>
    <li><a href="#">Logout</a></li>
  </ul>
</div>
```

In this page, we use the same template that we used in the web application, copying the navigation bar, the grid layout, and the left-hand side `#profile` column. So, the settings page will look like this:



Now we will add some content to this page. Our main goal here is to use some other navigation Bootstrap components that will be handy for us.

Pills of stack

The first navigation menu that we will use here is Pills, using the vertical stack option. Pills are a Bootstrap component used to create menus that can be horizontal or vertical. Many web apps use them for side menus, just as we will soon do.

The basic usage for navigation components is the use of the `.nav` class followed by another one regarding the navigation option. In our case, we will use the `.nav-pills` class to create the desired effect in conjunction with the `.nav-stacked` class to vertically stack the pills. Create a `.card` element just after `#profile` and add the code related to the pills navigation:

```
<div id="profile-settings" class="card">
  <ul class="nav nav-pills nav-stacked">
    <li role="presentation" class="active">
      <a href="#">
        Account
        <span class="glyphicon glyphicon-chevron-right pull-right" aria-hidden="true"></span>
      </a>
    </li>
```

```
<li role="presentation">
  <a href="#">
    Security
    <span class="glyphicon glyphicon-chevron-right pull-right"
aria-hidden="true"></span>
  </a>
</li>
<li role="presentation">
  <a href="#">
    Notifications
    <span class="glyphicon glyphicon-chevron-right pull-right"
aria-hidden="true"></span>
  </a>
</li>
<li role="presentation">
  <a href="#">
    Design
    <span class="glyphicon glyphicon-chevron-right pull-right"
aria-hidden="true"></span>
  </a>
</li>
</ul>
</div>
```

In the CSS file, also add the following rule:

```
.row .card + .card {
  margin-top: 2rem;
}
```

With this, every `.card` element followed by another `.card` element will automatically create a margin, because of the use of the `+` selector. Open the settings page, and you will see the result like what is shown in the following screenshot:



Basically, we created a list with the mentioned classes: `.nav`, `.nav-pills`, and `.nav-stacked`. The list is composed of four items, containing a link and a left arrow icon. To place the arrows at the right-hand side, we again used the `.pull-right` helper together with the `.glyphicon-chevron-right` icon. In the first item, we added the `.active` class, which turned this item blue and changed the colors of both the text and the icon.

This is almost awesome! It just needs some adjustments in the CSS to look perfect. Let's use some `:nth-child` selectors to style some elements based on their position, such as the first item in the menu (`:first-child`) and the last item (`:last-child`). Add the following CSS:

```
#profile-settings .nav-stacked li {  
    border-bottom: 1px solid #e5e5e5;  
    margin: 0;  
}  
  
#profile-settings .nav-stacked a {  
    border-radius: 0;  
}  
  
#profile-settings .nav-stacked li:first-child a {  
    border-radius: 0.4rem 0.4rem 0 0;  
}  
  
#profile-settings .nav-stacked li:last-child {  
    border-bottom: 0;  
}  
  
#profile-settings .nav-stacked li:last-child a {  
    border-radius: 0 0 0.4rem 0.4rem;  
}
```

We just removed unwanted margins and the border radius, while adding a border bottom to all elements in the list, except the last one. This is because of the `#profile-settings .nav-stacked li:last-child` selector, where we a specific rule only for the last item element.

We also changed `border-radius` for the first element and the last element to create a rounded border in the pill list. The `.nav-pill` menu will appear like what is shown in the next screenshot:



Tabs in the middle

In the `#main` column, we must create a tab option with the settings content. Bootstrap also offers a tabs component with the navigation components. First, we will work with the markup and CSS style and use it with JavaScript.

Therefore, inside the `#main` tag, place the following markup, corresponding to the Bootstrap tabs:

```
<ul id="account-tabs" class="nav nav-tabs nav-justified">
  <li role="presentation" class="active">
    <a href="#account-user">User info</a>
  </li>
  <li role="presentation">
    <a href="#account-language">Language</a>
  </li>
  <li role="presentation">
    <a href="#account-mobile">Mobile</a>
  </li>
</ul>
```

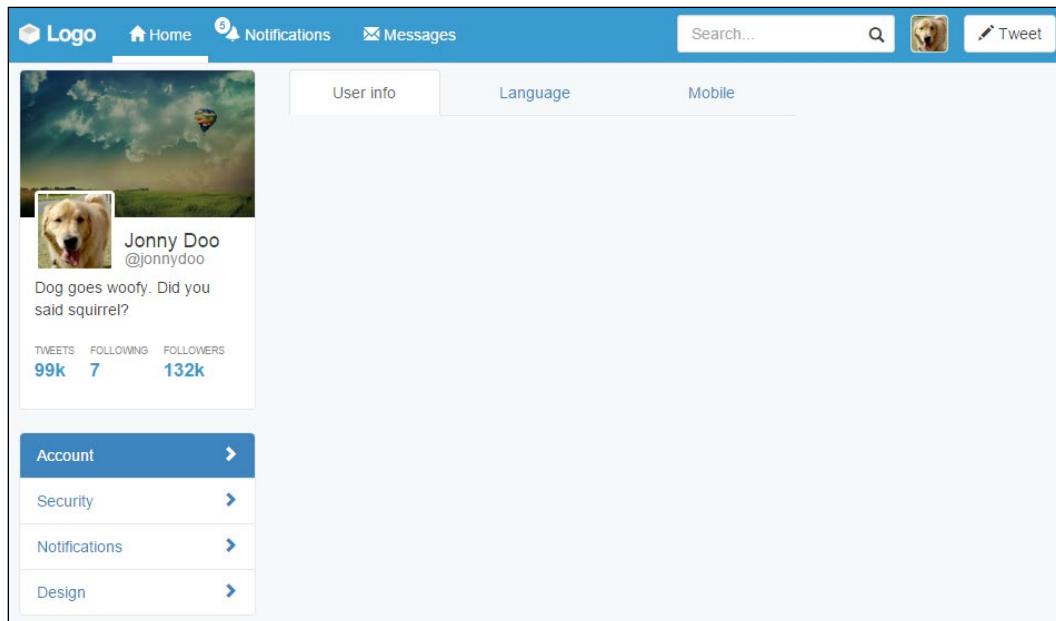
Just like the Pills, to use tabs, create a list with the `.nav` and `.nav-tabs` classes. We used the `.nav-justified` class as well to make the tabs equally distributed over the component.

Moreover, we populated the list with some items. Each link of the item contains an identifier, which will be used to link each tab to the corresponding content. Keep that information at the moment.

Let's add some CSS rules to keep the same border colors that we are using:

```
#account-tabs > li {  
    border-bottom: 0.1rem solid #e5e5e5;  
}  
  
#account-tabs a {  
    border-bottom: 0;  
}  
  
#account-tabs li.active {  
    border-bottom: 0;  
}
```

Refresh the web page and you should see it like this:



Adding the tab content

To add the tab content, we use the named *tab panes*. Each tab must be placed in an element with the `.tab-pane` class and have the corresponding identifier in the `tab` component. After the tab list, add the HTML code for the tab panes:

```
<ul id="account-tabs" class="nav nav-tabs nav-justified">
  <li role="presentation" class="active">
    <a href="#account-user">User info</a>
  </li>
  <li role="presentation">
    <a href="#account-language">Language</a>
  </li>
  <li role="presentation">
    <a href="#account-mobile">Mobile</a>
  </li>
</ul>

<div class="tab-content">
  <div role="tabpanel" class="tab-pane active" id="account-user">
    User info tab pane
  </div>
  <div role="tabpanel" class="tab-pane" id="account-language">
    Language tab pane
  </div>
  <div role="tabpanel" class="tab-pane" id="account-mobile">
    Mobile tab pane
  </div>
</div>
```

Refresh the web page and you will see only the content of `#account-user` appearing, although clicking on other tabs will not cause any switch between them. This is because we have not initialized the component through JavaScript yet.

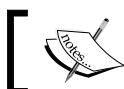
Using the Bootstrap tabs plugin

We can initialize the tabs with simple JavaScript code, like what is presented next. However, we will do that in a smarter way using data markup:

```
$( '#account-tabs a' ).click(function (e) {
  e.preventDefault()
  $(this).tab('show')
})
```

For data markup, we must add the `data-toggle="tab"` attribute on the link elements, inside the list. Change the list markup for the following:

```
<ul id="account-tabs" class="nav nav-tabs nav-justified">
  <li role="presentation" class="active">
    <a href="#account-user" data-toggle="tab">User info</a>
  </li>
  <li role="presentation">
    <a href="#account-language" data-toggle="tab">Language</a>
  </li>
  <li role="presentation">
    <a href="#account-mobile" data-toggle="tab">Mobile</a>
  </li>
</ul>
```



If you are using pills, add the `data-toggle="pill"` attribute, just as we did for tabs.



Refresh the web browser and switch between the tabs. It works like a charm! To make it even fancier, add the `.fade` class to each `.tab-pane` to create a fade effect when changing tabs.

Creating content in the user info tab

Inside the `.tab-pane` identified by `#account-user`, let's add some content. Since it is a configuration menu, we will create a form to set the user settings. Again, we will work with some forms in Bootstrap. Do you remember how to use them?

In the form, we will have three inputs (name, username, and e-mail) followed by a button to save the changes. We will use the `.form-horizontal` class to make each form input next its label and stacked by each other. So, place the following code inside the `#account-user` element:

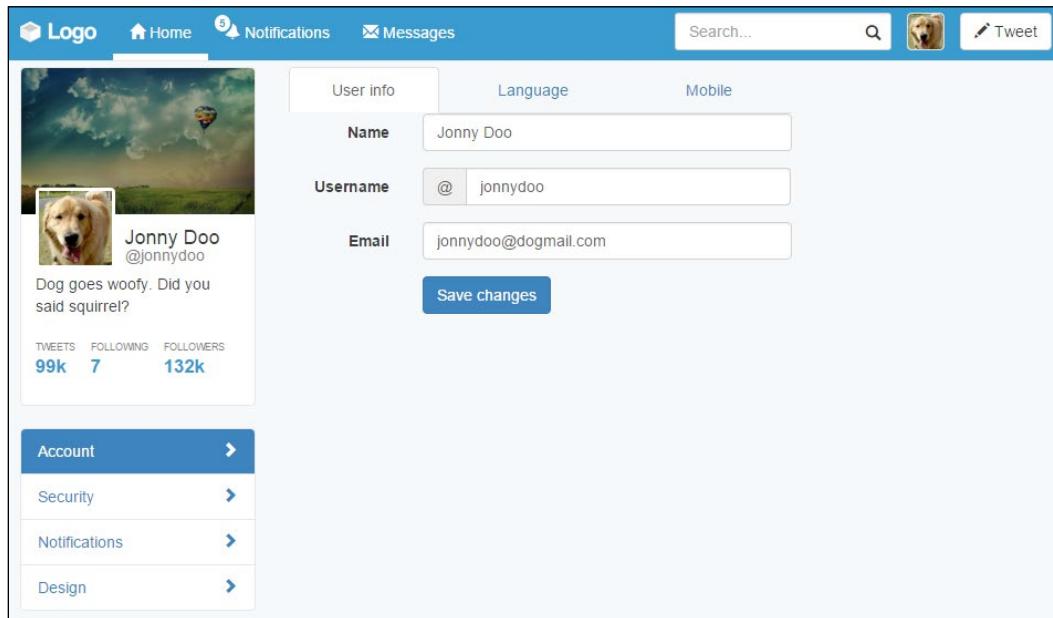
```
<div id="account-user" role="tabpanel" class="tab-pane active" >
  <form class="form-horizontal">
    <div class="form-group">
      <label class="col-sm-3 control-label">Name</label>
      <div class="col-sm-9">
        <input type="text" class="form-control" value="Jonny Doo">
      </div>
```

```
</div>
<div class="form-group">
    <label class="col-sm-3 control-label">Username</label>
    <div class="col-sm-9">
        <div class="input-group">
            <div class="input-group-addon">@</div>
            <input type="text" class="form-control"
value="jonnnydoo">
        </div>
    </div>
</div>
<div class="form-group">
    <label class="col-sm-3 control-label">Email</label>
    <div class="col-sm-9">
        <input type="email" class="form-control"
value="jonnnydoo@dogmail.com">
    </div>
</div>
<div class="form-group">
    <div class="col-sm-offset-3 col-sm-9">
        <button type="submit" class="btn btn-primary">
            Save changes
        </button>
    </div>
</div>
</form>
</div>
```

Break through the code, we set the labels to fill a quarter of the form row, while the input fills the rest. We again used the input groups to add the @ sign before the username field.

Each label has the `.col-sm-3` class and the input the `.col-sm-9` class. As we said, the forms respect the grid system layout, so we can apply the same classes here. We also added the `.col-sm-offset-3` class to the submit button for a correct offset.

Right now, the page should look like the following screenshot. Isn't it looking nicer?



CSS turn! We must make it look prettier, so let's play with some paddings. However, before anything, add the class to `.card` to the `.tab-content`. Next, let's fix the borders and margins in `.tab-content` by adding some padding and some negative margin, as follows:

```
#main .tab-content {  
    padding: 2em;  
    margin-top: -0.1rem;  
}
```

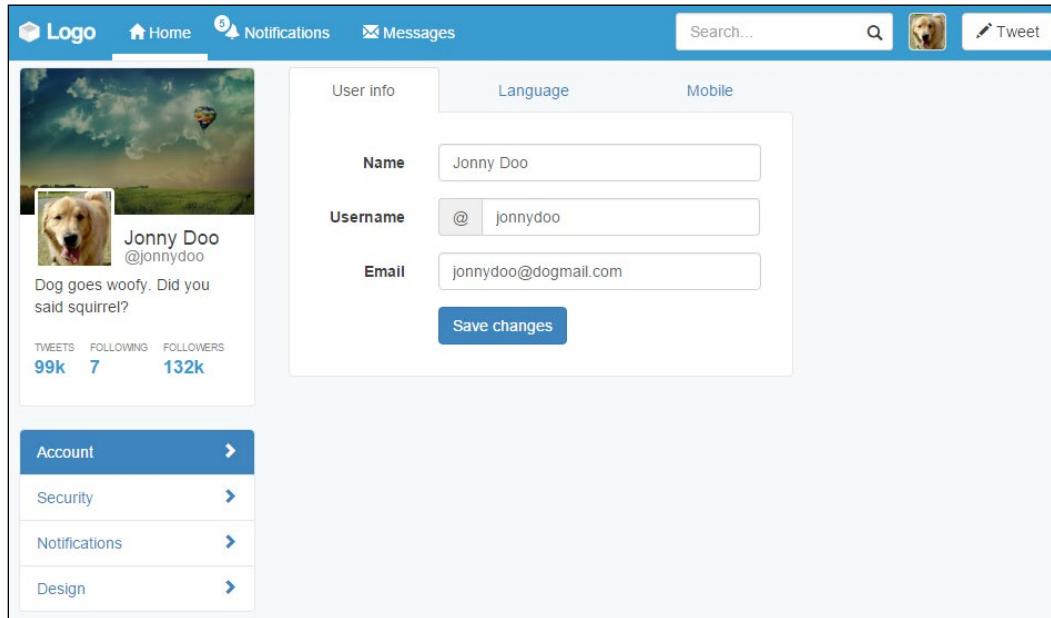
Then, remove the border from the tab list and change the `z-index` of the tabs:

```
#account-tabs > li {  
    border-bottom: 0;  
}  
  
#account-tabs {  
    position: relative;  
    z-index: 99;  
}
```

Finally, remove the margin from the save changes button using another nth recipe, as follows:

```
#main .form-horizontal .form-group:last-child {
    margin-bottom: 0;
}
```

Nice! Refresh the web app and see the result, like this:



With these changes, we have made the form from `.tab-content` look like a card. We also added some margins and padding to correct the designs. Lastly, with the margin top and z-index, we make the selected tab blend correctly with the content.

The stats column

To finalize the settings page, we must fill the right column of the web app. To do this, we will create a menu that presents some general statistics about the user.

In order to do that, we will use the Bootstrap List group component. List group is powerful in Bootstrap as it can seem similar to Bootstrap Cards, but it has more specific options for menu list generation, such as headers, disabled items, and much more.

So let's start doing this! Create the HTML markup at the #right-content element:

```
<div id="right-content" class="col-md-3 hidden-sm hidden-xs">
  <ul class="list-group">
    <li class="list-group-item list-group-item-info">
      Dog stats
    </li>
    <li class="list-group-item">
      Number of day rides
    </li>
    <li class="list-group-item">
      Captured mice
    </li>
    <li class="list-group-item">
      Postmen frightened
    </li>
    <li class="list-group-item">
      Always alert badge
    </li>
  </ul>
</div>
```

It is a simple list and we just added the `.list-group` class to the list, while adding the `.list-group-item` class for each item in the list. For the first item, we used the `.list-group-item-info` contextual color class to make it blue, according to the Bootstrap contextual colors.

Before we continue, we have to change the colors of the borders from list group to respect the same border color that we have been using:

```
.list-group-item {
  border-color: #e5e5e5;
}
```

The result that you should see right now is like the one presented in the next screenshot. List groups is a very handy component that we can use to achieve almost the same effect that we got using Pills, but here we are using fewer CSS rules.

Labels and badges

Labels and badges are Bootstrap components that can easily highlight some text or information. They can be used in any place, as we did in the navigation bar, on the notification item.

Just like most of Bootstrap's components, labels follow the contextual colors of Bootstrap, while badges do not have this option. To consolidate their utilization, let's add some of them to the right menu, regarding the statistics, because after all we want to know our stats!

It will be pretty easy since we have worked with this before. For the three first items, we will use labels, and for the last one, we will use a badge with an icon. Just add the highlighted code to the current `.list-group`:

```
<div id="right-content" class="col-md-3 hidden-sm hidden-xs">
  <ul class="list-group">
    <li class="list-group-item list-group-item-info">
      Dog stats
    </li>
    <li class="list-group-item">
      Number of day rides
      <span class="label label-success">3</span>
    </li>
```

```
<li class="list-group-item">
    Captured mice
    <span class="label label-danger">87</span>
</li>
<li class="list-group-item">
    Postmen frightened
    <span class="label label-default">2</span>
</li>
<li class="list-group-item">
    Always alert badge
    <span class="badge glyphicon glyphicon-star" aria-hidden="true">
    </span>
</li>
</ul>
</div>
```

We must pay attention to some points here. The first is that we used the `.label` class together with the contextual color class, like this: `.label-*`. Here, `*` is the name of the contextual class.

The second thing is that if you refresh your web page right now, you will see that Bootstrap does have a CSS for aligning all badges on the right, but it does not apply the same rule for labels. So, add the next CSS:

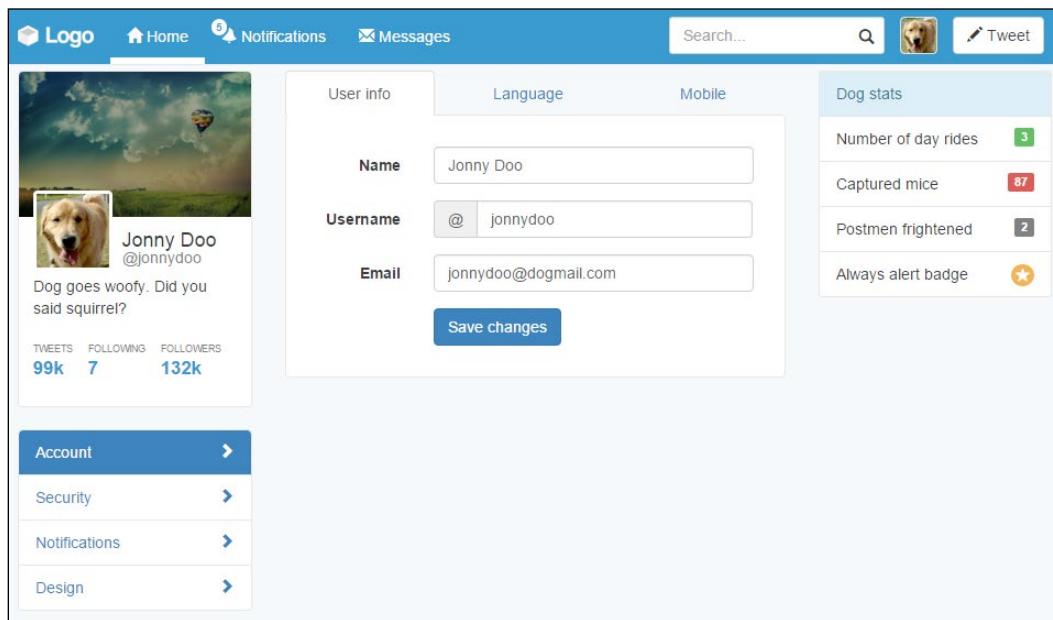
```
#right-content .list-group .label {
    float: right;
}
```

The third point is that we used badges and icons in the same span. This is possible, and then we just need to adjust the CSS render:

```
#right-content .list-group .label {
    padding: 0.3rem 0.6rem;
}

#right-content .list-group .badge.glyphicon-star {
    background-color: #f0ad4e;
    padding: 0.4rem;
    padding-left: 0.5rem;
}
```

By adding these rules, we adjusted the padding for the labels, adjusted the padding for the badge, and correctly set the yellow background for the star badge. See the result in the browser; it must look like this:



Cool! We've done another page for our web app. The stats menu looks cute. We will not cover the other page settings menus in this book, since they follow almost the same structure that we showed in this example. Try doing that as homework.

Summary

In this chapter, we covered a bunch of Bootstrap components. Now you can say that you know all of the most important Bootstrap components, even the new ones in version 4. This chapter was a challenging one, since you had to learn so many details about different components, so congratulations!

First, we started working with alerts. We only used the component alert with data attributes, but in the coming chapters, we will use it as a JavaScript plugin, with more options and animations. Don't worry! This is a glimpse of future chapters.

Next, you learned about progress bars and made use of some customization, and using animations by CSS. With that, we achieved a cool result at the end and are now able to use progress bars when needed.

After that, we switched to the settings page. There, taking advantage of the same web application structure, we changed the layout. We did this by creating a menu using the pills navigator, and created a main component that used tabs.

For tabs, that was the first time we presented how to use a Bootstrap plugin by using JavaScript, but do not worry about this either. We will go deep into this subject in the next chapter.

To finish the chapter, we worked with other component, called list group. This component offers some more capabilities for creating stacked menus.

Inside the items, we studied the use of labels and badges, which are nice things that Bootstrap offers to us.

In the next chapter, we will start working with some Bootstrap JavaScript plugins. We will go back to the main web application page and play with some posts on the timeline. It will be fun. See you there!

8

Working with JavaScript

The whole Internet would not have been the same without JavaScript, and so is Bootstrap. The JavaScript plugins from Bootstrap account for a chunk of Bootstrap's success. By having them, Bootstrap has allowed all of us to use modals, alerts, tooltips, and other plugins out of the box.

Therefore, the main focus of this chapter is to explain the main Bootstrap JavaScript plugins by using them in our web application. In the previous chapters, we used a few plugins. The purpose of this chapter is to go deep into this subject. The key points that we will cover now are:

- General usage of Bootstrap JavaScript plugins
- Data attributes
- Modals
- Tooltips
- Popover
- Affix

Understanding JavaScript plugins

As I said, Bootstrap offers a lot of JavaScript plugins. They all come together when we download the framework, and all of them are ready for use when the `bootstrap.js` file is loaded in HTML, although each plugin can be downloaded individually as well from the Bootstrap website.



Minifying JavaScript

In the production stage, you can use the minified version of Bootstrap JavaScript. We are not using that right now for learning purposes, but it is recommended that you use the minimal version when you go live.

The library dependencies

While we were setting up our development environment, we spoke about the need to import the jQuery library. Actually, jQuery is now the only required external dependency for Bootstrap.

Check out the `bower.json` file in the Bootstrap repository for further information about dependencies.



Bower

Bower is a package management control system for client-side components. To use Bower, you must have both Node and npm installed. Bower was developed by the developers of Twitter.

Data attributes

HTML5 introduced the idea of adding custom attributes to document tags in order to store custom information. Therefore, you can add an attribute to a tag with the `data-*` prefix, retrieve the information in JavaScript, and not get started with some plugin in your browser. An overwhelming majority of web browsers do support the use of custom data attributes.

With that ideology, Bootstrap implemented all the plugins to be used with just data attributes. This goes towards the framework idea to increase the speed of development and prototyping. This is because you can make use of plugins without typing JavaScript code.

To control that methodology, Bootstrap implemented an API so that you can access all plugins through only data attributes. Sometimes, however, you may want to turn off access through the API. To do so, insert the following command at the beginning of your JavaScript code:

```
$(document).off('.data-api');
```

To disable the API for some specific plugins, prepend the plugin namespace. For instance, to disable the alerts API, type this:

```
$(document).off('.alert.data-api');
```

Bootstrap JavaScript events

There is a set of events that Bootstrap produces for each plugin. They are triggered usually before and after the event starts. To exemplify this, let's say that we have a Bootstrap modal (you will learn about using modals in this chapter; don't worry) and we will call it to open by JavaScript:

```
$('#some-modal').modal();
```

When this happens, Bootstrap triggers the events called `show.bs.modal` and `shown.bs.modal`. The first one is called before the *open modal* call and the other is called after the action. Let's say we want to customize our modal before it is shown. To do this, we must use the first event:

```
$('#some-modal').on('shown.bs.modal', function(e) {
    // do some customization before shown
});
```

The events can be used for all plugins. Just change the namespace (in this case, `.modal` is the namespace) to achieve the result.

Awesome Bootstrap modals

It's time to learn how to use modals! Modals are really present nowadays in web development and Bootstrap plugins, for that is really complete and easy to use. To use it, let's go back to our main web application page, the one containing the feeds.

First, we add the `.hide` helper class to the `div.alert` that we created at the `#main` column. We will play with alerts later. Now, go to the `#tweet` button on the navigation bar. We want to open a modal for tweets when clicking on this button. So, add the markup to the element:

```
<!-- modal launch button -->
<button id="tweet" class="btn btn-default pull-right visible-xs-block"
data-toggle="modal" data-target="#tweet-modal">
    <span class="glyphicon glyphicon-pencil" aria-hidden="true"></span>
    Tweet
</button>
```

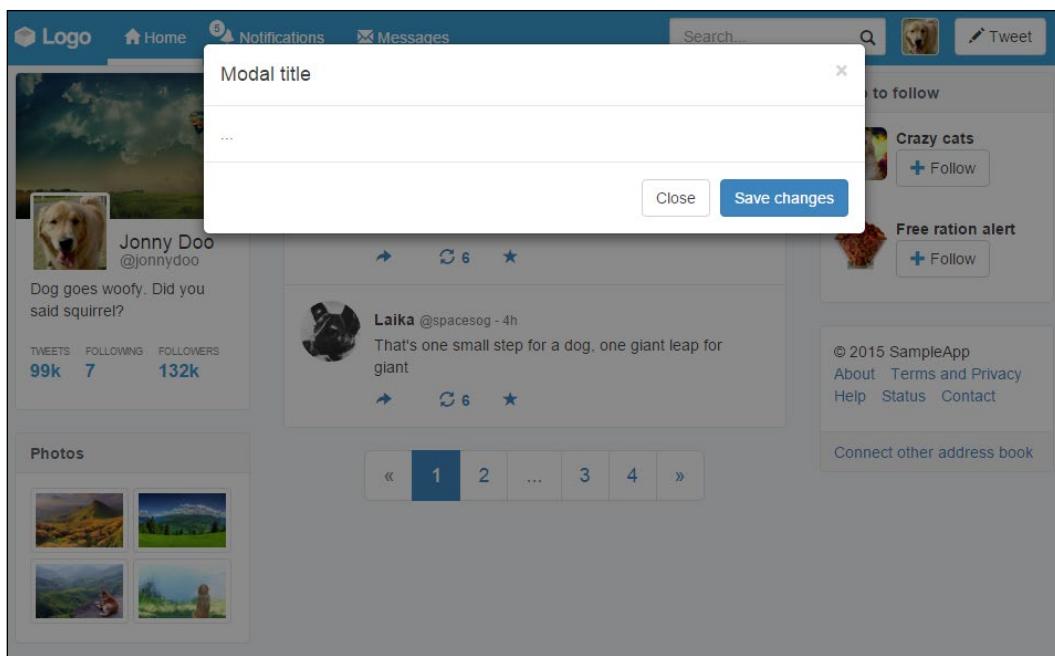
What we did is the call to open a modal, recognized by the `#tweet-modal` ID and the `data-toggle="modal"` data attribute. We could also have done that via JavaScript with this code:

```
$( '#tweet-modal' ).modal();
```

Create the modal by adding the next HTML code. We have created all the modals at the end of our HTML code, outside of all Bootstrap elements and right before the loading of the JavaScript libraries:

```
<div class="modal fade" id="tweet-modal" tabindex="-1"
role="dialog">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <button type="button" class="close" data-dismiss="modal"
aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
        <h4 class="modal-title">Modal title</h4>
      </div>
      <div class="modal-body">
        Modal content
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-default" data-
dismiss="modal">
          Close
        </button>
        <button type="button" class="btn btn-primary">
          Save changes
        </button>
      </div>
    </div>
  </div>
</div>
```

Reload the web application, click on the **Tweet** button and see the magic happen! This code is a little more complex, so let's explain each part separately. The following screenshot shows what the first draft of our modal looks like. Now let's understand what we did.



Modal general and content

The first tag used to initiate a modal is a `<div>` with the `.modal` class. Note that we also added the `.fade` class to create the effect of fade in and fade out when the modal appears and disappears.

Inside the `.modal` element, we created two more nested tags. The first one is `.modal-dialog`, which will wrap all of the modal dialog. Inside it, we created `.modal-content`, which will hold the content of the modal itself.

The modal header

Next, inside `.modal-content` we have the `.modal-header` element. In the modal header, we can add some title information about the modal. In our example, we have also added a close button that hides the modal using a `data-dismiss="modal"` data attribute.

The modal body

The modal body is where you should place the main content of the modal. A cool feature inside the modal is the ability to use scaffolding.

To use the grid system inside the modal body, you do not need to create a container. Just create a `.row` inside `.modal-body` and start adding columns, as shown in the following example in bold:

```
<div class="modal fade" id="tweet-modal" tabindex="-1"
role="dialog">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <button type="button" class="close" data-dismiss="modal"
aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
        <h4 class="modal-title">Modal title</h4>
      </div>
      <div class="modal-body">
        <div class="row">
          <div class="col-sm-2">Use</div>
          <div class="col-sm-4">the</div>
          <div class="col-sm-6">grid system</div>
        </div>
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-default" data-
dismiss="modal">
          Close
        </button>
        <button type="button" class="btn btn-primary">
          Save changes
        </button>
      </div>
    </div>
  </div>
</div>
```

The modal footer

At the end of the modal, you can create a `.modal-footer` element to place some other components, such as buttons, as we did in the previous example.

Creating our custom modal

Now that you have learned how to use a Bootstrap modal, let's customize it for our example. First, let's add some content inside our `.modal-body` and edit `.modal-header` and `.modal-footer` a little:

```
<div class="modal fade" id="tweet-modal" tabindex="-1"
role="dialog">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <button type="button" class="close" data-dismiss="modal"
aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
        <h4 class="modal-title">Dog a new tweet</h4>
      </div>
      <div class="modal-body">
        <textarea class="form-control" rows="4" placeholder="What you
want to bark?" maxlength="140"></textarea>
      </div>
      <div class="modal-footer">
        <span class="char-count pull-left" data-max="140">140</span>
        <button type="button" class="btn btn-default" data-
dismiss="modal">
          Close
        </button>
        <button type="button" class="btn btn-primary">
          Tweet
        </button>
      </div>
    </div>
  </div>
</div>
```

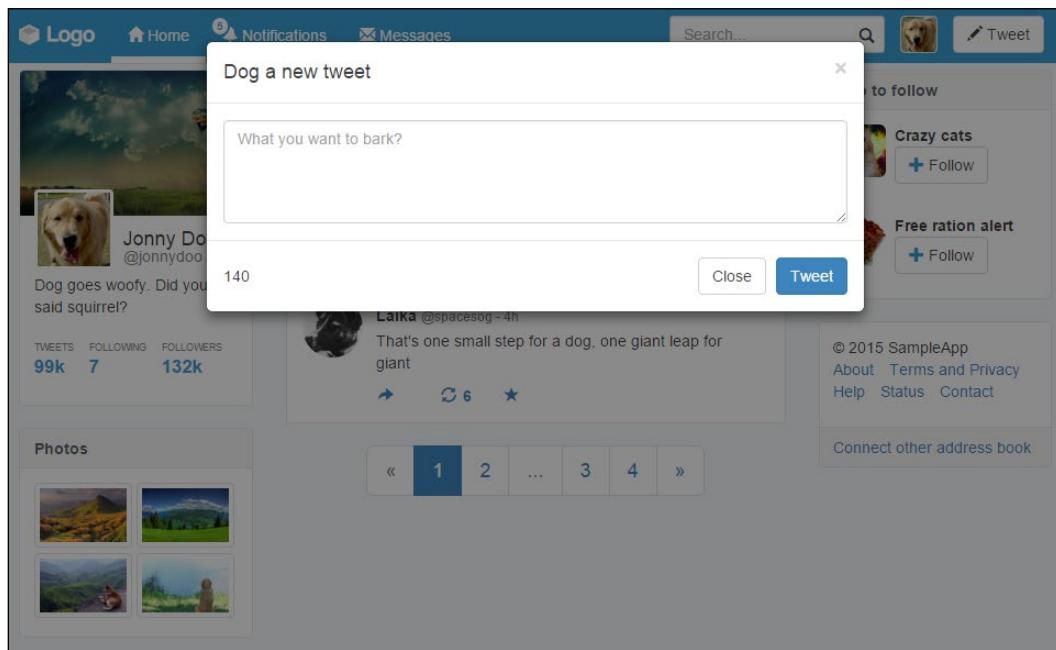
Here, we added a heading to `.modal-header`, a `textarea` in `.modal-body` and a `` element with the `.char-count` class in the footer.

The goal here is to type a tweet inside the `textarea` element and update the character count in the footer to show how many characters are left for the user to enter.

For styling, go to the CSS and add a style rule for .char-count:

```
#tweet-modal .char-count {  
    padding: 0.7rem 0;  
}
```

Refresh the web browser and see the result of the tweet modal, as shown in the following screenshot. Now, we need to add some JavaScript to count the number of remaining characters to tweet.



So, for the JavaScript for the character count, open (or create it if you do not have it yet) the `main.js` file. Ensure that you have the document ready to create the script, by having the following code in your file:

```
$(document).ready(function() {  
    // add code here  
});
```

Then, we must create a function that updates the remaining characters each time a letter is typed. Therefore, let's create an event handler for `keyup`.

The keyup event came from jQuery, which has a lot of event handlers that are triggered on different actions. There are also other events such as `click`, `hover`, and so on. In this case, `keyup` will trigger when you release a key that you pressed.

The basic usage to create a bind event is from a selector, call the `.on` function passing at the first argument of the event type (in this case, `keyup`), followed by the handler (in our case, a function). Here, we have presented the JavaScript code, and the event handler is in bold so as to highlight the usage:

```
$ (document) . ready (function () {
    var $charCount, maxCharCount;

    $charCount = $('#tweet-modal .char-count')
    maxCharCount = parseInt($charCount.data('max')), 10);

    $('#tweet-modal textarea') . on ('keyup', function (e) {
        var tweetLength = $(e.currentTarget) . val () . length;

        $charCount.html (maxCharCount - tweetLength);
    });
});
```



We used the `keyup` event handler to trigger the event after the key is released and the character is typed. Therefore, the new length of `textarea` is already computed when we make a comparison.

A tool for your tip

Tooltips are a very useful component for describing in detail an element or a web page. For example, when you have an image and want to describe it further, you add a tooltip. When users hover over the image, they see further information.

In our case, we will use tooltips for the buttons present in every tweet, such as `Reply`, `Retweet`, and `Start`. This Bootstrap plugin component is pretty simple and useful in many cases. To start it, just add the markup in bold to the tweets in the middle column (`` in the `ul#feed` element):

```
<ul id="feed" class="list-unstyled">
<li>
    
    <div class="feed-post">
```

```
<h5>Doge <small>@dogeoficial - 3h</small></h5>
<p>You can't hold a dog down without staying down with
him!</p>
</div>
<div class="action-list">
    <a href="#" data-toggle="tooltip" data-placement="bottom"
title="Reply">
        <span class="glyphicon glyphicon-share-alt" aria-
hidden="true"></span>
    </a>
    <a href="#" data-toggle="tooltip" data-placement="bottom"
title="Retweet">
        <span class="glyphicon glyphicon-refresh" aria-
hidden="true"></span>
        <span class="retweet-count">6</span>
    </a>
    <a href="#" data-toggle="tooltip" data-placement="bottom"
title="Start">
        <span class="glyphicon glyphicon-star" aria-hidden="true"></
span>
    </a>
</div>
</li>

<li>
    
    <div class="feed-post">
        <h5>Laika <small>@spacesog - 4h</small></h5>
        <p>That's one small step for a dog, one giant leap for
giant</p>
    </div>
    <div class="action-list">
        <a href="#" data-toggle="tooltip" data-placement="bottom"
title="Reply">
            <span class="glyphicon glyphicon-share-alt" aria-
hidden="true"></span>
        </a>
        <a href="#" data-toggle="tooltip" data-placement="bottom"
title="Retweet">
            <span class="glyphicon glyphicon-refresh" aria-
hidden="true"></span>
            <span class="retweet-count">6</span>
        </a>
        <a href="#" data-toggle="tooltip" data-placement="bottom"
title="Star">
```

```

    <span class="glyphicon glyphicon-star" aria-hidden="true"></


```

As you can notice, by using data attributes, you just need to add three of them to make a tooltip. The first one is `data-toggle`, which says the type of toggle. In our case, it is `tooltip`. The `data-placement` attribute is concerned with the placement of the tooltip (obviously). In this case, we set it to appear at the bottom, but we can set it to `left`, `top`, `bottom`, `right`, or `auto`. Finally, we add the `title` attribute, which is not a data attribute, because HTML already has the attribute `title`, so we can call it by this attribute.

Refresh the web app in the browser, hover the icon and you will see that... nothing happens! Unlike the other plugins, the tooltip and popover Bootstrap plugins cannot be activated simply through data attributes. They did this because of some issues, so it must be initialized through a JavaScript command. Therefore, add the following line to the `main.js` file:

```

$(document).ready(function() {
    ...
    // to rest of the code
    $('[data-toggle="tooltip"]').tooltip();
});
```

The `[data-toggle="tooltip"]` selector will retrieve all the tooltip elements and start it. You can also pass some options inside while calling the `.tooltip()` start function. The next table shows some main options (to see all of them, refer to the official documentation of Bootstrap) that can be passed through JavaScript or data attributes:

Option	Type	Default	Description
<code>animation</code>	Boolean	<code>true</code>	This adds fade in and fade out animation to a tooltip.
<code>placement</code>	String or function	<code>top</code>	This is the placement position of the tooltip. The options are the same as those mentioned for the usage with data attributes (<code>top</code> , <code>bottom</code> , <code>left</code> , <code>right</code> , and <code>auto</code>). You can pass <code>auto</code> with another option, such as <code>auto left</code> , which will always show the tooltip on the left as long as it is possible, and then show it on the right.
<code>selector</code>	String	<code>false</code>	If you provide a selector, the tooltip will be delegated to the specified target.

Option	Type	Default	Description
trigger	String	hover focus	The trigger to the tooltip will be shown. The options are <code>click</code> , <code>hover</code> , <code>focus</code> , and <code>manual</code> . You can pass multiple options for trigger, unless you use <code>manual</code> , in which case you need to write a function to activate the plugin.

The tooltip plugin also has some useful methods for doing things such as showing all tooltips. You can call them using the `.tooltip()` method. As mentioned, if you want to show all tooltips, just use `$('.tooltip-selector').tooltip('show')`. The other options are `hide`, `toggle`, and `destroy`.

Pop it all over

In some cases, you may want to show more information that does not fit in a simple tooltip component. For that, Bootstrap has created popovers, which are components that create small overlays of content to show detailed secondary information.

The popover plugin is an extension of the tooltip plugin, so if you are using separate plugins, you must load both to make it work. Also, just like tooltips, popovers cannot be activated simply through data attributes. You must call them via JavaScript to make them work.

Let's use a popover in our web app example, on the right-hand-side column, the one identified by `div#who-follow`. We will add the popover to the **Follow** buttons, and for that, we need to do two things. The first one is to change the `<button>` element to an `<a>` element and then add the popover markup.

 **Why do we need to change buttons to links in popover?**
Actually, we don't have to change the buttons' markup; we will do that just because of cross-browser compatibility. There are some browsers that do not support all the functionalities, such as the click **Dismiss** option present in the popover.

First, about the `<buttons>` inside the `div#who-follow` element. Change them to `<a>` elements in the HTML. Also add the `role="button"` and `tabindex="-1"` attributes to the links to fix the issue of cross-browser compatibility:

```
<div id="who-follow" class="card">
  <div class="card-header">
    Who to follow
  </div>
```

```

<div class="card-block">
  <ul class="list-unstyled">
    <li>
      
      <div class="info">
        <strong>Crazy cats</strong>
        <a href="#" role="button" tabindex="-1" class="btn btn-default">
          <span class="glyphicon glyphicon-plus" aria-hidden="true"></span> Follow
        </a>
      </div>
    </li>
    <li>
      
      <div class="info">
        <strong>Free ration alert</strong>
        <a href="#" role="button" tabindex="-1" class="btn btn-default">
          <span class="glyphicon glyphicon-plus" aria-hidden="true"></span> Follow
        </a>
      </div>
    </li>
  </ul>
</div>

```

The code in bold refers to the changes from button to link. Now, we must add the popover markup. It is pretty simple and follows most of the data attributes presented in the tooltip plugin:

```

<div id="who-follow" class="card">
  <div class="card-header">
    Who to follow
  </div>
  <div class="card-block">
    <ul class="list-unstyled">
      <li>
        
        <div class="info">
          <strong>Crazy cats</strong>
          <a href="#" role="button" tabindex="-1" class="btn btn-default" data-toggle="popover" data-trigger="focus" title="You may want to follow">

```

```
<span class="glyphicon glyphicon-plus" aria-
hidden="true"></span> Follow
</a>
</div>
</li>
<li>
    
    <div class="info">
        <strong>Free ration alert</strong>
        <a href="#" role="button" tabindex="-1" class="btn btn-
default" data-toggle="popover" data-trigger="focus" title="You may
want to follow">
            <span class="glyphicon glyphicon-plus" aria-
hidden="true"></span> Follow
        </a>
    </div>
</li>
</ul>
</div>
```

Just like the popover, add the following line to the JavaScript code to make popovers appear:

```
$(document).ready(function() {
    ... // the rest of the JavaScript
    $('[data-toggle="popover"]').popover();
});
```

Refresh the web browser, click on the **Follow** button and see the popover appearing to the right of the button. Now we will make some changes using the options to customize it. First of all, let's create the content that will appear inside the popover and change its placement in JavaScript:

```
$(document).ready(function() {
    ... // rest of the JavaScript
    var popoverContentTemplate = ' ' +
        '' +
        '<div class="info">' +
            '<strong>Dog Breeds</strong>' +
            '<a href="#" class="btn btn-default">' +
                '<span class="glyphicon glyphicon-plus" aria-
hidden="true"></span>' +
                    'Follow' +
            '</a>' +
        '</div>';
```

```
$(' [data-toggle="popover"] ') .popover({
    placement: 'bottom',
    html: true,
    content: function() {
        return popoverContentTemplate;
    }
});
```

In the preceding code, we changed the placement of the popover to `bottom` and set the content that will appear inside the popover to be HTML with the `html: true` option. The content was provided by a function that simply returned the `popoverContentTemplate` variable.

For instance, we could have used the template in very different ways that are more optimized, but we did this to show the method of adding HTML content onto a popover via JavaScript and using a function for that. We could have called and used some options of the target clicked button inside the function by accessing the current scope in the `this` variable.

Popover events

Popovers and tooltips provide some nice events. As was said before, Bootstrap triggers some events when plugin elements appear, hide, and are inserted. To play with these, let's use the `show.bs.popover` event, which is an event that is fired immediately when the popover is `show`. In this case, we want to create an action before the popover is `show`. We want to change the text of the **Follow** button that we clicked on to **Following**, while changing the icon next to the text from a plus sign to an okay sign. We can take advantage of the `show.bs.popover` Bootstrap event to make these changes. In the JavaScript file, insert the following delegation to the popovers:

```
$(document).ready(function() {
    ... // the rest of the JavaScript code
    $(' [data-toggle="popover"] ') .on('show.bs.popover', function()
    {
        var $icon = $(this).find('span.glyphicon');

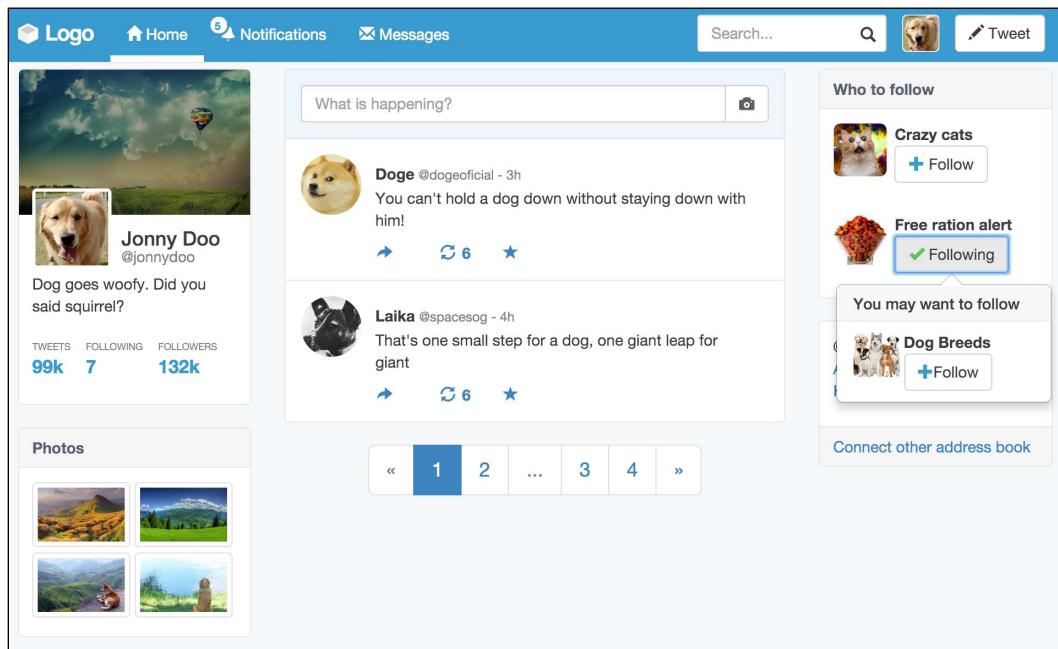
        $icon.removeClass('glyphicon-plus').addClass('glyphicon-ok');
        $(this).append('ing');
    });
});
```

The scope of this event is the element of `data-toggle`, which is the **Follow** button. We query the icon inside the button, and change it from `glyphicon-plus` to `glyphicon-ok`. Finally, we append the infinitive `ing` to **Follow**, which means that we are now following **Crazy cats** or **Free ration alert** suggestions:

To add a cherry to the pie, let's change the color of the icon from blue to green when the okay icon appears:

```
div#who-follow li .info .glyphicon-ok {  
    color: #5cb85c;  
}
```

Refresh the web browser and click on the **Follow** button. You should see something similar to this screenshot:



There are many other places where the Bootstrap events can be used. This is a nice example where we want to change the element that we are interacting with. Keep in mind to change it whenever you need some related interaction.

Making the menu affix

The affix plugin is present only in version 3 of Bootstrap (it was removed in version 4), and it aims to toggle the position of an element between fixed and relative, emulating the effect of `position: sticky`, which is not present in all browsers.

We will apply the sticky effect to the left `#profile` element although we do not have enough elements to make a scroll on our web page. Therefore, to make it simple, replicate the `` in `ul#feed` to increase the number of items in the list. Do this three times or more to make a scroll in your web browser.

In `div#profile`, add the markup related to affix:

```
<div id="profile" class="col-md-3 hidden-sm hidden-xs" data-  
spy="affix" data-offset-top="0">  
...  
// rest of the profile HTML  
</div>
```

Refresh the web browser. You will see that the affix is not working yet. Since we are making the left column with a fixed position with the affix plugin, it is removing the entire column from the grid, making the columns glitch from left to right.

So, we need a workaround for that. We must create some piece of JavaScript code using the events triggered for the plugin.

Let's use the `affix.bs.affix` event, which is an event fired just before the affixing of the element:

```
$(document).ready(function() {  
... // rest of the JavaScript code  
  
$('#profile').on('affix.bs.affix', function() {  
    $(this).width($(this).width() - 1);  
    $('#main').addClass('col-md-offset-3');  
}).on('affix-top.bs.affix', function() {  
    $(this).css('width', '');  
    $('#main').removeClass('col-md-offset-3');  
});  
});
```

Thus, we have played with some tricks in the preceding JavaScript code.

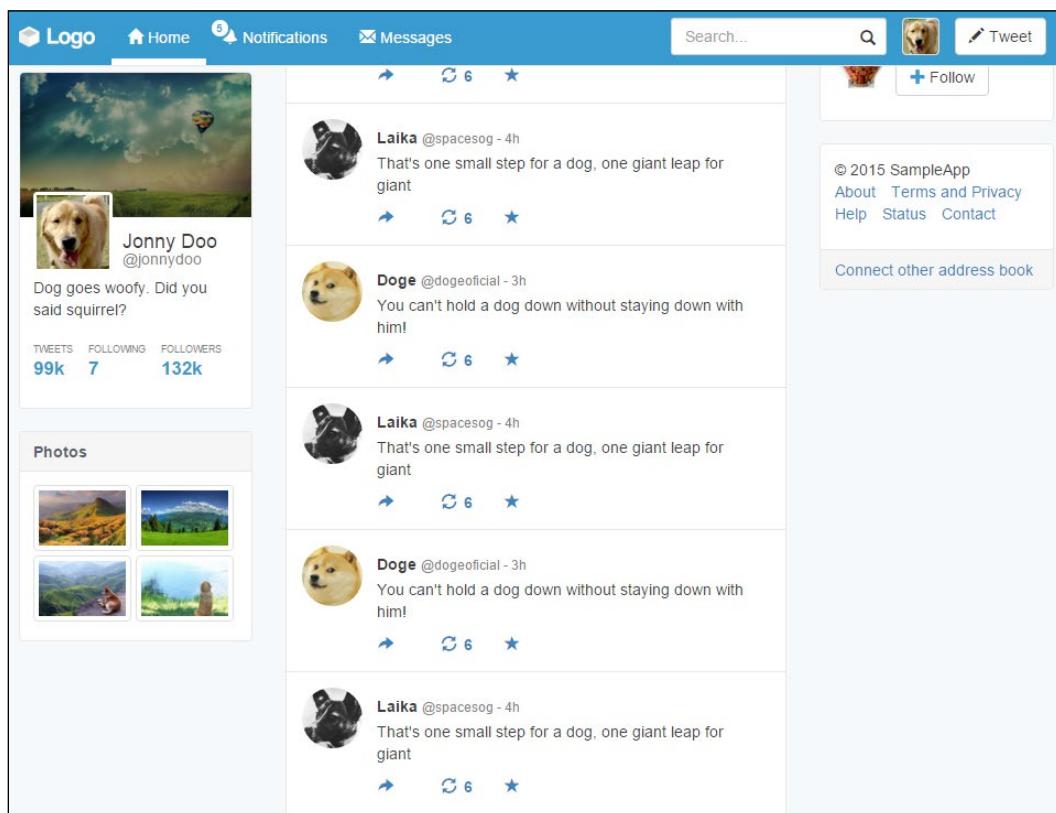
In the first delegated event, `.on('affix.bs.affix', handler)`, when the element switches to `position: fixed`, we keep the width of the left column. It would change the width because the `.col-md-3` class does not have a fixed width; it uses a percentage width.

We also added the offset to the middle column, corresponding to the detached left column, the `.col-md-offset-3` class.

The `affix-top.bs.affix` event does the opposite action, firing when the element returns to the original top position and removing the custom width and the offset class in the middle column.

To remove the fixed width and return to the `.col-md-3` percentage width, just add the `$(this).css('width', '')` line. Also remove the `.col-md-offset-3` class from the `#main` content.

Refresh the web browser, scroll the page, and see the result, exemplified in the next screenshot. Note that the profile is fixed on the left while the rest of the content scrolls with the page:



Finishing the web app

To finish the web application example, we just need to create another modal when we click on the **Messages** link at the navigation bar.

To create it, we will use the same methodology used to create the modal for the **Tweet** button. So, add the data attributes' markups to the **Messages** link in `.nav.navbar-nav`, as follows:

```
<ul class="nav navbar-nav">
  <li class="active">
    <a href="#">
      <span class="glyphicon glyphicon-home" aria-hidden="true"></span>
      Home
    </a>
  </li>
  <li>
    <a href="#">
      <span class="badge">5</span>
      <span class="glyphicon glyphicon-bell" aria-hidden="true"></span>
      Notifications
    </a>
  </li>
  <li>
    <a href="#" role="button" data-toggle="modal" data-target="#messages-modal">
      <span class="glyphicon glyphicon-envelope" aria-hidden="true"></span>
      Messages
    </a>
  </li>
  <li class="visible-xs-inline">
    <a href="#">
      <span class="glyphicon glyphicon-user" aria-hidden="true"></span>
      Profile
    </a>
  </li>
  <li class="visible-xs-inline">
```

```
<a href="#">
    <span class="glyphicon glyphicon-off" aria-hidden="true"></span>
    Logout
</a>
</li>
</ul>
```

The highlighted code says that this link plays the role button, toggling a modal identified by the #messages-modal ID. Create the base of this modal at the end of the HTML code, just after #tweet-modal:

```
<div id="messages-modal" class="modal fade" tabindex="-1"
role="dialog">
    <div class="modal-dialog" role="document">
        <div class="modal-content">
            <div class="modal-header">
                <button type="button" class="close" data-dismiss="modal"
aria-label="Close">
                    <span aria-hidden="true">&times;</span>
                </button>
                <h4 class="modal-title">Dog messages</h4>
                <button type="button" class="btn btn-primary btn-message">New
message</button>
            </div>
            <div class="modal-body">
            </div>
        </div>
    </div>
</div>
```

We made some changes in comparison to #tweet-modal. Firstly, we removed .modal-footer from this modal, since we do not need these options in the modal. Like almost the entire framework, Bootstrap allows us to include or exclude elements as per our wishes.

Secondly, we created a new button, **New message**, in the header, identified by the .btn-message class. To present the button correctly, create the following CSS style:

```
#messages-modal .btn-message {
    position: absolute;
    right: 3em;
    top: 0.75em;
}
```

Now let's create the content inside the modal. We will add a list of messages in the modal. Check out the HTML with the content added:

```
<div class="modal fade" id="messages-modal" tabindex="-1"
role="dialog">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <button type="button" class="close" data-dismiss="modal"
aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
        <h4 class="modal-title">Dog messages</h4>
        <button type="button" class="btn btn-primary btn-message">New
message</button>
      </div>
      <div class="modal-body">
        <ul class="list-unstyled">
          <li>
            <a href="#">
              
              <div class="msg-content">
                <h5>Laika <small>@spacesog</small></h5>
                <p>Hey Jonny, how is down there?</p>
              </div>
            </a>
          </li>
          <li>
            <a href="#">
              
              <div class="msg-content">
                <h5>Doge <small>@dogeoficial </small></h5>
                <p>Wow! How did I turned in to a meme?</p>
              </div>
            </a>
          </li>
          <li>
            <a href="#">
              
              <div class="msg-content">
                <h5>Cat <small>@crazycat</small></h5>
                <p>You will never catch me!</p>
              </div>
            </a>
          </li>
        </ul>
      </div>
    </div>
  </div>
</div>
```

```
        </a>
    </li>
    <li>
        <a href="#">
            
            <div class="msg-content">
                <h5>Laika <small>@spacesog</small></h5>
                <p>I think I saw you in Jupiter! Have you been there
recently?</p>
            </div>
        </a>
    </li>
</ul>
</div>
</div>
</div>
```

To finish our job, we just create some style in the CSS in order to display our list correctly:

```
#messages-modal .modal-body {
    max-height: 32rem;
    overflow: auto;
}

#messages-modal li {
    padding: 0.75rem;
    border-bottom: 0.1rem solid #E6E6E6;
}

#messages-modal li:hover {
    background-color: #E6E6E6;
}

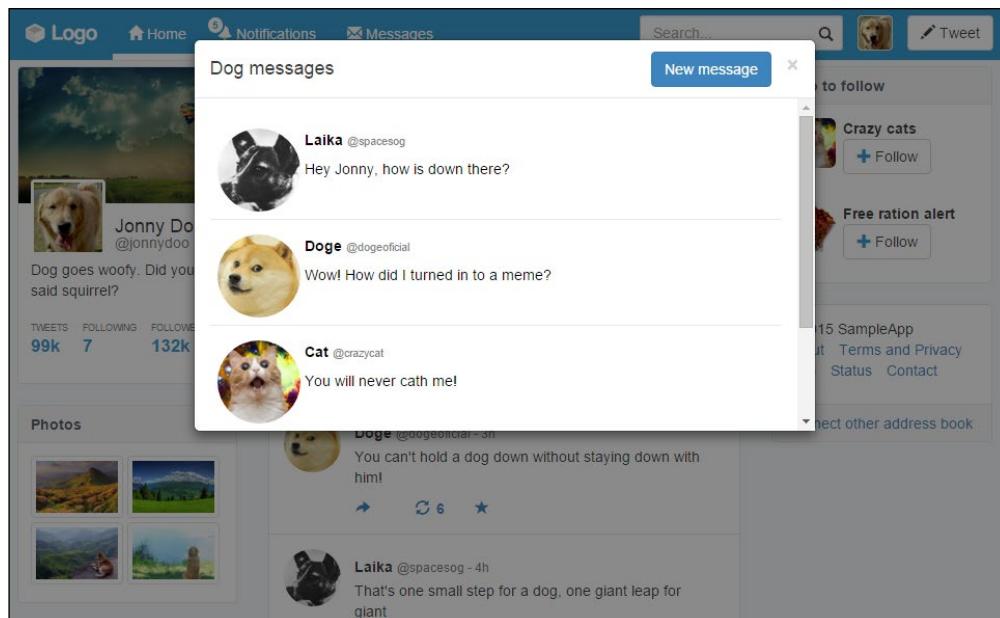
#messages-modal li a:hover {
    text-decoration: none;
}

#messages-modal li img {
    max-width: 15%;
}
```

```
#messages-modal .msg-content {  
    display: inline-block;  
    color: #000;  
}  
  
#messages-modal .msg-content h5 {  
    font-size: 1em;  
    font-weight: bold;  
}
```

In this CSS, we simply set a maximum height for the modal body, while adding a scroll overflow. For the list and the link, we changed the style for hover and adjusted the font weight, size, and color for display.

Refresh the web browser, click on the **Messages** link in the navigation bar and see your nice modal, as follows:



Summary

In this chapter, we finished our web application example. The main objective here was to learn about the Bootstrap plugins that we had not described before.

First, you learned about data attributes and how to use them with Bootstrap. After that, we saw both the possible ways to call plugins: via pure JavaScript or just through data attributes APIs.

We started and finished plugins with modals. Modals are one of the main plugins in Bootstrap because they are very versatile and customizable. Thus, they are fit for multiple contexts where you need some interaction with the user but do not want to move to another page.

In the middle of the chapter, we talked about two plugins that are closely related. They are the tooltip and the popover. Both came from the same initial plugin but with different contexts. Tooltips are used for auxiliary content, and popovers are something midway between a modal and a tooltip, so they can display more content compared to tooltips, but not too much intrusive like modals.

Creating a web application that is Twitter-like is an important kind of knowledge, since this can be replicated to different sources. Web applications have revolutionized the Web in different ways, and Bootstrap has taken the lead by helping us create faster and more beautiful web pages.

In the next chapter, we will step into an even more challenging example—we will build a dashboard web application from scratch! Just like the web application presented in this chapter, web dashboards are very popular across the Internet, and building one will place us at the same stratum as some of the best web developers. Ready for the advanced level?

9

Entering in the Advanced Mode

Alright, there is no more time to spend on testing our skills. Now it's time to truly test ourselves with a big challenge: creating an admin dashboard using Bootstrap. Now Bootstrap can help us, but we must know how to handle the majority of the framework to deliver professional work.

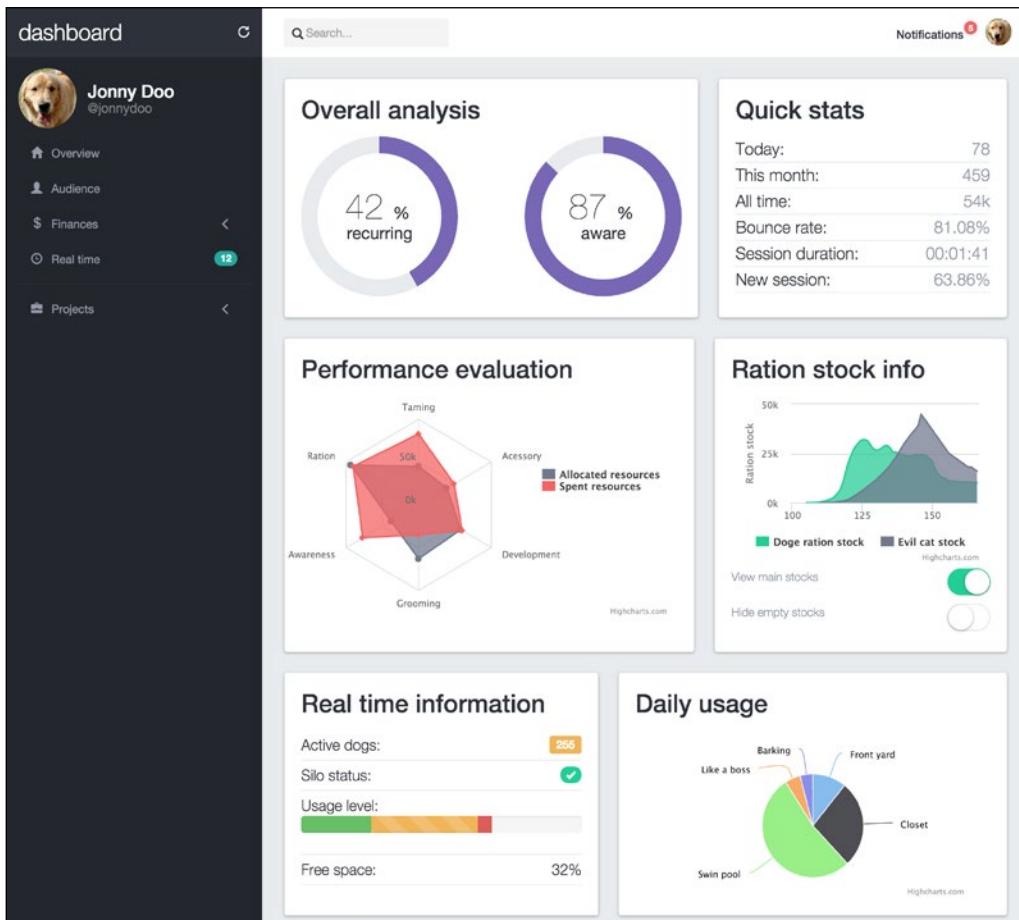
We need a plan to build this dashboard from scratch to its final form. Therefore, we will follow a designer template and recreate it from an image to a web page. Following this concept, you will learn about:

- The fluid container
- The flexbox layout
- Bootstrap custom stacked navigation
- The collapse plugin
- Bootstrap and advanced CSS
- External plugin integration
- Single-page application loading

This is the final example of the book. Let's face it and nail the Bootstrap framework. I know you are thoroughly able to defeat this final boss!

The master plan

As I mentioned, this is a professional job and it deserves a professional treatment. Now we will have a design guideline to follow. Up next is a screenshot of the dashboard that we have to reproduce by code:



As you can see, the dashboard is composed of a navigation header containing some information, a search bar, and notifications. On the left-hand side is a menu with the sections of the web application. In the center is a set of charts about the page status. It looks good in the screenshot and will look even better in the browser!

The page scaffolding consists of the following:

- The Bootstrap navigation component at the header, which is affixed to the top of the page
- A fluid container with two columns
- The left-hand-side column contains the dashboard menu and is affixed
- The right-hand-side column is the main content, which holds a set of cards that display some statistics

First of all, before you create any element, create a new file using the same structure that we pointed out for starting every example in the book (refer to the Bootstrap required tags section in *Chapter 1, Getting Started*, for more information). Create a file named `dashboard.html` and apply the default starter HTML to the file. Now we are ready to go!

The last navigation bar with flexbox

You may be bored of doing navigation bars; however, because of the acquired experience, we will do this one very quickly, taking advantage of the code written in previous examples.

Create a `<nav>` element, and inside it, create a `.container-fluid` and a `.row`:

```
<nav class="navbar navbar-fixed-top">
  <div class="container-fluid">
    <div class="row">
      </div>
    </div>
  </div>
</nav>
```

This `.row` element will have two columns, just as we mentioned that will be done for the main container. On the first one, let's create the dashboard title and a refresh button, as follows:

```
<nav class="navbar navbar-fixed-top">
  <div class="container-fluid">
    <div class="row">
      <div class="col-sm-3 top-left-menu">
        <div class="navbar-header">
          <a class="navbar-brand" href="webapp.html">
            <h1>dashboard</h1>
          </a>
        </div>
      </div>
    </div>
  </div>
</nav>
```

```
</div>
<a href="#" data-toggle="tooltip" data-placement="bottom"
data-delay="500" title="Refresh data" class="header-refresh pull-
right">
    <span class="glyphicon glyphicon-repeat" aria-
hidden="true"></span>
</a>
</div>
</div>
</div>
</nav>
```

Note that for the refresh button, we have used `.glyphicon` and added a tooltip. Do not forget to activate the tooltip in the `main.js` file that you have loaded:

```
$(document).ready(function() {
    $('[data-toggle="tooltip"]').tooltip();
});
```

In the tooltip, we added a delay to it show up with the `data-delay="500"` attribute. We mentioned this as an option for tooltip, but haven't made use of it so far. This will just delay the appearance of the tooltip for 500 milliseconds, while hovering the refresh link.

Inside `.nav-header`, add `.navbar-toggle`, which will be displayed for small screens and collapse the menu:

```
<nav class="navbar navbar-fixed-top">
    <div class="container-fluid">
        <div class="row">
            <div class="col-sm-3 top-left-menu">
                <div class="navbar-header">
                    <a class="navbar-brand" href="webapp.html">
                        <h1>dashboard</h1>
                    </a>
                    <button type="button" class="navbar-toggle collapsed"
data-toggle="collapse" data-target="#nav-menu" aria-expanded="false">
                        <span class="sr-only">Toggle navigation</span>
                        <span class="icon-bar"></span>
                        <span class="icon-bar"></span>
                        <span class="icon-bar"></span>
                    </button>
                </div>
                <a href="#" data-toggle="tooltip" data-placement="bottom"
data-delay="500" title="Refresh data" class="header-refresh pull-
right">
```

```
<span class="glyphicon glyphicon-repeat" aria-hidden="true"></span>
</a>
</div>
</div>
</div>
</nav>
```

So far, we have no secrets. We have just replicated components that we used before. Following our pipeline, we should create some CSS rules to style our page, although first let's create some common CSS style. At the beginning of `base.css`, which is loaded in our HTML, we add the style:

```
.transition,
.transition:hover,
.transition:focus {
    -webkit-transition: all 150ms ease-in-out;
    -moz-transition: all 150ms ease-in-out;
    -ms-transition: all 150ms ease-in-out;
    -o-transition: all 150ms ease-in-out;
    transition: all 150ms ease-in-out;
}

html, body {
    position: relative;
    height: 100%;
    background-color: #e5e9ec;
}
```

First, we created a common `.transition` class to be used in multiples cases (we will use it in the chapter). Transitions were introduced in CSS3 and they allow us to create transition effects. In this case, it's an effect of `ease-in-out` for any element that has this class.

Also, for `html` and `body`, we changed the background and set the position and height to fill the entire screen.

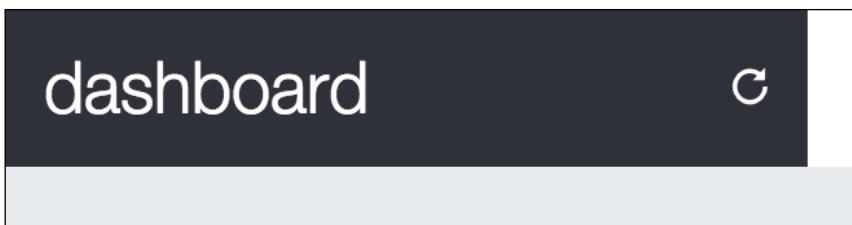
Next, we must add the CSS for the navigation header:

```
nav.navbar-fixed-top {
    background-color: #FFF;
    border: none;
}
```

```
nav .top-left-menu {  
    background-color: #252830;  
    display: -webkit-flex;  
    display: flex;  
    align-items: center;  
}  
  
.navbar-brand {  
    height: auto;  
}  
  
.navbar-brand h1 {  
    margin: 0;  
    font-size: 1.5em;  
    font-weight: 300;  
    color: #FFF;  
}  
  
nav .header-refresh {  
    margin-left: auto;  
    color: #FFF;  
}
```

Here, we changed the color of the elements. But the most important thing here is the usage of the flexbox rules (do you remember flexbox, which we discussed in *Chapter 5, Making It Fancy*, in the *Understanding flexbox* section?). Remember that Bootstrap 4 will support flex display, so it is nice to keep using it, since it should be the standard in the near future for every browser.

The result of this part must look like what is shown in the following screenshot:



The navigation search

Following our design, we have to create a search form. So, just after the closure of `.top-left-menu`, add the form code, such as the portion in bold:

```
<nav class="navbar navbar-fixed-top">
  <div class="container-fluid">
    <div class="row">
      <div class="col-sm-3 top-left-menu">
        ...
      </div>

      <form id="search" role="search" class="hidden-xs col-sm-3">
        <div class="input-group">
          <span class="glyphicon glyphicon-search" aria-
hidden="true"></span>
          <input type="text" class="form-control transition"
placeholder="Search..."/>
        </div>
      </form>

    </div>
  </div>
</nav>
```

As usual, it's CSS time:

```
nav form#search {
  padding: 0.9em;
}

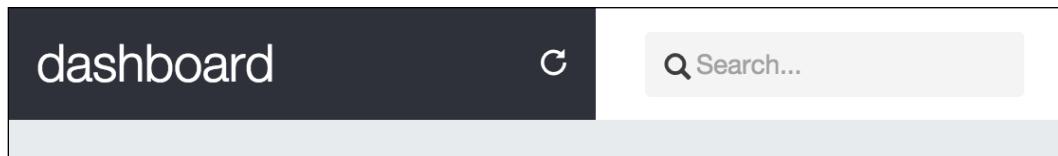
nav form#search .input-group {
  display: -webkit-flex;
  display: flex;
  align-items: center;
}

nav form#search .input-group .form-control {
  border-radius: 0.25em;
  border: none;
  width: 70%;
```

```
padding-left: 1.9em;  
background-color: #F3F3F3;  
box-shadow: none;  
}  
  
nav form#search .input-group .form-control:focus {  
width: 100%;  
box-shadow: none;  
}  
  
nav form#search .glyphicon-search {  
z-index: 99;  
left: 1.7em;  
}
```

In this CSS, we have again used the `display: flex` property. In addition to this, we created a pseudo-class rule for `.form-control`. The `:focus`, which is activated whenever the input has focus, in other words, is receiving some text. This `:focus` rule will change the width of the input when you focus the input, which happens when you click on it.

Refresh the web page and click on the input on the search form. Note that we applied the `.transition` class in this element, so when we focus it, the change of width is smoothed in a transition. The result should look like this:



The menu needs navigation

To finish the navigation bar, we have to create the right-hand-side content of the navigation bar, which we call `#nav-menu`. This menu will hold the notification list, placed as a button dropdown.

After `<form>`, place the presented HTML:

```
<div id="nav-menu" class="collapse navbar-collapse pull-right">  
  <ul class="nav navbar-nav">  
  </ul>  
</div>
```

Inside this `` tag, we will place the notifications. Right now, we just have this option, but with this list, we can add multiple items in the navigation bar. So, add the following code for the item:

```
<div id="nav-menu" class="collapse navbar-collapse pull-right">
  <ul class="nav navbar-nav">
    <li>
      <div id="btn-notifications" class="btn-group">
        <span class="badge">3</span>
        <button type="button" class="btn btn-link dropdown-toggle"
data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
          Notifications
        </button>
      </div>
    </li>
  </ul>
</div>
```

Explaining this item, we can say that it is a button for the notification. There is a wrapper element named `#btn-notifications`. Inside it is a `.badge` to verbalize the number of new notifications, and a `button.btn` that must seem like a link, so we applied the `.btn-link` class to it. The button also contains the tags needed for a Bootstrap drop-down button, such as the `.dropdown-toggle` class and the `data-toggle="dropdown"` data property.

Therefore, every `button.dropdown-toggle` button needs a `ul.dropdown-menu`. Just after `<button>`, create the list:

```
<div id="nav-menu" class="collapse navbar-collapse pull-right">
  <ul class="nav navbar-nav">
    <li>
      <div id="btn-notifications" class="btn-group">
        <span class="badge">3</span>
        <button type="button" class="btn btn-link dropdown-toggle"
data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
          Notifications
        </button>
        <ul id="notification-list" class="dropdown-menu pull-right">
          <li>
            <a href="#">
              <span class="badge"></span>
              
              <div class="notification-message">
                <strong>Laika</strong>
                <p>Hey! How are you?</p>

```

```
        <em class="since">2h ago</em>
      </div>
    </a>
  </li>
  <li>
    <a href="#">
      <span class="badge"></span>
      
      <div class="notification-message">
        <strong>Devil cat</strong>
        <p>I will never forgive you...</p>
        <em class="since">6h ago</em>
      </div>
    </a>
  </li>
  <li>
    <a href="#">
      <span class="badge"></span>
      
      <div class="notification-message">
        <strong>Doge</strong>
        <p>What are you doing? So scare. It's alright
now.</p>
        <em class="since">yesterday</em>
      </div>
    </a>
  </li>
</ul>
</div>
</li>
</ul>
</div>
```

The new list element is pointed out in bold. Even though the content seems long, it is just a repetition of three items with different contents inside our notification list.

Refresh the page, open the dropdown, and you will feel an uncontrollable desire to add some CSS and stop the dropdown from being ugly anymore:

```
/*nav menu*/
nav #nav-menu {
  padding: 0.4em;
  padding-right: 1em;
}
```

```
/*nav menu and notifications*/
#nav-menu #btn-notifications > .badge {
  color: #FFF;
  background-color: #f35958;
  font-size: 0.7em;
  padding: 0.3rem 0.55rem 0.3rem 0.5rem;
  position: absolute;
  right: -0.4rem;
  top: 1rem;
  z-index: 99;
}

#btn-notifications .btn-link {
  padding-top: 1.5rem;
  color: #252830;
  font-weight: 500;
}

#btn-notifications .btn-link:hover {
  text-decoration: none;
}
```

Great! This will make the button and notification badge appear more beautiful. Then it's time for `#notification-list`:

```
#notification-list {
  max-height: 20em;
  overflow: auto;
}

#notification-list a {
  display: -webkit-flex;
  display: flex;
  opacity: 0.7;
  margin: 1.5rem;
  border-radius: 0.5rem;
  padding: 0.5rem 1.3rem;
  background-color: #EFEFEE;
  position: relative;
}

#notification-list a:hover {
  color: #262626;
  text-decoration: none;
  opacity: 1;
```

```
}

/notification-list img {
  display: inline-block;
  height: 35px;
  width: 35px;
  margin-right: 1em;
  margin-top: 1em;
}

/notification-list .notification-message {
  display: inline-block;
  white-space: normal;
  min-width: 25rem;
}

/notification-list .badge:empty {
  display: inline-block;
  position: absolute;
  right: 0.5rem;
  top: 0.5rem;
  background-color: #f35958;
  height: 1.4rem;
}

/notification-list em.since {
  font-size: 0.7em;
  color: #646C82;
}
```

For the notification, we did just some common rules, such as spacing, color, and so on. The only different thing is, again, the use of flexbox to align the content. See this screenshot for the final result of the navigation bar:



 Did you notice that the images appear rounded? Do you know why? This is because of the `.img-circle` Bootstrap helper class; it is present in every `` element.

Checking the profile

In the navigation bar, the last present component is a picture that, when you click on it, opens a user menu, just like what we did in the example of the web application. With no further delay, place the next HTML just after the `` of `#nav-menu`:

```
<div id="nav-menu" class="collapse navbar-collapse pull-right">
  <ul class="nav navbar-nav">
    ...
  </ul>

  <div id="nav-profile" class="btn-group pull-right">
    <button type="button" class="btn btn-link dropdown-toggle thumbnail" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
      
    </button>
    <ul class="dropdown-menu">
      <li><a href="#">Profile</a></li>
      <li><a href="settings.html">Setting</a></li>
      <li role="separator" class="divider"></li>
      <li><a href="#">Logout</a></li>
    </ul>
  </div>
</div>
```

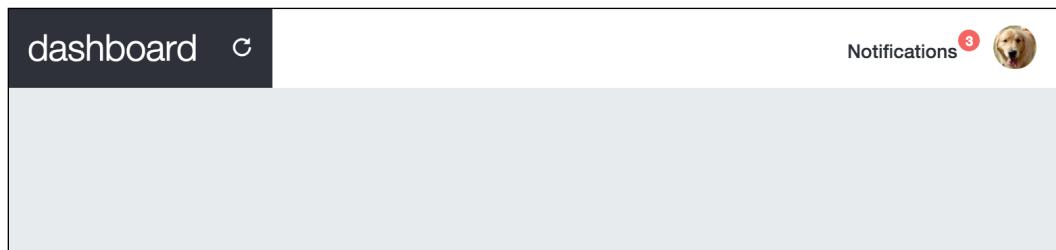
So, it is another button dropdown. The CSS for this HTML is as follows:

```
#nav-profile {
  margin: 0.5em;
  margin-left: 1em;
}

#nav-profile button.thumbnail {
  margin: 0;
  padding: 0;
  border: 0;
}

#nav-profile img {
  max-height: 2.3em;
}
```

We are done! Refresh the web browser and see the result, which should be like what is shown in this screenshot:



Filling the main fluid content

After the navigation bar, we must fill the main content using a fluid layout. For that, we create a `.container-fluid`, just as we did in the `<nav>`. Inside the container, we create a single `.row` and two columns with spacing three and nine, respectively:

```
<div class="container-fluid">
  <div class="row">
    <div id="side-menu" class="col-md-3 hidden-xs">
    </div>

    <div id="main" class="col-md-9">
    </div>
  </div>
</div>
```

It is a common grid, containing one row. In the row, the first column, `#side-menu`, is shown from small viewports up to larger ones, while the `#main` column fills 9 out of 12 grids for medium resolutions.

However, we must not forget that `#side-menu` is actually an affix component. So, let's add the data properties to make it stitch to the top of the page, as we did in the web application example when you were learning this plugin:

```
<div class="container-fluid">
  <div class="row">
    <div id="side-menu" class="col-md-3 hidden-xs" data-spy="affix"
data-offset-top="0">
    </div>

    <div id="main" class="col-sm-offset-3 col-md-9">
    </div>
```

```
</div>
</div>
```

Note that because of the addition of the affix, we must set an offset in the `#main` div with the `.col-sm-offset-3` class.

From the side stacked menu

Let's fill `#side-menu` with content. At first, we have to create the profile block, which contains the user data. Place the following HTML inside the referred element:

```
<div id="side-menu" class="col-md-3 hidden-xs" data-spy="affix"
data-offset-top="0">
  <div class="profile-block">
    
    <h4 class="profile-title">Jonny Doo
    <small>@jonnydoo</small></h4>
  </div>
</div>
```

Check out the page in the browser, and you will see that it is not displaying nicely. For the CSS, we must follow this style:

```
#side-menu {
  background-color: #1b1e24;
  padding-top: 7.2rem;
  height: 100%;
  position: fixed;
}

#side-menu .profile-block > * {
  display: inline-block;
}

#side-menu .profile-block img {
  width: 70px;
}

#side-menu .profile-title {
  color: #FFF;
  margin-left: 1rem;
  font-size: 1.5em;
  vertical-align: middle;
}

#side-menu .profile-title small {
  display: block;
}
```

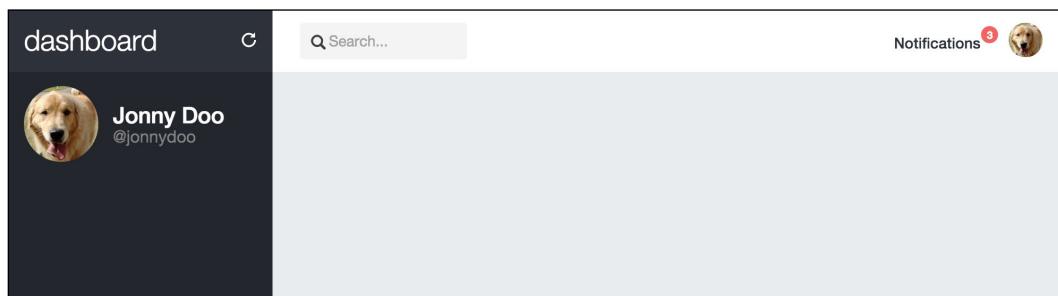
With that, the `#side-menu` should fill the entire left height, but if you resize the browser to a smaller resolution, you will see that `#nav-header` does not resize together with the main content. This is a small challenge. Do you know why it is happening?

That was a little prank! Did I get you? In `#side-menu`, we applied only the class for medium viewports, that is, the `.col-md-3` class. What we should have done was apply the class for small devices as well to make it responsive to small viewports and resize like all the other elements, which needs the `.col-sm-*` class. In this case, just change the class of `#side-menu` and in the `#main` element as well:

```
<div class="container-fluid">
  <div class="row">
    <div id="side-menu" class="col-sm-3 hidden-xs" data-spy="affix"
data-offset-top="0">
      </div>

      <div id="main" class="col-sm-offset-3 col-sm-9">
        </div>
      </div>
    </div>
  </div>
```

Here is a screenshot that shows the result of the side menu for the moment:



I heard that the left menu is great!

A web application is never a web application if it does not have a menu. After the profile info in `#side-menu`, we will add a stacked menu.

Hearing the word "stacked" for a menu, what you remember? Of course, the `.nav-stacked` menu from Bootstrap! Let's create a `.nav-stacked` component in this menu. Therefore, after `#profile-block`, append the following HTML:

```
<ul class="nav nav-pills nav-stacked">
  <li>
```

```
<a href="#" class="transition">
    <span class="glyphicon glyphicon-home" aria-hidden="true"></span>
        Overview
    </a>
</li>
<li>
    <a href="#" class="transition">
        <span class="glyphicon glyphicon-user" aria-hidden="true"></span>
            Audience
    </a>
</li>
<li>
    <a href="#" class="transition">
        <span class="glyphicon glyphicon-usd" aria-hidden="true"></span>
            Finances
        <span class="glyphicon glyphicon-menu-left pull-right
transition" aria-hidden="true"></span>
    </a>
</li>
<li>
    <a href="#" class="transition">
        <span class="glyphicon glyphicon-time" aria-hidden="true"></span>
            Real time
        <span class="badge pull-right">12</span>
    </a>
</li>
<li class="nav-divider"></li>
<li>
    <a href="#" class="transition">
        <span class="glyphicon glyphicon-briefcase" aria-
hidden="true"></span>
            Projects
        <span class="glyphicon glyphicon-menu-left pull-right
transition" aria-hidden="true"></span>
    </a>
</li>
</ul>
```

No secrets here! Just create a simple stacked list using the `.nav`, `.nav-pills`, and `.nav-stacked` classes. Bootstrap will do the magic for you. You will learn a little trick now – the collapse Bootstrap plugin.

Learning the collapse plugin

Collapse is another plugin from Bootstrap that toggles the visualization behavior of an element. It will show or collapse an item regarding the trigger of an action.



The collapse plugin requires the transition plugin present in the Bootstrap framework.

To add the collapse event to an element, you should add a data attribute called `data-toggle="collapse"`. If the element is a link `<a>`, point the anchor to the identifier of the element, like this: `href="#my-collapsed-element"`. If it is a `<button>`, add the data attribute pointing the identifier, like this for instance: `data-target="#my-collapsed-element"`. The difference between using `href` for a link and `data-target` for a button is because of the semantics of the element. Naturally, every link is expected to have a reference in the `href`, although we do not have this requirement in buttons. So, Bootstrap binds the element through a `data-target` data attribute.

We will create a sublist in our menu using the collapse plugin for the **Finances** and **Projects** entries. After the link of each one of these items, create a secondary list, as is pointed in the following highlighted HTML. Also, since we are using `<a>` tags, we add to the `href` the identifier of the element that will be collapsed and the `data-toggle` corresponding to collapse:

```
<ul class="nav nav-pills nav-stacked">
  <li>
    <a href="#" class="transition">
      <span class="glyphicon glyphicon-home" aria-hidden="true"></span>
      Overview
    </a>
  </li>
  <li>
    <a href="#" class="transition">
      <span class="glyphicon glyphicon-user" aria-hidden="true"></span>
      Audience
    </a>
  </li>
  <li>
    <a href="#finances-opt" class="transition" role="button"
      data-toggle="collapse" aria-expanded="false" aria-controls="finances-
      opts">
```

```
<span class="glyphicon glyphicon-usd" aria-hidden="true"></span>
Finances
<span class="glyphicon glyphicon-menu-left pull-right
transition" aria-hidden="true"></span>
</a>
<ul class="collapse list-unstyled" id="finances-opts">
<li>
    <a href="#" class="transition">
        Incomes
    </a>
</li>
<li>
    <a href="#" class="transition">
        Outcomes
    </a>
</li>
</ul>
</li>
<li>
    <a href="#" class="transition">
        <span class="glyphicon glyphicon-time" aria-hidden="true"></span>
        Real time
        <span class="badge pull-right">12</span>
    </a>
</li>
<li class="nav-divider"></li>
<li>
    <a href="#projects-opts" class="transition" role="button"
data-toggle="collapse" aria-expanded="false" aria-controls="projects-
opts">
        <span class="glyphicon glyphicon-briefcase" aria-
hidden="true"></span>
        Projects
        <span class="glyphicon glyphicon-menu-left pull-right
transition" aria-hidden="true"></span>
    </a>
    <ul class="collapse list-unstyled" id="projects-opts">
        <li>
            <a href="#" class="transition">
                Free ration nation
            </a>
        </li>
        <li>
```

```
<a href="#" class="transition">
    Cats going crazy
</a>
</li>
</ul>
</li>
</ul>
```

To make it clear, take as an example the first collapsed menu from **Finances**. Below the **Finances** link, we created the list to be collapsed, identified by #finances-opt. We also added the .collapse class, which is a Bootstrap class used to collapse elements.

Going back to the **Finances** link, add to the href the ID of the collapsed list, #finances-opt. Also, we added the data-toggle="collapse" required to Bootstrap Collapse work. Finally, we added the aria attributes, aria-controls and aria-controls, for semantic notation.

Refresh the browser and you will see how Bootstrap does almost the entire job for us. In the CSS, we need to add some simple styles for color and spacing:

```
#side-menu ul.nav {
    margin-top: 1rem;
}

#side-menu ul.nav a {
    color: #8b91a0;
}

#side-menu ul.nav a:hover,
#side-menu ul.nav a:focus {
    color: #FFF;
    background-color: inherit;
}

#side-menu ul.nav a .glyphicon {
    margin-right: 0.7rem;
}

#side-menu ul.nav a .glyphicon.pull-right {
    margin-top: 0.2rem;
}

#side-menu ul.nav a .badge {
    background-color: #1ca095;
```

```
}
```

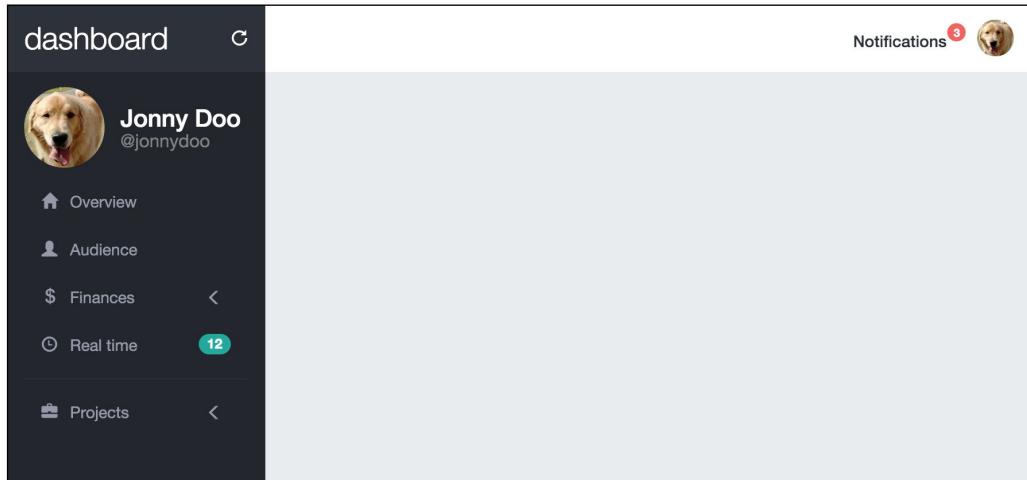
```
#side-menu ul.nav .nav-divider {
  background-color: #252830;
}

#side-menu ul.nav ul {
  margin-left: 10%;
}

#side-menu ul.nav ul a {
  display: block;
  background-color: #2b303b;
  padding: 1rem;
  margin-bottom: 0.3rem;
  border-radius: 0.25em;
}

#side-menu ul.nav ul a:hover {
  text-decoration: none;
  background-color: #434857;
}
```

Go to the browser and refresh it to see the result. It should look like what is shown in this screenshot:



Using some advanced CSS

Let's add a cherry to this pie while learning other CSS properties. What do you think if we could rotate the arrow of the items that have collapsed menus by 90 degrees anticlockwise to create an opening effect? It would be awesome—even more if we did that with only CSS.

Add the next CSS rule for the effect using the `transform` property:

```
#side-menu ul.nav a:focus .glyphicon.pull-right {  
    -moz-transform: rotate(-90deg);  
    -webkit-transform: rotate(-90deg);  
    -o-transform: rotate(-90deg);  
    -ms-transform: rotate(-90deg);  
    transform: rotate(-90deg);  
}
```

This `transform` property will do exactly what we want; when the link is in focus (which means it is clicked on), the icon from the arrow will rotate 90 degrees anticlockwise, because of the minus signal.

To be more pro, let's use a supernew property called `will-change`. Add the style to the following selector:

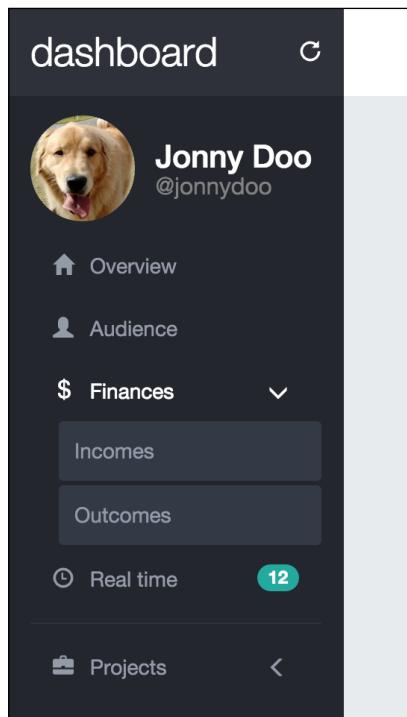
```
#side-menu ul.nav a .glyphicon.pull-right {  
    margin-top: 0.2rem;  
    will-change: transform;  
}
```



The `will-change` property

The `will-change` property optimizes animations by letting the browser know which elements will change and need careful treatment. Currently, this property is not supported by all browsers, but soon it will be. Check out its availability at <http://caniuse.com/#feat=will-change>.

Click to open a submenu and see the opening menu animation with the arrow rotation. The next screenshot presents an open menu:



Filling the main content

To finish the content of the first step of our dashboard, we will move on to the main content, referred to by the column identified by `#main`. In this section, we will create a set of cards almost similar to the cards made in the web application demo, along with the use of some external plugins for chart generation.

However, before everything else, we need to create some common CSS in our main content. Add the following style to the `base.css` file:

```
#main {  
    padding-top: 7.2rem;  
    display: -webkit-flex;  
    display: flex;  
    align-items: stretch;  
    flex-flow: row wrap;  
}  
  
.card {  
    position: relative;  
    border-radius: 0.25em;  
    box-shadow: 0 1px 4px 0 rgba(0, 0, 0, 0.37);  
    background-color: #FFF;  
    margin: 1.25rem;  
    flex-grow: 5;  
}  
  
.card * {  
    color: #252830;  
}  
  
.card-block {  
    padding: 2rem;  
}  
  
.card-block h2 {  
    margin: 0;  
    margin-bottom: 1.5rem;  
    color: #252830;  
}
```

As I said, we will play with cards inside this element, so let's create the classes that relate to it almost similarly to what we did in the cards for the web application example. In this case, even though you are using Bootstrap 4, you must add to those classes to correctly style the cards component.

Our first card will be placed inside the `#main` element. So, create the following HTML. The first card will be an **Overall analysis** card:

```
<div id="main" class="col-sm-offset-3 col-sm-9">
  <div class="card" id="pie-charts">
    <div class="card-block">
      <h2>Overall analysis</h2>

      </div>
    </div>
  </div>
```

Rounding the charts

The first plugin that we will use is called *Easy Pie Chart* (<https://rendro.github.io/easy-pie-chart/>). This plugin generates only rounded pie charts. It is a lightweight, single-purpose plugin.

In order to use this plugin, you can get it through bower, through npm, or by simply downloading the ZIP file from the repository. In any case, what you will need to do at the end is load the plugin in the HTML file.

We will use the jQuery version of the plugin, so we place the JavaScript file in our `js` folder and load the plugin at the end of our file:

```
<script src="js/jquery-1.11.3.js"></script>
<script src="js/bootstrap.js"></script>
<script src="js/jquery.easypiechart.min.js"></script>
<script src="js/main.js"></script>
```

Inside our `#pie-charts` card that we just created, let's add some HTML that is needed for the corresponding plugin:

```
<div class="card" id="pie-charts">
  <div class="card-block">
    <h2>Overall analysis</h2>
    <div class="round-chart" data-percent="42">
      <span>
        42
        <small>
          % <br>
          recurring
        </small>
      </span>
    </div>
  </div>
```

```
<div class="round-chart" data-percent="87">
  <span>
    87
    <small>
      % <br>
      aware
    </small>
  </span>
</div>
</div>
</div>
```

To make the *Easy Pie Chart* plugin work, you must apply it to an element, and you can pass arguments by data attributes. For example, in this case, we have `data-percent`, which will say the fill of the chart.

Go to your JavaScript file (the `main.js` file), and inside the `ready` function (just as we did in *Chapter 8, Working with JavaScript, Creating our custom modal*), add the following code to initialize the plugin:

```
$(document).ready(function() {
  $('.round-chart').easyPieChart({
    'scaleColor': false,
    'lineWidth': 20,
    'lineCap': 'butt',
    'barColor': '#6d5cae',
    'trackColor': '#e5e9ec',
    'size': 190
  });
});
```

What we are telling here is the style of the chart. But we need more style! We append the following CSS to our `base.css`:

```
.round-chart {
  display: inline-block;
  position: relative;
}

.round-chart + .round-chart {
  float: right;
}

.round-chart span {
  font-size: 3em;
```

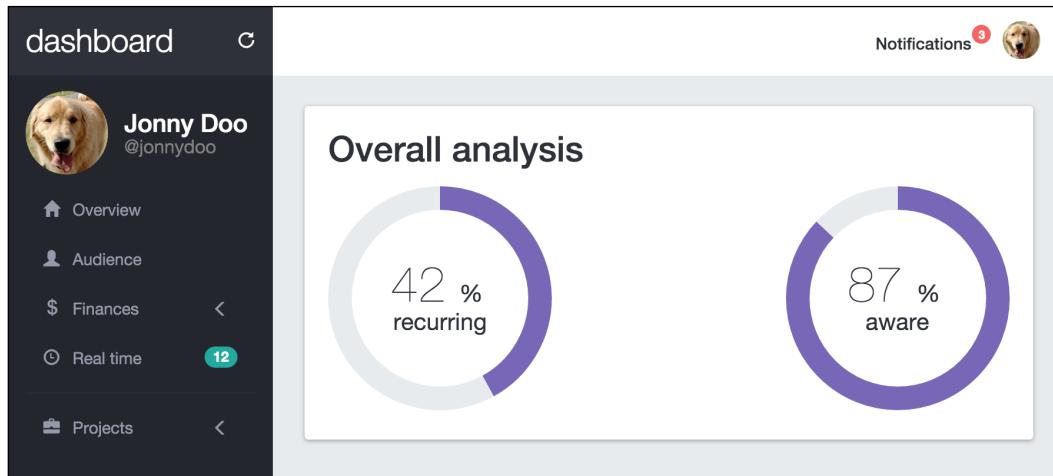
```

font-weight: 100;
line-height: 1.7rem;
width: 12rem;
height: 4.4rem;
text-align: center;
position: absolute;
margin: auto;
top: 0;
bottom: 0;
left: 0;
right: 0;
}

.round-chart span > small {
  font-size: 2rem;
  font-weight: 400;
}

```

What we are doing here, besides changing some spacing, is the centralization of the percentage text that we have added. Refresh the page and you should see something like this:



As you can see, the card has filled the entire line. This is because of the flexbox layout that we are using in the `#main` element. Check out the CSS that we used for this element:

```
#main {
  padding-top: 7.2rem;
```

```
display: -webkit-flex;
display: flex;
align-items: stretch;
flex-flow: row wrap;
}
```

With the `flex` display, if we use `align-items: stretch`, the layout will stretch to fill the content in the cross axis.

The `flex-flow` style is a shorthand for `flex-direction` and `flex-wrap`. By using this property, we can apply both options to specify the direction of the items, in this case as a `row`, and set the row to wrap to the next lines.

Also, for each card, we have created the `flex-grow: 5` property, which says to the element that it can assume five different sizes inside the `#main` container.

Creating a quick statistical card

The next card contains statistical information and we will create it just by using Bootstrap components. So, after the `#pie-charts` card, create another one in HTML:

```
<div class="card" id="quick-info">
  <div class="card-block">
    <h2>Quick stats</h2>
    <div class="quick-stats">
      <strong>Today:</strong>
      <span>78</span>
    </div>
    <div class="quick-stats">
      <strong>This month:</strong>
      <span>459</span>
    </div>
    <div class="quick-stats">
      <strong>All time:</strong>
      <span>54k</span>
    </div>
    <div class="quick-stats">
      <strong>Bounce rate:</strong>
      <span>81.08%</span>
    </div>
```

```
<div class="quick-stats">
  <strong>Session duration:</strong>
  <span>00:01:41</span>
</div>
<div class="quick-stats">
  <strong>New session:</strong>
  <span>63.86%</span>
</div>
</div>
</div>
```

The `#quick-info` card contains only the common elements that will be displayed, each one in a line inside `.card`. Add the next CSS style to correctly display this card:

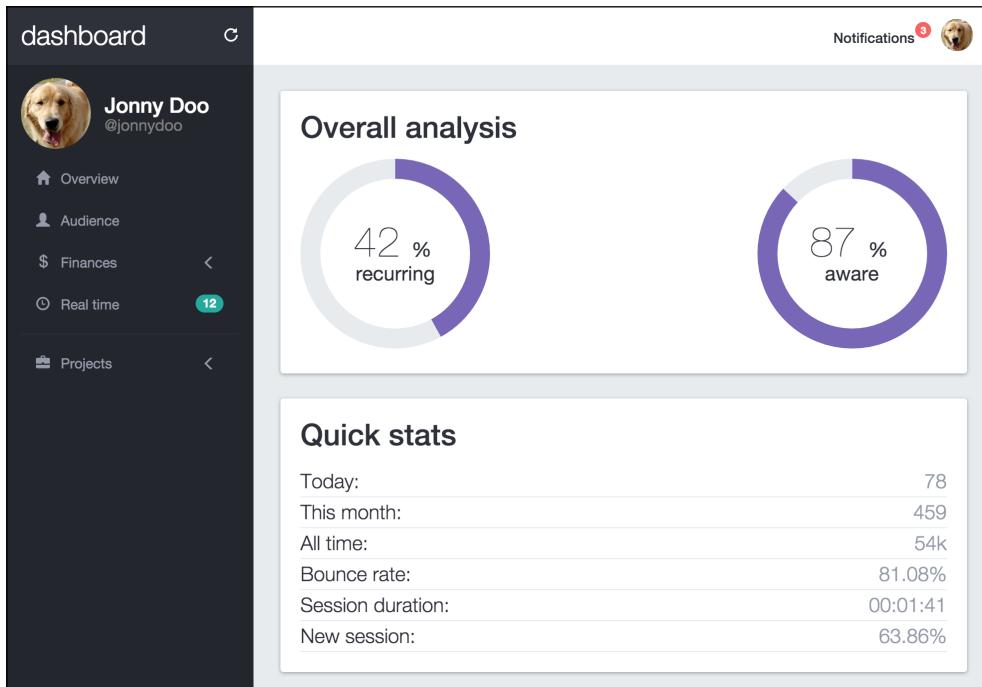
```
#quick-info .card-block {
  display: flex;
  flex-direction: column;
}

#quick-info .quick-stats {
  font-size: 2rem;
  line-height: 3rem;
  border-bottom: 0.1rem solid #e5e9ec;
}

#quick-info .quick-stats strong {
  font-weight: 300;
}

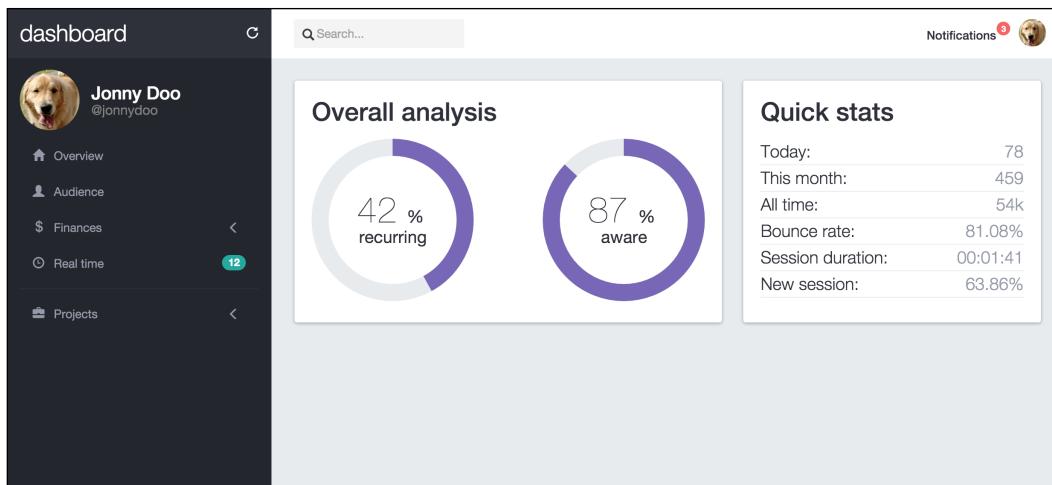
#quick-info .quick-stats span {
  font-weight: 300;
  float: right;
  color: #8b91a0;
}
```

In the web browser, you should see the following result:



But wait! If you look at the initial layout, you will realize that those two cards should be displayed side by side! What happened here?

This is another advantage of using the flex display! With a flex display, each item inside the container will adapt for the display. The previous screenshot was taken from a medium viewport. If you take it from a large-resolution screen, you will see how the elements appear side by side, like this:



Getting a spider chart

The next chart is called a spider chart. The Highcharts plugin (<http://www.highcharts.com/>) is one of the most definitive plugins used to create charts. It provides a wide variety of charts divided over different modules, so you can load only the plugins that you will use.

Just like Easy Pie Chart, you can get Highcharts from different sources. After getting it, let's load the first Highcharts module that we need. Load them along with the other JavaScript files (the plugin is loaded using CDN in this case):

```
<script src="https://code.highcharts.com/highcharts.js"></script>
<script src="https://code.highcharts.com/highcharts-more.js"></script>
```

Create another .card in the HTML after #quick-info, this new one being identified by #performance-eval:

```
<div class="card" id="performance-eval">
  <div class="card-block">
    <h2>Performance evaluation</h2>
    <div class="spider-chart"></div>
  </div>
</div>
```

Highcharts does not require too much HTML, but you need to customize it all over JavaScript. To initialize and customize the plugin, add the following code inside the ready function of the JavaScript:

```
$( '#performance-eval .spider-chart' ).highcharts({  
  
    chart: {  
        polar: true,  
        type: 'area'  
    },  
  
    title: {  
        text: ''  
    },  
  
    xAxis: {  
        categories: ['Taming', 'Accessory', 'Development',  
        'Grooming', 'Awareness', 'Ration'],  
        tickmarkPlacement: 'on',  
        lineWidth: 0  
    },  
  
    yAxis: {  
        gridLineInterpolation: 'polygon',  
        lineWidth: 0,  
        min: 0  
    },  
  
    tooltip: {  
        shared: true,  
        pointFormat: '<span style="color:{series.color}">{series.  
name}<br>  
<b>${point.y:.0f}</b><br/>'  
    },  
  
    legend: {  
        align: 'right',  
        verticalAlign: 'top',  
        y: 70,  
        layout: 'vertical'  
    },  
  
    series: [{  
        name: 'Allocated resources',  
        data: [100, 100, 100, 100, 100, 100]  
    }]  
});
```

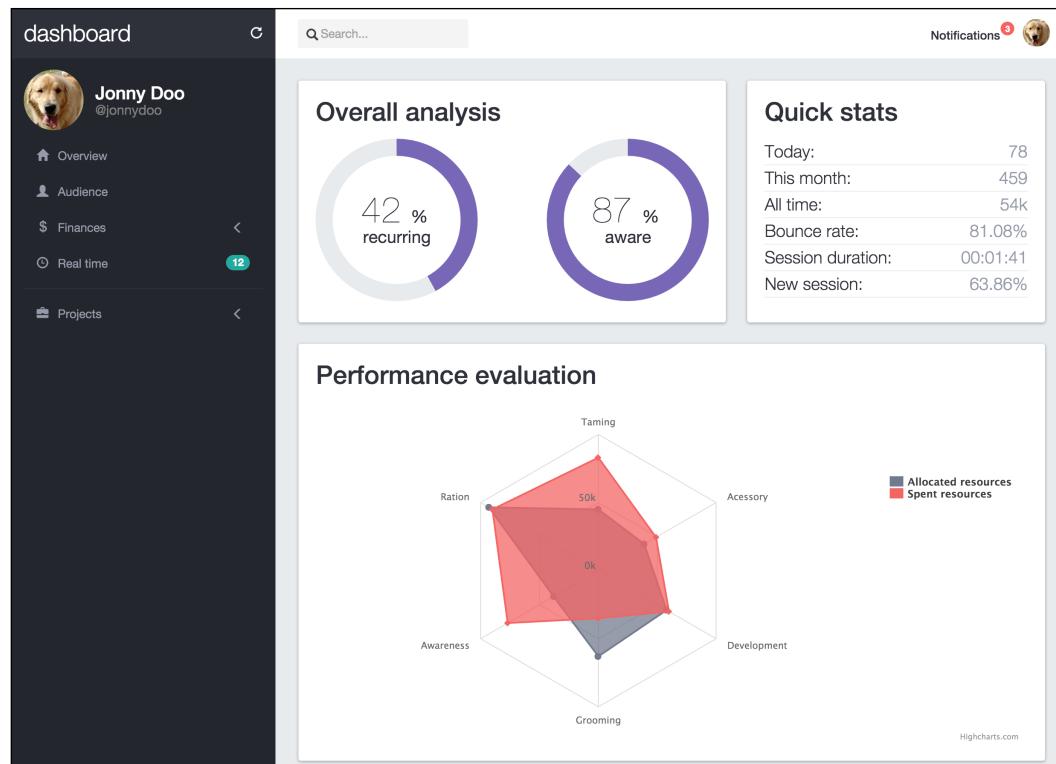
```

        data: [45000, 39000, 58000, 63000, 38000, 93000],
        pointPlacement: 'on',
        color: '#676F84'
    },
    {
        name: 'Spent resources',
        data: [83000, 49000, 60000, 35000, 77000, 90000],
        pointPlacement: 'on',
        color: '#f35958'
    }
]
}) ;

```

In this JavaScript code, we called the `highcharts` function for the selector chart `#performance-eval .spider-chart`. All the properties are fully described in the official documentation. Just note that we instructed the chart that we want a polar chart with the `polar: true` key inside the `chart` key option.

The dashboard application must look and feel like this:



Overhead loading

Another cool feature from these plugins is that they provide animations for charts, making the final result very user friendly.

By loading the pieces of JavaScript code at the end of the HTML, we will acquire more speed in page rendering for the end user. The side effect of this is that the elements created by the JavaScript libraries will render the page after it is shown to the user, causing some temporary glitches in the screen.

To solve this, many pages use the strategy of creating an overlay loading that will be hidden after the document is ready.

To do this, just after the opening of the `<body>` tag, create a `<div>` to keep the loading, as follows:

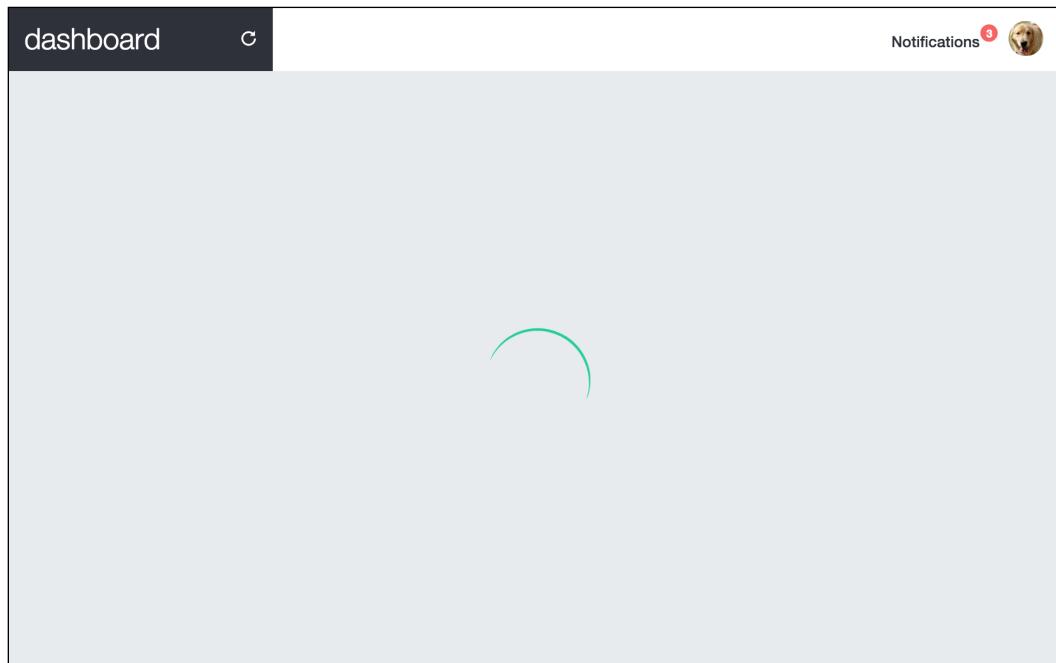
```
<body>
  <div class="loading">
    </div>
    ... <!--rest of the HTML content -->
</body>
```

We added a loading animated .svg file in the images folder, so in the CSS, we create the following rules:

```
.loading {
  position: fixed;
  z-index: 999;
  width: 100%;
  height: 100%;
  background-image: url('../imgs/loading.svg');
  background-repeat: no-repeat;
  background-attachment: fixed;
  background-position: center;
  background-color: #e5e9ec;
}
```

This will create an overlay element that will appear at the top of all elements, except for the navigation bar. This element will fill the complete width and height of the page, while containing the loading animated image in the center of it.

Refresh the page and you will see the loading on top of your dashboard web application, as follows:



Now we must remove the loading image after the page is ready. So, in the beginning of the JavaScript file, before the first line inside the `$(document).ready` function, remove the loading element:

```
$(document).ready(function() {  
    // when page is loaded, remove the loading  
    $('.loading').remove();  
    // below goes the rest of the JavaScript code  
});
```

Done! Refresh the web browser and you may see the loading screen depending on your computer and network.

The loading element may see an overreaction now, because we still do not have too many cards on our dashboard, but we will keep adding them, so it is cautious to start handling this problem.

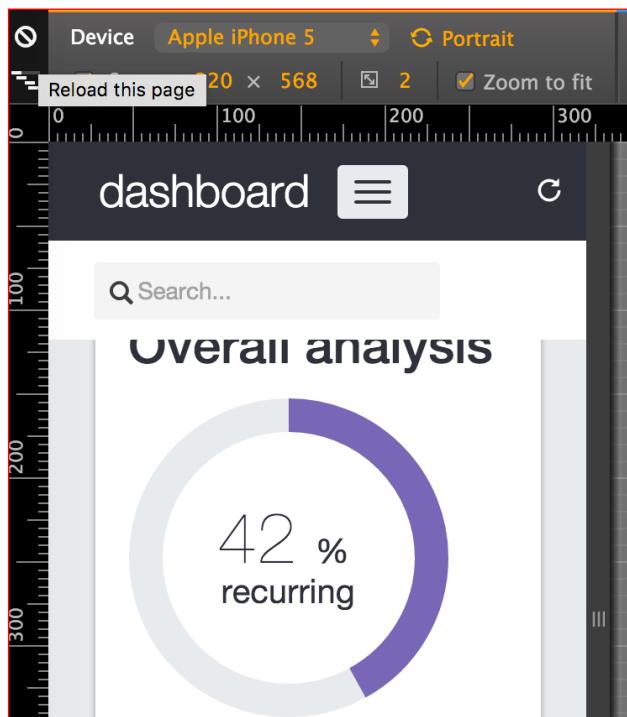
Fixing the toggle button for mobile

We created our page using the principles of mobile-first development, although some of the components here are not appearing properly or are simply not appearing, and we must fix that.

First is the toggle button, `.navbar-toggle`, in the navigation bar. It is actually appearing, but with a really bad colorization. Let's fix that with some CSS:

```
.navbar-toggle {  
    border-color: #252830;  
    background-color: #e5e9ec;  
    margin-top: 13px  
}  
  
.navbar-toggle .icon-bar {  
    background-color: #252830;  
}
```

The toggle button should appear like what is shown in the next screenshot, now using gray colors:



As you can see, there are many other things that we can do to improve the visualization in mobiles and small viewports. We will fix all that in the next chapters while adding some more cool effects. Wait and you will see!

Summary

In this chapter, we started another example – the dashboard web application.

At first, it may appear a little difficult, but we are breaking down every line of code to explain it while using the top of the methodologies for frontend development.

This time, we have an initial design that we aim to create. This is cool because we do have a guideline on what we must do towards our final goal. Usually, when working in a project, you have this kind of scenario.

First, we created another navigation bar, this one being a little more complicated, using a fluid container. The rest of the navigation was made using almost the same methodology that we used when learning about this Bootstrap component.

On the left-hand-side menu, we customized the Bootstrap stacked navigation component, and you learned how to use the Bootstrap collapse plugin.

In the main content, we started to import external plugins to create nice charts for our dashboard. Also, we used the flex display to increase responsiveness, while using the best of CSS.

Finally, we created a loading element and fixed the first issue regarding viewports. Let's continue fixing this in the upcoming chapters.

Congratulations! The first chapter of the final example is nailed! I am pretty sure that you were able to understand the development concepts and how Bootstrap greatly increases our productivity.

In the next chapter, we will continue the construction of the dashboard – moreover, the main content – adding more external plugins and cards using Bootstrap components. We will also fix known issues for different viewports and explain the remaining Bootstrap plugins.

10

Bringing Components to Life

The last chapter was tough! Although the dashboard is not ready yet, following our layout, we must create three more cards in the main, while fixing issues regarding visualization for multiple viewports. After that, we will move on to creating more components for our dashboard. Let's proceed towards this new challenge!

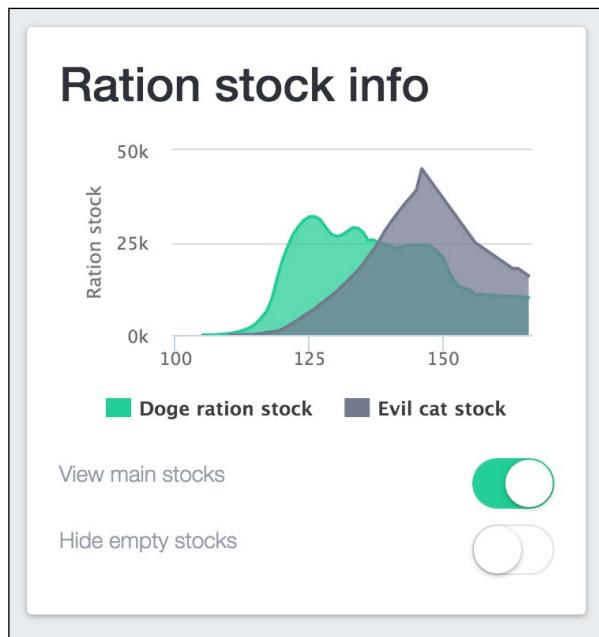
In this chapter, we will cover the following topics:

- A custom checkbox
- External plugin integration
- Advanced Bootstrap media queries
- The viewport's advanced adjustments
- The Bootstrap Carousel plugin
- The Bootstrap Scrollspy plugin

Creating the main cards

Taking a look at our layout, we must create three more cards. The first of them is the hardest one, so let's face it!

The following .card is composed of an area chart with two series and some iOS-styled checkboxes. This screenshot reminds us of what the card must look like:



For the chart area, we will again use the highcharts library, while for the checkbox, we will use a plugin called switchery (<https://github.com/abpetkov/switchery>). After we've considered the documentation, let's create the following HTML:

```
<div class="card" id="ration-stock">
  <div class="card-block">
    <h2>Ration stock info</h2>
    <div class="stacked-area"></div>
    <div class="switch">
      View main stocks
      <input type="checkbox" class="switchery" checked />
      <div class="clearfix"></div>
    </div>
    <div class="switch">
      Hide empty stocks
      <input type="checkbox" class="switchery" />
      <div class="clearfix"></div>
    </div>
  </div>
</div>
```

Breaking the code down, to create the chart, we just have to set the `div.stacked-area` element. For the checkbox, we must create an `input` with `type="checkbox"` and the `.switchery` class to identify it.

Load the CSS of switchery in `<head>`, after the Bootstrap CSS:

```
<link rel="stylesheet" href="css/switchery.min.css">
```

Also, in the HTML, import the `switchery` library in the bottom part that contains the JavaScript loads:

```
<script src="js/switchery.min.js"></script>
```

We do not need much CSS here, since most of it will be created by JavaScript. So, just add the following rules to specify the height of the chart and the font style for the checkbox text:

```
#ration-stock .stacked-area {  
    height: 200px;  
}  
  
#ration-stock .switch {  
    font-weight: 300;  
    color: #8b91a0;  
    padding: 0.5rem 0;  
}  
  
#ration-stock .switchery {  
    float: right;  
}
```

The JavaScript contains the core portion of this card. First, let's initialize the `switchery` plugin. In `main.js`, inside the `.ready` function, add these lines:

```
var elems, switcheryOpts;  
  
elems =  
Array.prototype.slice.call(document.querySelectorAll('.switchery'));  
  
switcheryOpts = {  
    color: '#1bc98e'  
};  
  
elems.forEach(function(el) {  
    var switchery = new Switchery(el, switcheryOpts);  
});
```

In `elems`, we store the elements that contain the `.switchery` class. This plugin does not use jQuery, so we must create a query using native JavaScript. The query needed to select the elements follows the one provided in the documentation, and I recommend that you check it out for further information, since this is not the main focus of the book.

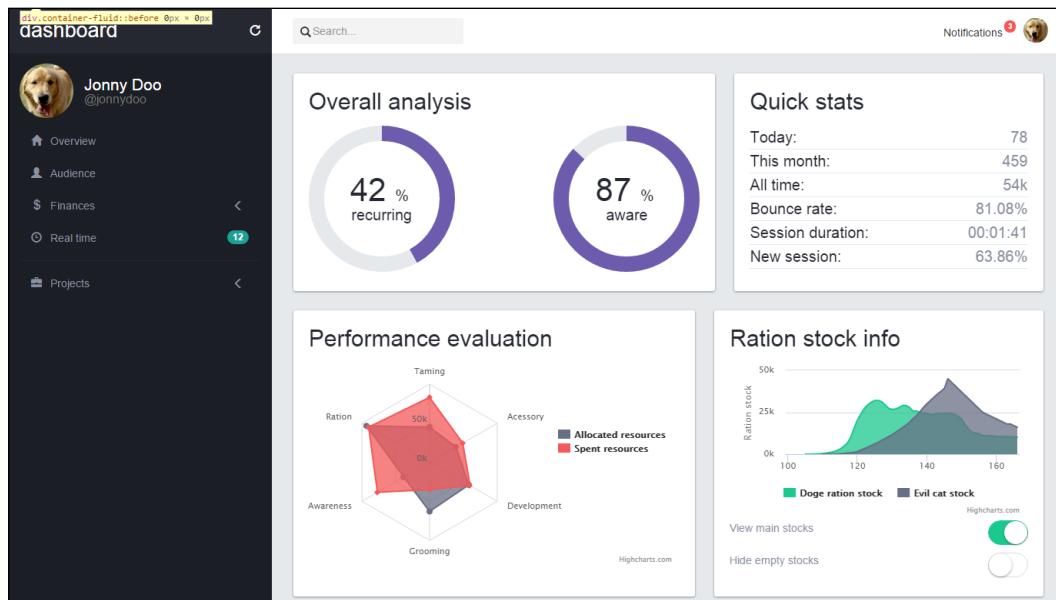
The query is performed by `document.querySelectorAll('.switchery')`. `Array` is a global JavaScript object used to create high-level list objects present in most recent browsers.

 The prototype is an object present on every JavaScript object. It contains a set of properties and methods for the regarding object.

The `slice` function chops the array using a shallow copy into another array. In summary, we are getting an array of elements with the `.switchery` class.

Next, we set the options for the plugin, in this case just the background color, using the `color` property in the `switcheryOpts` variable. Finally, we start each `Switchery` object inside the `forEach` loop.

Refresh the web page and the new card should appear as what is shown in the following screenshot:



The other card using Bootstrap components

To create the next card, we will use the Bootstrap progress bar component, labels, and badges. This card represents some kind of real-time information, and we will create it using the progress bar and make it animated through JavaScript.

First, let's create this new card identified by `#real-time` in the HTML. Place the code after the last card, `#ration-stock`:

```
<div class="card" id="real-time">
  <div class="card-block">
    <h2>Real time information</h2>
    </div>
  </div>
```

After `<h2>`, we must create a list containing each item of the information. A label, a badge, a progress bar, and a piece of sample text compose the list. Create it like the highlighted HTML code shown here:

```
<div class="card" id="real-time">
  <div class="card-block">
    <h2>Real time information</h2>
    <ul class="list-unstyled">
      <li>
        Active dogs:
        <span class="label label-warning pull-right">255</span>
      </li>
      <li>
        Silo status:
        <span class="badge ok pull-right">
          <span class="glyphicon glyphicon-ok" aria-hidden="true"></span>
        </span>
      </li>
      <li>
        Usage level:
        <div class="progress">
          <div class="progress-bar progress-bar-success"
            role="progressbar" aria-valuenow="25" aria-valuemin="0" aria-
            valuemax="100" style="width: 25%">
            <span class="sr-only">25%</span>
          </div>
          <div class="progress-bar progress-bar-warning progress-
            bar-striped active" role="progressbar" aria-valuenow="38" aria-
            valuemin="0" aria-value max="100" style="width: 38%">
        </div>
      </li>
    </ul>
  </div>
```

```
<span class="sr-only">38% alocated</span>
</div>
<div class="progress-bar progress-bar-danger"
role="progressbar" aria-valuenow="5" aria-valuemin="0" aria-
valuemax="100" style="width: 5%">
    <span class="sr-only">5% reserved</span>
    </div>
</div>
</li>
<li>
    Free space:
    <span id="free-space" class="pull-right">
        32%
    </span>
</li>
</ul>
</div>
</div>
```

Because we are mostly using only Bootstrap elements and components, we do not need too much CSS but just the following:

```
#real-time li {
    font-size: 1.8rem;
    font-weight: 300;
    border-bottom: 0.1rem solid #e5e9ec;
    padding: 0.5rem 0;
}

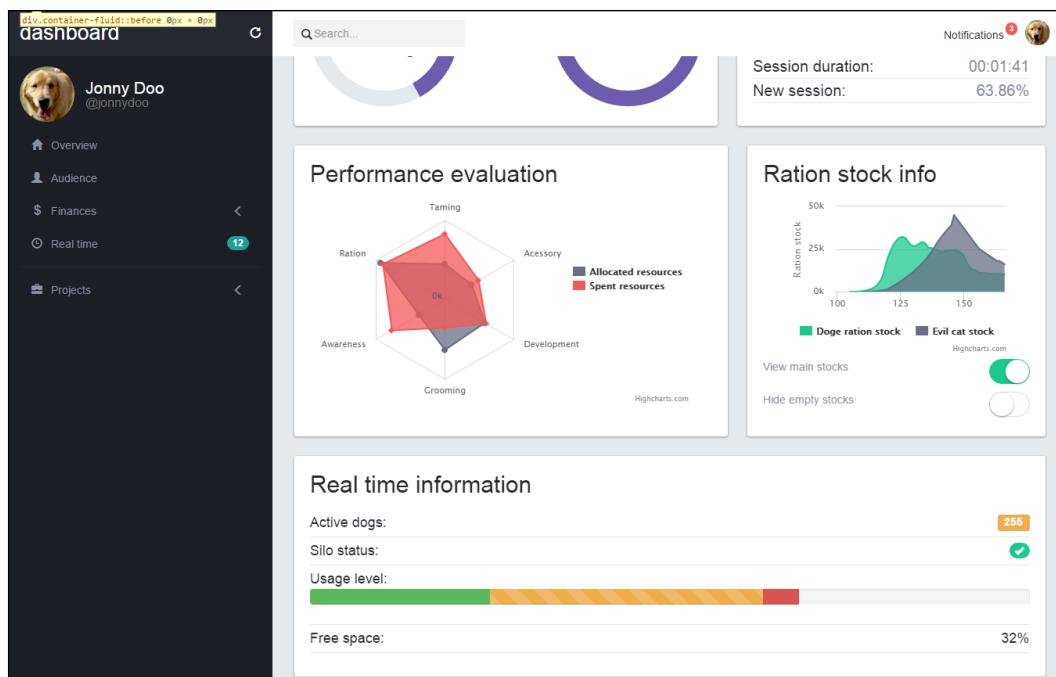
#real-time .badge.ok {
    background-color: #1bc98e;
}

#real-time .badge span,
#real-time .label {
    color: #FFF;
}

#real-time .badge,
#real-time .label {
    margin-top: 0.25rem;
}
```

This CSS will change the font size of the text in the card and the borders from one to another item in the list. Also, for the badge and the labels, we've customized the colors and margins.

Refresh the page and it should look like this:



The new card looks nice! Now let's create some CSS to animate it. Let's change the free space percentage periodically. To do this, create the following JavaScript function:

```
changeMultiplier = 0.2;
window.setInterval(function() {
    var freeSpacePercentage;

    freeSpacePercentage = $('#free-space').text();
    freeSpacePercentage = parseFloat(freeSpacePercentage);

    delta = changeMultiplier * (Math.random() < 0.5 ? -1.0 : 1.0);

    freeSpacePercentage = freeSpacePercentage + freeSpacePercentage
    * delta;
    freeSpacePercentage = parseInt(freeSpacePercentage);

    $('#free-space').text(freeSpacePercentage + '%');
}, 2000);
```

With this JavaScript code, we are executing a function every 2 seconds. We did this because of the usage of the `setInterval` function, and we call it every 2,000 ms (or 2 seconds).

What is done first is just a parse of the text inside the `#free-space` percentage element. Then we create a delta that could be 20 percent positive or negative, randomly generated by using the `changeMultiplier` parameter.

Finally, we multiply the delta by the current value and update the value in the element. To update the value in the element, we use the `.text()` function from jQuery. This function sets the content for the element to the specified text passed as a parameter; in this case, it's the percentage change in `freeSpacePercentage` that we randomly generated.

Refresh the page and see the number update every 2 seconds.

Creating our last plot

The last card in the main content is another plot, this time a pie chart. Just like the last charts, let's again use the Highcharts library. Remember that we must first create a simple HTML card, placed after the last `#real-time` card:

```
<div class="card" id="daily-usage">
  <div class="card-block">
    <h2>Daily usage</h2>
    <div class="area-chart"></div>
  </div>
</div>
```

In the CSS, just set the height of the plot:

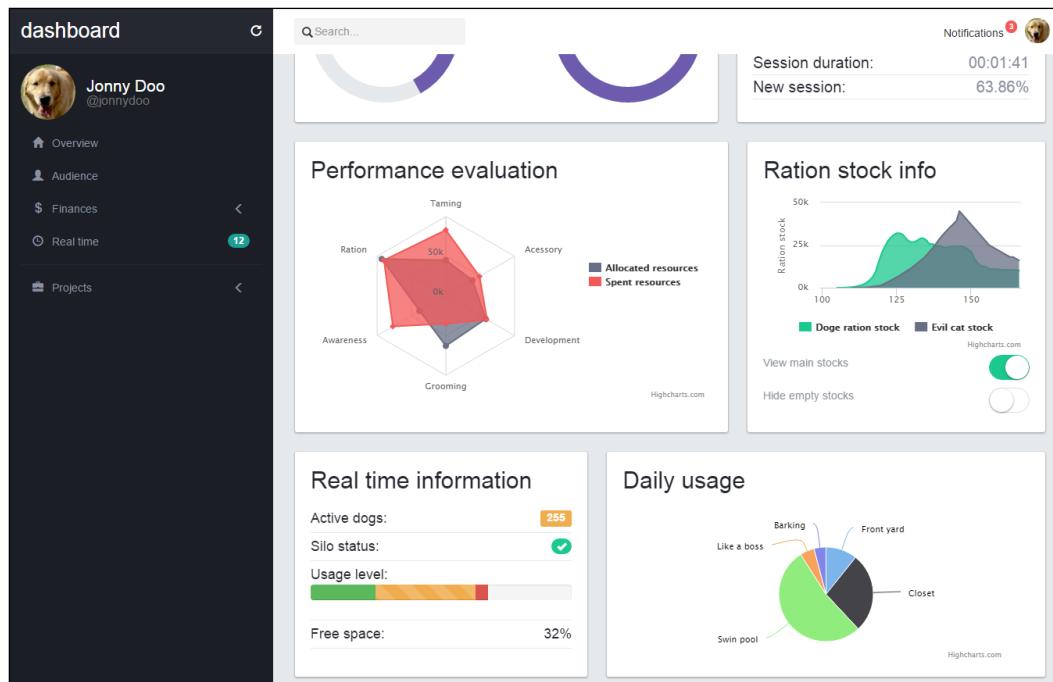
```
#daily-usage .area-chart {
  height: 200px;
}
```

To complete it—the most important part for this card—create the function calls in the JavaScript:

```
$( '#daily-usage .area-chart' ).highcharts( {
    title: {
        text: ''
    },
    tooltip: {
        pointFormat: '{series.name}:
<b>{point.percentage:.1f}%</b>'
    },
    plotOptions: {
        pie: {
            dataLabels: {
                enabled: true,
                style: {
                    fontWeight: '300'
                }
            }
        }
    },
    series: [{
        type: 'pie',
        name: 'Time share',
        data: [
            ['Front yard', 10.38],
            ['Closet', 26.33],
            ['Swim pool', 51.03],
            ['Like a boss', 4.77],
            ['Barking', 3.93]
        ]
    }]
});
```

As you can see in the preceding code, we set the graph to be of the `pie` type and create the share for each segment in the `data` array.

The following screenshot shows how the last card must be displayed on the web browser:

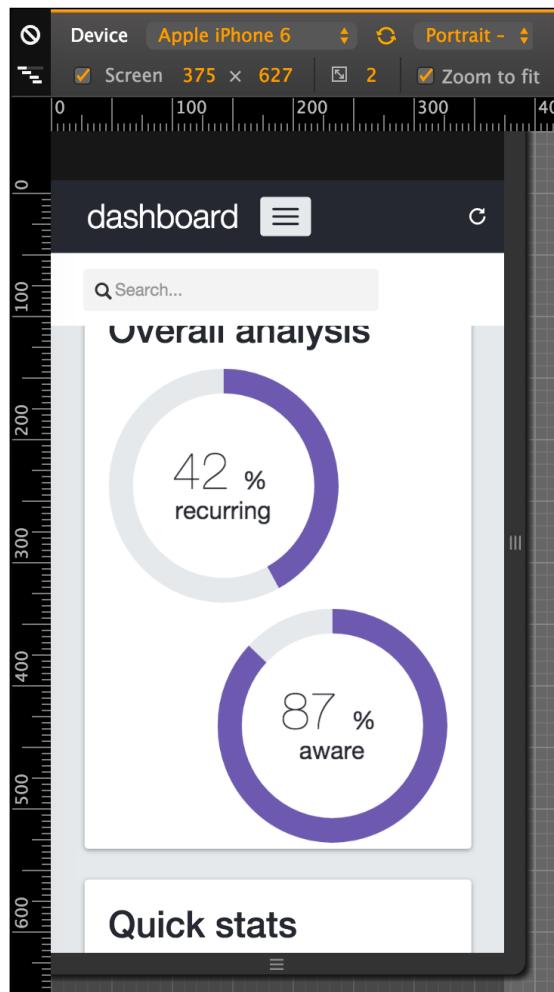


And we are done! The main page of the dashboard is complete. Now let's proceed to the next pages in this component.

Fixing the mobile viewport

If you resize the dashboard to a mobile visualization (treated as an extra-small viewport in Bootstrap,) you should see some problems with the elements that are not appearing correctly. As shown in the next screenshot, note that the search appears and the card with the round chart is completely unaligned.

In this visualization mode, we are using the viewport of iPhone 6 in portrait orientation in the Chrome developer inspector:



Regarding the search bar, it will be better if this bar appears just when required, for example, when clicking on a button. So, next to the refresh button, let's create another icon to toggle the search bar.

The HTML for this section must be like the following code:

```
<div class="col-sm-3 top-left-menu">
  <div class="navbar-header">
    <a class="navbar-brand" href="dashboard.html">
      <h1>dashboard</h1>
    </a>

    <button type="button" class="navbar-toggle collapsed" data-
    toggle="collapse" data-target="#nav-menu" aria-expanded="false">
      <span class="sr-only">Toggle navigation</span>
      <span class="icon-bar"></span>
      <span class="icon-bar"></span>
      <span class="icon-bar"></span>
    </button>
  </div>
  <a href="#" id="search-icon" data-toggle="tooltip" data-
  placement="bottom" data-delay="500" title="Display search bar"
  class="header-buttons pull-right visible-xs">
    <span class="glyphicon glyphicon-search" aria-hidden="true"></
  span>
  </a>
  <a href="#" data-toggle="tooltip" data-placement="bottom" data-
  delay="500" title="Refresh data" class="header-buttons pull-right">
    <span class="glyphicon glyphicon-repeat" aria-hidden="true"></
  span>
  </a>
</div>
```

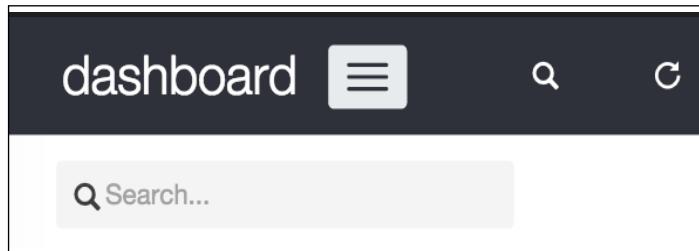
Let's discuss this code. First, we made a change in the class name. The link in the refresh icon was a `.header-refresh`. Now, since we have multiple header buttons, we changed it to a `.header-button` class for generalization.

We also added the Bootstrap tooltip for this button, just as we did for the refresh icon, displaying the message: "Display search bar".

To complete the changes, replace the class names in the CSS as well:

```
nav .header-buttons {
  margin-left: auto;
  color: #FFF;
}
```

Then the header should look like this:



Now we have to fix the search bar. Let's change the classes on the `form#search`. Replace the classes from `.hidden-sm.col-md-3` to just `.col-sm-3` for better visualization.

 Remember the gridding foundations? By setting the `form` for this class, it will fill 3 out of 12 columns in the template until the small viewports and appear as a line block for extra small viewports.

Now, let's hide the form using a media query in CSS for extra small viewports:

```
@media (max-width:48em) {
  form#search {
    display: none;
  }
}
```

To toggle the visualization of the search input, let's add some JavaScript events. The first one is for opening the search when we click on the magnifier icon at the header, identified by `#search-icon`. So in our `main.js` file, we add the following function:

```
$('#search-icon').on('click', function(e) {
  e.preventDefault();
  $('form#search').slideDown('fast');
  $('form#search input:first').focus();
});
```

What this will do first is prevent the default click action with the `e.preventDefault()` caller. Then, we use the `.slideDown` function from jQuery, which slides down an element. In this case, it will toggle `form#search`.

After toggling the form, we add focus to the input, which will open the keyboard if we are accessing the page from a mobile phone.

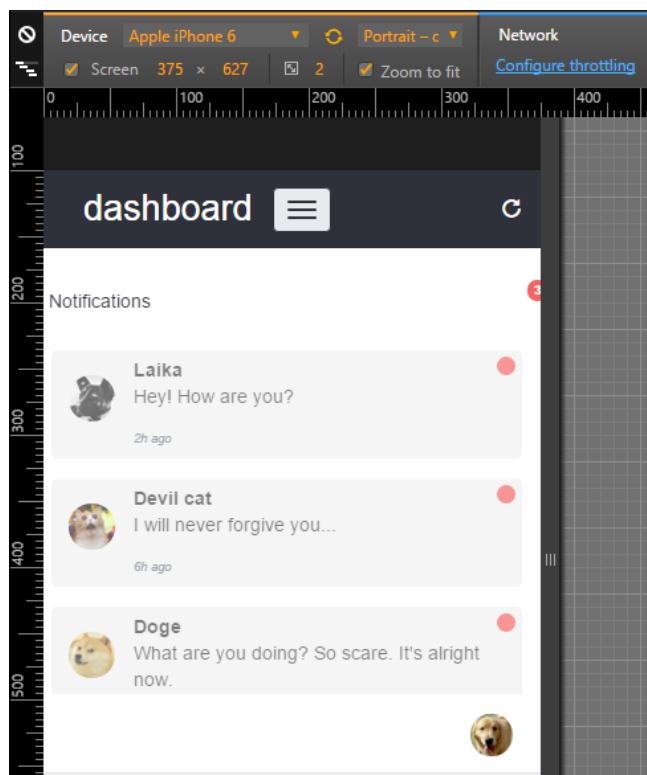
To increment that, it would be nice if the search bar can hide when the user blurs the focus on the search input. To do this, add the following event handler to the JavaScript:

```
$( 'form#search input' ).on('blur', function(e) {  
    if($('#search-icon').is(':visible')) {  
        $('form#search').slideUp('fast');  
    }  
});
```

What we are doing here is using the `blur` event, which is triggered whenever the element loses the focus. The trigger performs a check to find out whether the `#search-icon` is visible, meaning that we are in the extra small viewport, and then hides the search bar using the `slideUp` function, doing the opposite of what the `slideDown` function does.

Fixing the navigation menu

Click on the collapse toggle navigation (the hamburger button) and you will see how the `#nav-menu` looks so messy, as shown in the next screenshot. We must fix it just like the way we did in the last web application example:



To do this, we will first need to remove the `.pull-right` class from `#nav-menu`. The `.pull-*` classes add a float to the element by applying the `!important` flag, which cannot be overridden. In this case, we must override this style rule to remove the `.pull-right` class and add the float to the current element style rule:

```
#nav-menu {
    float: right;
}
```

Create a media query for extra small devices for `#nav-menu` and remove the `float: right`:

```
@media (max-width:48em) {
    #nav-menu {
        float: none;
    }
}
```

After that, we must hide `#nav-profile` and move its button to the `#nav-menu` list. First, add the `.hidden-xs` class to the profile element:

```
<div id="nav-profile" class="btn-group pull-right hidden-xs">
    ...
</div>
```

This will prevent the element from appearing for extra small devices using the Bootstrap viewport helper class. Then, in `#nav-menu > ul`, append the options that were in the `#nav-profile` drop-down button:

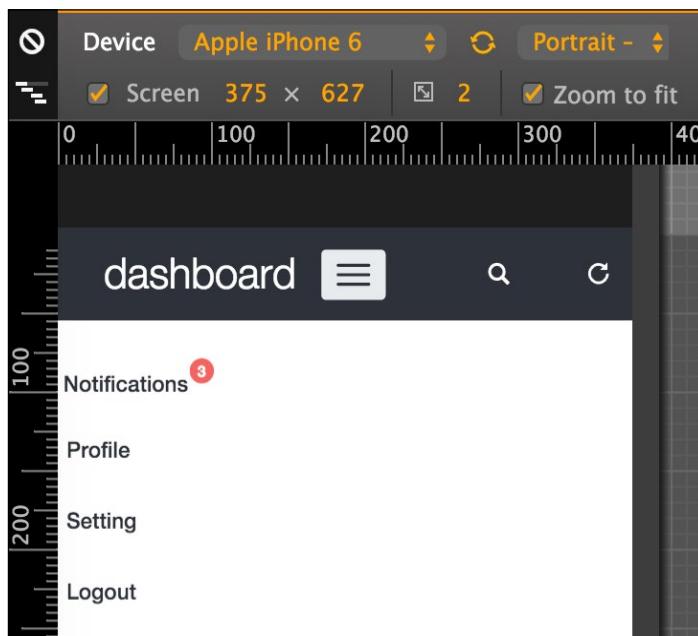
```
<div id="nav-menu" class="collapse navbar-collapse">
    <ul class="nav navbar-nav">
        <li>...</li>
        <li class="visible-xs">
            <a href="#">Profile</a>
        </li>
        <li class="visible-xs">
            <a href="settings.html">Setting</a>
        </li>
        <li class="visible-xs">
            <a href="#">Logout</a>
        </li>
    </ul>
</div>
```

Note that we make this new item list visible only for extra small viewports with the `.visible-xs` class.

These new item lists must now look just like the notification one, already present in this list. So, append the selector of the new item list to the current CSS style of `#btn-notification`:

```
#btn-notifications .btn-link,  
#nav-menu li a {  
    padding-top: 1.5rem;  
    color: #252830;  
    font-weight: 500;  
}
```

The opened list should look like this:



Now, try to change the viewport and see how the elements on the header correctly change its visualization. The `#nav-profile` will appear only for small-to-large viewports and will shrink into `#nav-menu ul` in a small visualization for extra small viewports.

The notification list needs some style

If you click on the notification list to open it, you will see three problems: firstly, the badge holding the number of new notifications jumps to the right portion; then the notification button is not filling the entire width; and finally, the notification list can appear a little nicer when opened.

To fix the jumping badge on the notification button, just add the following CSS:

```
@media (max-width:48em) {
    #nav-menu #btn-notifications > .badge {
        right: inherit;
        left: 10rem;
    }
}
```

Note that we use a media query to change the position of the badge for extra small viewports only.

To modify the notification button's width, we have to create a media query as well. So, add this CSS style to it:

```
@media (max-width:48em) {
    #btn-notifications,
    #btn-notifications > button {
        width: 100%;
        text-align: left;
    }
}
```

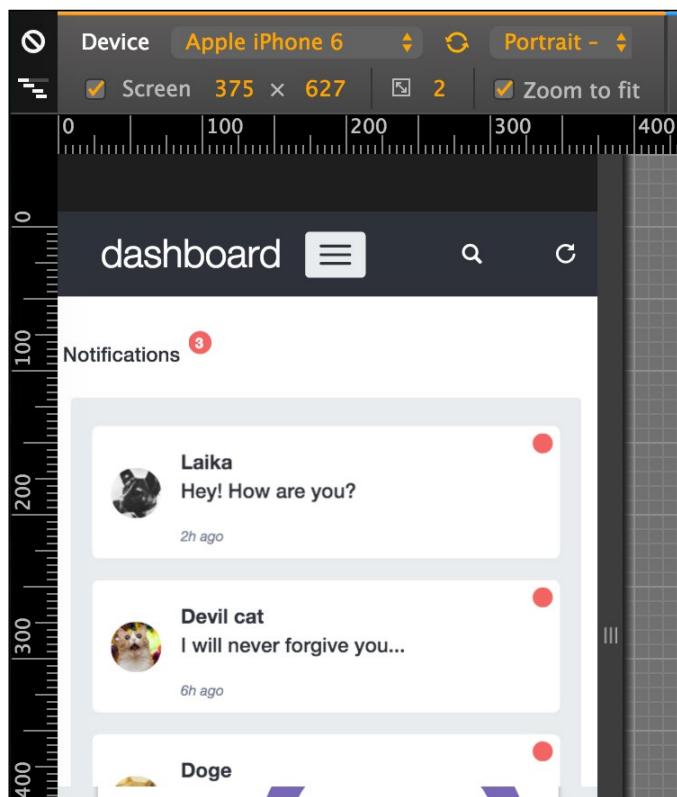
This style will change the width for both the notification button dropdown and the button itself.

Finally, the style for the notification list must be changed. We create the next CSS rule in our `main.css` file, and it should instantly look good:

```
@media (max-width:48em) {
    #notification-list {
        margin: 1.25rem;
        margin-left: 2rem;
        background-color: #e5e9ec;
    }

    #notification-list a {
        background-color: #FFF;
        opacity: 1;
    }
}
```

Awesome! Update your web browser and #notification-list should look like what is shown in this screenshot:



Adding the missing left menu

Where are the items of the left menu? If you check out the HTML of `#side-menu`, you will see that we have added the `.hidden-xs` class to it. So, we must move the navigation options to another place in this extra small viewport.

Let's add the links to #nav-menu ul just as we did for #nav-profile:

```
<div id="nav-menu" class="collapse navbar-collapse">
  <ul class="nav navbar-nav">
    <li>...</li>
    <li class="visible-xs">
      <a href="#">Audience</a>
    </li>
    <li class="visible-xs">
      <a href="#">Finances</a>
    </li>
    <li class="visible-xs">
      <a href="#">Realtime</a>
    </li>
    <li class="visible-xs">
      <a href="#">Projects</a>
    </li>

    <li role="separator" class="divider visible-xs"></li>
    ...
  </ul>
</div>
```

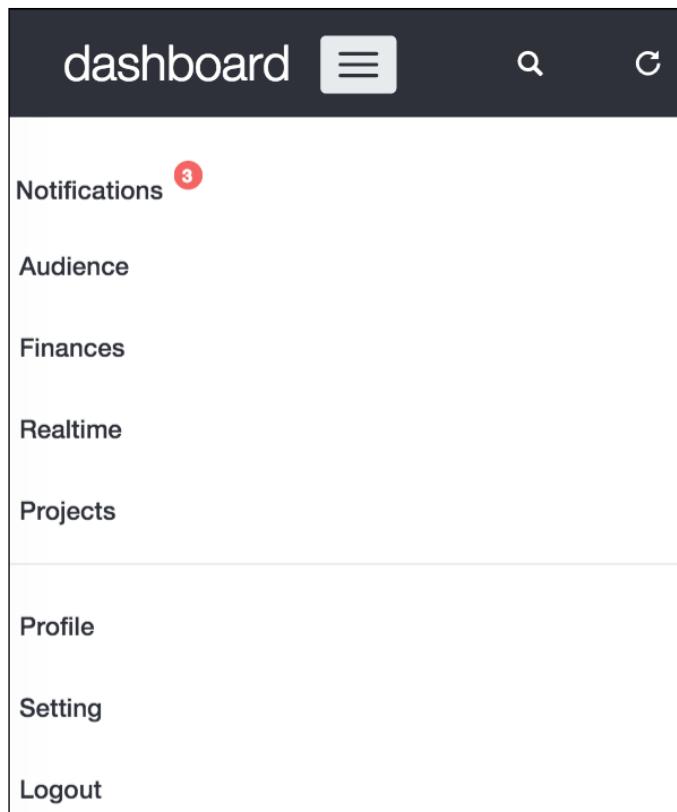
Modify the maximum height of #nav-menu when collapse is toggled with the style:

```
#nav-menu.navbar-collapse {
  max-height: 39rem;
}
```

For the .divider element in the list, create the following CSS:

```
#nav-menu .divider {
  height: 0.1rem;
  margin: 0.9rem 0;
  overflow: hidden;
  background-color: #e5e5e5;
}
```

Notice that in `#nav-menu ul`, the notification button will appear above the new elements added and the options from `#nav-profile` will appear below. The next screenshot represents the visualization of the final arrangement of the `#nav-menu` toggle:



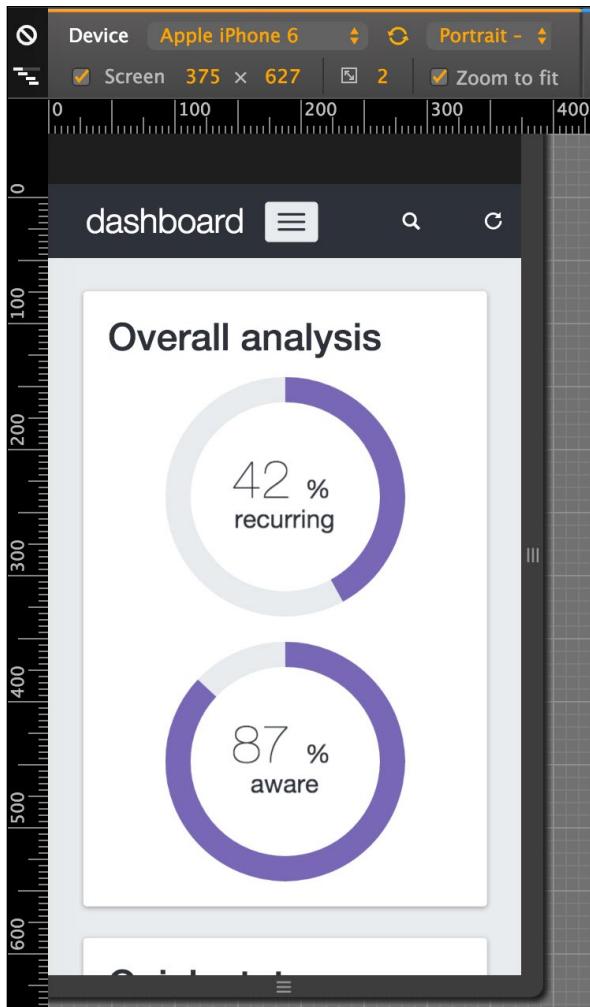
Aligning the round charts

The `.round-charts` inside the `#pie-charts` element does not appear correctly aligned. However, we can quickly fix this with two CSS rules using media queries. So, create the following style:

```
@media (max-width:48em) {  
    .round-chart,  
    .round-chart canvas {  
        display: block;  
        margin: auto;  
    }  
}
```

```
.round-chart + .round-chart {  
  margin-top: 2rem;  
  float: none;  
}  
}
```

Refresh the web page and see the result, like the following screenshot:



Great! Now we have our dashboard nailed for every viewport and device! That was a thorough task, but with a great payoff, because we have now created a complete dashboard. Let's move forward to some other pages in our example.

Learning more advanced plugins

Now that we have created the main page of the dashboard example and nailed almost every element, plugin, and component in Bootstrap, let's use some other advanced JavaScript plugins to complete our journey.

For this part, let's create another file named `audience.html` in the same folder of `dashboard.html`. In this file, copy the exact same code of `dashboard.html`, except the HTML inside the `div#main` element, because that is where we will make some new changes.

Using the Bootstrap carousel

Bootstrap provides us with a nice plugin to slideshow components through cycling elements, although it's pretty verbose and a little complicated to understand at first sight.

First of all, we need to create an element inside our `div#main`:

```
<div id="main" class="col-sm-offset-3 col-sm-9">
  <div id="carousel-notification" class="carousel" data-
    ride="carousel">
    ...
  </div>
</div>
```

We must identify this element for the Bootstrap Carousel plugin, so we have called it `#carousel-notification` at our outmost `div` of the plugin.

Bootstrap will start a carousel via the data attributes for elements marked with `data-ride="carousel"`, just like our element. In addition, this element must have the `.carousel` class for the CSS style.

We must create the elements inside the slides for the carousel, so we use the following markup to create the notification slides:

```
<div id="main" class="col-sm-offset-3 col-sm-9">
  <div id="carousel-notification" class="carousel" data-
    ride="carousel">
    <div class="carousel-inner" role="listbox">
      <div class="item active">
        
        <div class="carousel-caption">
          <p>What are you doing? So scare. It's alright now.</p>
        </div>
      </div>
      <div class="item">
```

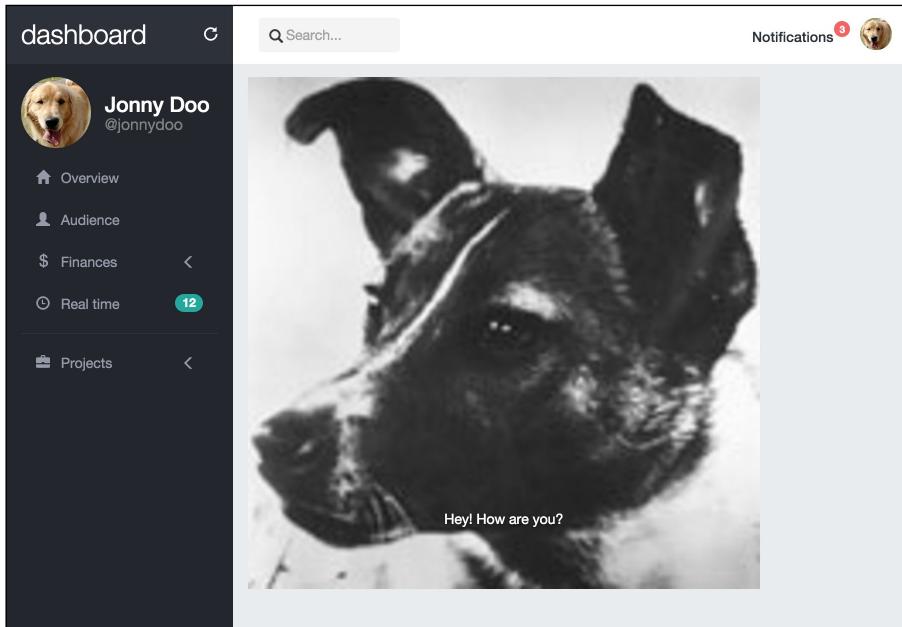
```

<div class="carousel-caption">
    <p>I will never forgive you...</p>
</div>
</div>
<div class="item">
    
    <div class="carousel-caption">
        <p>Hey! How are you?</p>
    </div>
</div>
</div>
</div>
```

Note that we have created three items. Each item has been created inside the `.carousel-inner` element. Inside this element, the items have been created with the `.item` class.

Inside each `.item`, there is an image followed by another element with the `.carousel-caption` class, which contains text to be displayed as captions for each slide. Note that the first slide also contains the `.active` class, which must necessarily be added to one (and only one) of the slides.

At this point, refresh Bootstrap carousel your browser and you should see the page like what is shown in this screenshot:



If you wait 5 seconds, you will see the image and caption change. You can set this interval value by the `data-interval` data attribute or through JavaScript, as an initializer parameter, as shown in this example:

```
$('.carousel').carousel({
  interval: 1000 // value in milliseconds
})
```

Also observe that there is no animation between the changes of the slides. To add it, put the `.slide` class into the `.carousel` element and you will see a left slide of the images.



Remember that versions 8 and 9 of Internet Explorer do not support CSS animations. Therefore, the Bootstrap carousel plugin will work unless you add transition fallbacks on your own, such as jQuery animations.

Customizing carousel items

For each item in the carousel, we just created a simple `<p>` element inside it. However, you can add other elements, as in the following example, where we are adding a heading 3:

```
<div class="item">
  
  <div class="carousel-caption">
    <h3>Laika said:</h3>
    <p>Hey! How are you?</p>
  </div>
</div>
```

Creating slide indicators

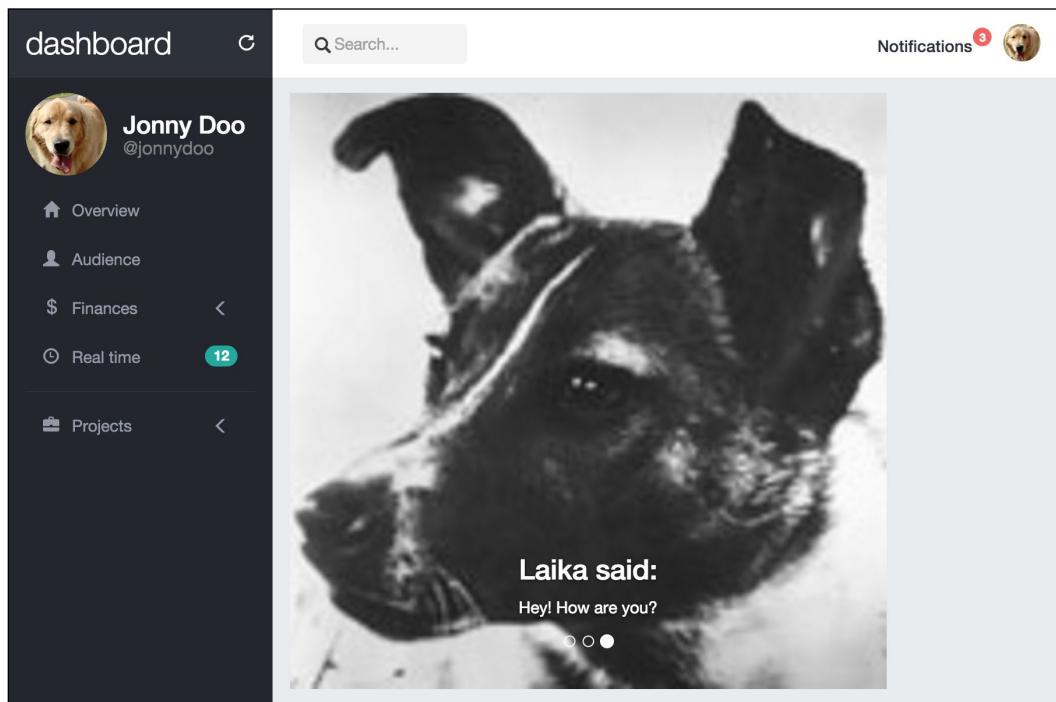
The Bootstrap carousel also offers the ability to create bullet slide indicators. To do this, add the following code after the `.carousel-inner` element:

```
<div id="carousel-notification" class="carousel slide" data-
ride="carousel">
  <div class="carousel-inner" role="listbox">
    ...
  </div>
```

```
<!-- Indicators -->
<ol class="carousel-indicators">
  <li data-target="#carousel-notification" data-slide-to="0" class="active"></li>
  <li data-target="#carousel-notification" data-slide-to="1"></li>
  <li data-target="#carousel-notification" data-slide-to="2"></li>
</ol>
</div>
```

We just created an ordered list, ``. On each item, we have to say which is the carousel element identifier through `data-target` (in this case, it is `#carousel-notification`) and which slide each bullet will correspond to through `data-slide-to`. To do this, we just create the number of list items from the same size of the image items and enumerate them.

Refresh the browser, and now you should see the carousel with the bullets and all the modifications (the slide transition, the heading on the image item, and the bullet identifier), like this:



Adding navigation controls

Another cool option in the Bootstrap carousel is creating side navigation controls to change slides from left to right.

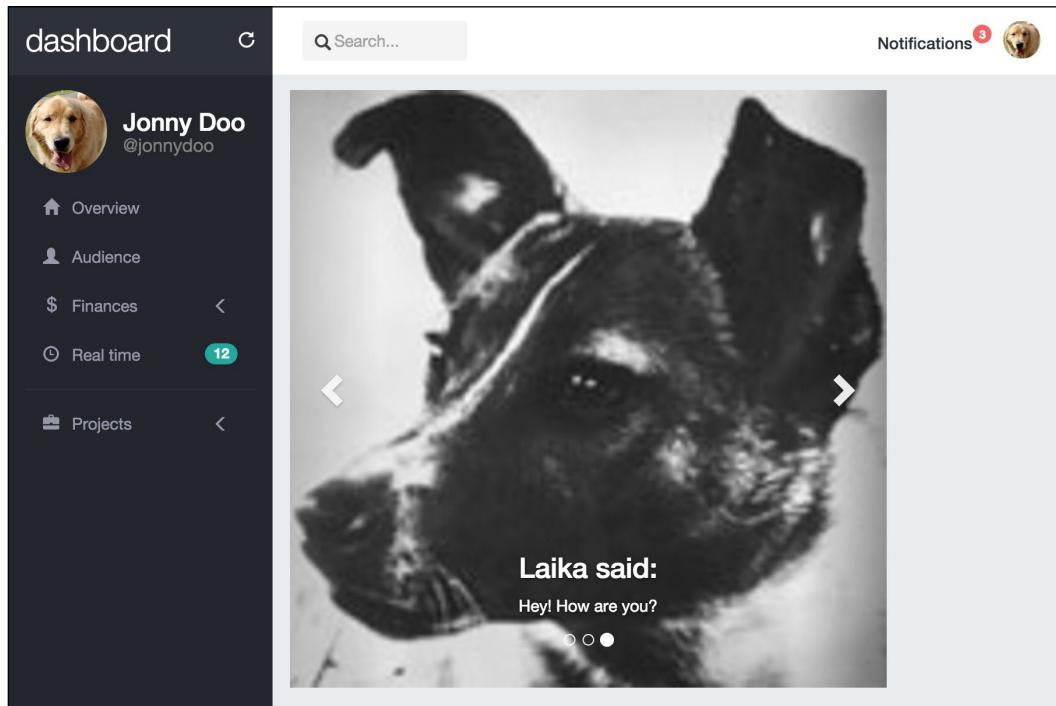
We add the markup for this after the indicator's one, as shown in the following HTML code:

```
<div id="carousel-notification" class="carousel slide" data-  
ride="carousel">  
  <div class="carousel-inner" role="listbox">  
    ...  
  </div>  
  
  <!-- Indicators -->  
  <ol class="carousel-indicators">  
    </ol>  
  
  <!-- Controls -->  
  <a class="left carousel-control" href="#carousel-notification"  
role="button" data-slide="prev">  
    <span class="glyphicon glyphicon-chevron-left" aria-  
hidden="true"></span>  
    <span class="sr-only">Previous</span>  
  </a>  
  <a class="right carousel-control" href="#carousel-notification"  
role="button" data-slide="next">  
    <span class="glyphicon glyphicon-chevron-right" aria-  
hidden="true"></span>  
    <span class="sr-only">Next</span>  
  </a>  
</div>
```

As you can see, we created two carousel controls, one to the right and one to the left. Each of them must be inside an `<a>` tag identified by the `.carousel-control` class and the class for the action, which is `.right` or `.left`.

The `href` of the element represents the identifier of the carousel, just as `data-target` in the bullet indicators. The `data-slide` indicates the action that should be performed by the control, which can be `next` to move to the next slide or `prev` to move to the previous slide.

The next screenshot presents the final expected result of the carousel:



Using multiple Bootstrap carousels on the same page



If you plan to use multiple Bootstrap carousels on the same page, remember to correctly apply a unique `id` to the parent element (the one with the `.carousel` class). Also remember to update the target for the bullet indicator and slide controls.

Other methods and options for the carousel

Just like every Bootstrap plugin, the carousel offers a set of parameters and methods that can be used. Check out the official documentation for detailed info (<http://getbootstrap.com/javascript/#carousel-options>).

There are some options that we should be talking about, such as `wrap`, which defines whether the carousel should be cyclic or not. By default, this option has the value `true`.

You can also call via JavaScript to the carousel go to a certain slide or just force a slide switch. To go to a certain slide, use this function:

```
$('.carousel').carousel(2); // which 2 is the slide enumerated as  
2 in the data-slide-to
```

To call the carousel to switch slides, use the same function but pass the prev or next string as the argument:

```
$('.carousel').carousel('next') // or 'prev'
```

Just like other Bootstrap plugins, the carousel is great for creating slide images on your page. There are a plenty of customizations available to fit the required styles. Always check out the documentation for further information.

The Bootstrap spy

You will now learn another Bootstrap plugin—Bootstrap Scrollspy. Scrollspy is a plugin used to automatically update any kind of Bootstrap navigation based on the scroll position. Many sites use it, including the Bootstrap documentation, in the side navigation. There, when you scroll the page, the active elements in the navigation bar change.

To exemplify the utilization of the plugin, let's create a .card on our audience.html page:

```
<div class="card">  
  <div class="card-block">  
  
    </div>  
  </div>
```

Inside .card-block, we will create two columns, the left one for the spy navigation and one to the right for the content itself. Remember to always place your .col-*-* inside a .row element:

```
<div class="card">  
  <div class="card-block">  
    <div class="row">  
      <div class="col-sm-3" id="content-spy"></div>  
      <div id="content" class="col-sm-9"></div>  
    </div>  
  </div>  
</div>
```

We are identifying the navigation column as `#content-spy` and the content column as just `#content`.

First, let's create the left navigation using the `.nav-pills.nav-stacked` Bootstrap component. Do you remember it? Let's refresh your memory by using it again, as follows:

```
<div class="card">
  <div class="card-block">
    <div class="row">
      <div class="col-sm-3" id="content-spy">
        <ul class="nav nav-pills nav-stacked">
          <li role="presentation" class="active">
            <a href="#lorem">The Lorem</a>
          </li>
          <li role="presentation">
            <a href="#eros">The Eros</a>
          </li>
          <li role="presentation">
            <a href="#vestibulum">The Vestibulum</a>
          </li>
        </ul>
      </div>
      <div id="content" class="col-sm-9"></div>
    </div>
  </div>
</div>
```

All we need to do is create a `` with the `.nav`, `.nav-pills`, and `.nav-stacked` classes. Then we create three item lists, each one with a link inside, referencing an ID in the HTML (`#lorem`, `#eros`, and `#vestibulum`). We will use these IDs later to refer to the Scrollspy.

Now create the content. It must be in the `#content` three `<div>`, each one with the ID corresponding to the references in the `href` of the link in the item list, as shown in this code:

```
<div class="card">
  <div class="card-block">
    <div class="row">
      <div class="col-sm-3" id="content-spy">
        <ul class="nav nav-pills nav-stacked">
          <li role="presentation" class="active">
            <a href="#lorem">The Lorem</a>
          </li>
```

```
<li role="presentation">
    <a href="#eros">The Eros</a>
</li>
<li role="presentation">
    <a href="#vestibulum">The Vestibulum</a>
</li>
</ul>
</div>
<div id="content" class="col-sm-9">
    <div id="lorem">
        <h2>The Lorem</h2>
        <p>
            Lorem ipsum dolor sit amet... <!-- Rest of the text -->
        </p>
    </div>
    <div id="eros">
        <h2>The Eros</h2>
        <p>
            Curabitur eget pharetra risus... <!-- Rest of the text -->
        </p>
    </div>
    <div id="vestibulum">
        <h2>The Vestibulum</h2>
        <p>
            Integer eleifend consectetur... <!-- Rest of the text -->
            
        </p>
    </div>
    </div>
</div>
</div>
```

Note the identifiers on each `<div>` corresponding to the `href` in the link of the item list. This is used to correlate the scroll with the active item in the `nav` element.



Do not forget to add the `.img-responsive` class to the image at the end of the third content item.

To activate the plugin, we have two options. Activate it by data attributes or by JavaScript. If you choose JavaScript, place the call function in `main.js`:

```
$( '#content' ).scrollspy({  
    target: '#content-spy'  
})
```

Refresh the page and see it working. If you want to use data attributes, place a `data-spy` and a `data-target` in the `#content` element:

```
<div class="card">  
    <div class="card-block">  
        <div class="row">  
            <div class="col-sm-3" id="content-spy">  
                <ul class="nav nav-pills nav-stacked">  
                    <li role="presentation" class="active">  
                        <a href="#lorem">The Lorem</a>  
                    </li>  
                    <li role="presentation">  
                        <a href="#eros">The Eros</a>  
                    </li>  
                    <li role="presentation">  
                        <a href="#vestibulum">The Vestibulum</a>  
                    </li>  
                </ul>  
            </div>  
            <div id="content" class="col-sm-9" data-spy="scroll" data-  
target="#content-spy">  
                <div id="lorem">  
                    <h2>The Lorem</h2>  
                    <p>  
                        Lorem ipsum dolor sit amet... <!-- Rest of the text -->  
                    </p>  
                </div>  
                <div id="eros">  
                    <h2>The Eros</h2>  
                    <p>  
                        Curabitur eget pharetra risus... <!-- Rest of the text -->  
                    </p>  
                </div>  
                <div id="vestibulum">  
                    <h2>The Vestibulum</h2>  
                    <p>
```

```
Integer eleifend consectetur... <!-- Rest of the text -->

</p>
</div>
</div>
</div>
</div>
</div>
```

The `data-spy` must have the `scroll` value in order to identify that the spy must be active for the scrolling action. The `data-target` works just like the parameter target passed in the activation by JavaScript. It should represent the element that will spy on the element, `#content-spy` in this case.

To make a final effect for the scrolling, create the following CSS to limit the height and adjust the scroll of the content:

```
#content {
    height: 30em;
    overflow: auto;
}
```

Refresh your web browser and the card should appear like what is shown in the next screenshot. Note that here we have scrolled the content to the second item.

The screenshot shows a user interface with a dark sidebar on the left and a light-colored main content area on the right. In the sidebar, there are several navigation items: 'Overview', 'Audience', 'Finances' (with a '12' badge), 'Real time' (with a '12' badge), and 'Projects'. The main content area features two cards. The first card, titled 'The Eros', is highlighted with a blue background and white text. Its content is placeholder text: 'Curabitur eget pharetra risus, ac bibendum magna. Pellentesque dapibus ipsum a metus feugiat, ac rutrum quam pretium. Morbi scelerisque, elit in tincidunt vehicula, sem nulla condimentum dui, vitae sollicitudin tellus ligula ut justo. Proin consequat at velit sed lacinia. Nunc ac tellus molestie, consequat odio vel, tristique felis. Aliquam porttitor, justo ac aliquet vehicula, erat quam dictum est, nec facilisis mi ligula ut purus. Donec viverra mauris metus, aliquet porta quam vulputate quis. Sed bibendum tortor tortor, et malesuada lectus faucibus at. Proin lectus elit, congue eu dapibus ac, maximus commodo nunc. Vestibulum nunc augue, scelerisque sed augue at, viverra pellentesque magna. In non mauris id justo vehicula congue ultrices at ante. Nulla in ipsum diam. Nam in lacus ipsum.' The second card, titled 'The Vestibulum', is partially visible below it. Both cards have a similar structure with a title, a short paragraph of text, and a horizontal dashed line at the bottom.

Great! Now you have learned another Bootstrap plugin! The Scrollspy plugin is very useful, especially on pages with extensive content, subdivided into sections. Make great use of it.

Summary

In this chapter, we greatly finished our dashboard example. We had to create some cards from the last chapter and adjust the visualization for any viewport. The result is a great dashboard that can be used in multiple contexts and web applications.

The user experience and the visuals of the dashboard were impressive in the end, creating a desirable page for this kind of application that works on any device.

Then we moved forward to explain some more Bootstrap plugins. We analyzed Bootstrap Scrollspy, which is a great plugin for you when you are creating pages with large content and need to summarize the sections while the user is scrolling. We used this plugin just with sample content, but remember to use it whenever needed.

We analyzed the Bootstrap carousel as well. The carousel is a great plugin for making slides of images with caption text. In my opinion, the only downside to this plugin is that it is too much typed. Imagine if we could create the same carousel using lines of code. I think we can fix that in the last chapter!

In the next chapter, we will start a plugin customization and use the carousel as an example. We can create a kind of wrapper to reduce typing of the carousel and automate plugin creation. Also, we will go deep into some Bootstrap plugin customizations.

For sure, it will be another challenge to create a plugin for Bootstrap, but I am confident that we can nail that as well.

11

Making It Your Taste

At this point, you can be called a Bootstrap master around the world! You nailed the framework as few people do these days – you should be proud of that!

Now, you are about to face a challenge to overpass the boundaries of learning. In this chapter, we will see how to create and customize your own Bootstrap plugin. This could be tough, but if you reached this point you can go a step further to become a true Bootstrap master.

The topics covered are as follows:

- Customizing Bootstrap components
- Customizing Bootstrap plugins
- Creating a Bootstrap plugin

When we finish this chapter, we will also reach the end of the book. I hope this last chapter will help you empower yourself with all the Bootstrap framework skills.

To follow this chapter, create a `sandbox.html` file and just place the default template that we are using all over the book. We will place all the code snippets of this chapter in this file.

Customizing a Bootstrap component

In my years of experience of using Bootstrap, one of the major issues that I received is how can I change a Bootstrap component to appear like I need?

Most of the time, the answer is to take a look at the CSS and see how you can override the style. However, this orientation can be obscure sometimes and the developer will be unable to find a solution.

In this section, we will customize some Bootstrap components. We did some of that in previous chapters, but now we will go a step further into this subject. Let's start customizing a single button.

The taste of your button

We must start with a button, because of two factors. First, it is a quite simple component and second we have to customize a button very often.

Let's assume we have a simple button placed in a page that already has the Bootstrap fully loaded. We will call it as the sandbox page. The HTML for it should be like this:

```
<button type="button" class="btn btn-primary" aria-pressed="false"  
autocomplete="off">  
    This is a simple button  
</button>
```

As we saw so many times, this button is a simple one with the `.btn` and `.btn-default` classes that will make the button blue, as shown in the next screenshot:



If you want a different color for the button, you can use one of the others contextual classes provided by Bootstrap (`.btn-success`, `.btn-info`, `.btn-warning`, `.btn-danger`, and so on) by using them together with the base class `.btn` class.

If you want to define a new color, the suggestion is to create a new class and define the necessary pseudo-class. Let's assume we want a purple button defined by a class `.btn-purple`. Define a CSS for it:

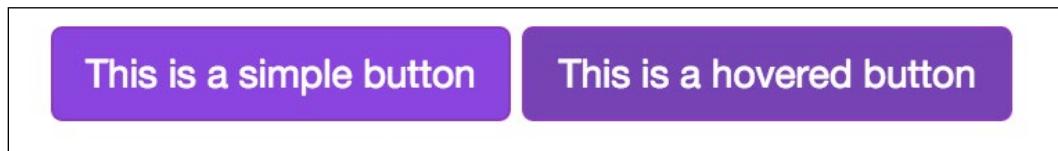
```
.btn-purple {  
    color: #fff;  
    background-color: #803BDB;  
    border-color: #822FBA;  
}
```

This is the base CSS. Now we must define all the pseudo-classes for the button:

```
.btn-purple:hover,
.btn-purple:focus,
.btn-purple:active,
.btn-purple.active {
    color: #ffffff;
    background-color: #6B39AD;
    border-color: #822FBA;
}
```

Now, for every interaction with the button (such as hovering over it), the button will have a background color a little darker. Not all same pseudo-classes can have the same style; you can customize it as per your choice.

The next screenshot represents our new button. What we did was replace the `.btn-default` for the class `.btn-purple`. The one on the left is `.btn-purple` and the one on the right is `.btn-purple:hover`:



Using button toggle

Bootstrap has a nice feature for button toggle. It is native from the framework and can be used in different ways. We will take a look at the single toggle button. For that, create a normal button in the sandbox page:

```
<button type="button" class="btn btn-primary" autocomplete="off">
    Single toggle
</button>
```

To make this button turn into a single toggle, we have to add the data attribute `data-toggle="button"` and the attribute `aria-pressed="true"`. This will turn the button into a toggle button. Now when you click on the button, Bootstrap will add a class `.active` to it, making it appear pressed. The code is as follows:

```
<button type="button" class="btn btn-default" data-toggle="button"
aria-pressed="false" autocomplete="off">
    Single toggle
</button>
```

The checkbox toggle buttons

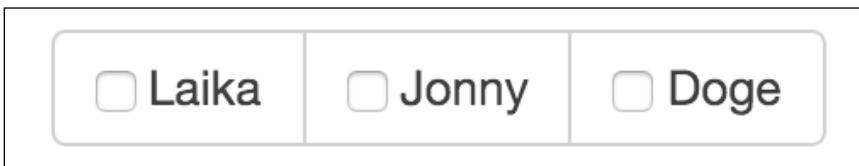
The toggle buttons can turn into buttons checkbox or buttons radio. At first, we need to remember the concept of button group. So let's create a simple .btn-group in the HTML:

```
<div class="btn-group">
  <button class="btn btn-default">
    Laika
  </button>
  <button class="btn btn-default">
    Jonny
  </button>
  <button class="btn btn-default">
    Doge
  </button>
</div>
```

The concept of using button groups is to create a `div` with the class `.btn-group` and insert a bunch of `button` elements inside it. However, we want a bunch of checkboxes, so let's substitute the `button` element for a `label` and `input` elements with type `checkbox`:

```
<div class="btn-group">
  <label class="btn btn-default">
    <input type="checkbox" autocomplete="off"> Laika
  </label>
  <label class="btn btn-default">
    <input type="checkbox" autocomplete="off"> Jonny
  </label>
  <label class="btn btn-default">
    <input type="checkbox" autocomplete="off"> Doge
  </label>
</div>
```

Refresh the page and you will see that the button list now has a checkbox input on each label, as shown in the following screenshot:

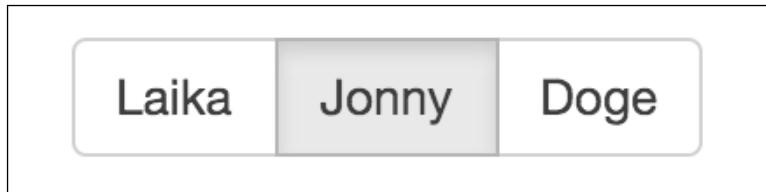


To change it to toggle and hide the checkboxes, we just need to simple add the data attribute `data-toggle="buttons"`.

There is an option to preselect a checkbox, just need to add the `.active` class to the label and add the attribute `checked="checked"` to the input:

```
<div class="btn-group" data-toggle="buttons">
  <label class="btn btn-default">
    <input type="checkbox" autocomplete="off"> Laika
  </label>
  <label class="btn btn-default active">
    <input type="checkbox" autocomplete="off" checked="checked">
Jonny
  </label>
  <label class="btn btn-default">
    <input type="checkbox" autocomplete="off"> Doge
  </label>
</div>
```

The next image shows the final output of the checkbox with the second checkbox selected on the page reload:



The button as a radio button

The other option for the toggle button is to become a radio button. The procedure is very similar to the checkbox. We just need to change the input from `type="checkbox"` to `type="radio"`:

```
<div class="btn-group" data-toggle="buttons">
  <label class="btn btn-default">
    <input type="radio" autocomplete="off"> Laika
  </label>
  <label class="btn btn-default active">
    <input type="radio" autocomplete="off" checked="checked"> Jonny
  </label>
  <label class="btn btn-default">
    <input type="radio" autocomplete="off"> Doge
  </label>
</div>
```

This will create a `.btn-group` formed by radio button, been just one selected at once.

Doing the JavaScript customization

Buttons can be customized using JavaScript as well. For instance, any toggle button can be toggled by calling:

```
$( 'button selector' ).button('toggle')
```

This will toggle the state of the button from active to not active.

Before Version 3.3.6, it was possible to change the text of a button via JavaScript by calling the button passing a string. First, you should define a state text. For instance, let's define a button with the attribute `data-statesample-text="What a sample"`:

```
<button type="button" class="btn btn-primary" autocomplete="off"
data-statesample-text="What a sample">
    Single toggle
</button>
```

Using JavaScript, you can change the text with the value or the data text by calling:

```
$( 'button' ).button('statesample');
```

Reset the text to original with the following function:

```
$( 'button' ).button('reset');
```

However, this feature is deprecated after Version 3.3.6 and will be removed in Version 4 of Bootstrap.

Working with plugin customization

Just like the customization for components, it is also possible to customize the behavior of the Bootstrap plugins.

To illustrate that, let's consider the Bootstrap Modal. This plugin is one of the most used among the others. The Modal is able to create a separated flow in your web page without changing the context.

Let's create an input and a button and make the button open the modal when clicked. What we are expecting here is when the user inputs the GitHub username at the input, we will get the info in the GitHub open API and show some basic info at the Modal. For this, create the following code in the sandbox page:

```
<!-- Button trigger modal -->
<input id="github-username" type="text" class="form-control"
placeholder="Type your github username here">
<button type="button" class="btn btn-success btn-lg btn-block"
data-toggle="modal" data-target="#githubModal">
    Launch demo modal
</button>

<!-- Modal -->
<div class="modal fade" id="githubModal" tabindex="-1"
role="dialog">
    <div class="modal-dialog" role="document">
        <div class="modal-content">
            <div class="modal-header">
                <button type="button" class="close" data-dismiss="modal"
aria-label="Close"><span aria-hidden="true">&times;</span></button>
                <h4 class="modal-title"></h4>
            </div>
            <div class="modal-body">
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-default" data-
dismiss="modal">Close</button>
                <button type="button" class="btn btn-success">Save
changes</button>
            </div>
        </div>
    </div>
</div>
```

Refresh the web page and you will see the input followed by a button. When you click on it, the Modal will show. The Modal is completely empty; to interact with that, we will play with some JavaScript.

In the code, let's use the Bootstrap event `show.bs.modal`, which will be triggered whenever a Modal is shown (like we discussed previously):

```
$( '#githubModal' ).on( 'show.bs.modal', function (e) {
    var $element = $(this),
        url = 'https://api.github.com/users/{username}';
}) ;
```

Inside the function, we defined two variables. The `$element` corresponds to the triggered element, in this case it is the modal `#githubModal`. The `url` is the endpoint for the GitHub API. We will replace the `{username}` parameter on the string based on the text passed at the input by doing that:

```
$( '#githubModal' ).on( 'show.bs.modal', function (e) {
    var $element = $(this),
        url = 'https://api.github.com/users/{username}';

    url = url.replace(/{username}/, $('#github-username').val());
}) ;
```

Then, we must make a request to the API to retrieve the user info. To do so, we must make a GET request to the API, which will return us a JSON.

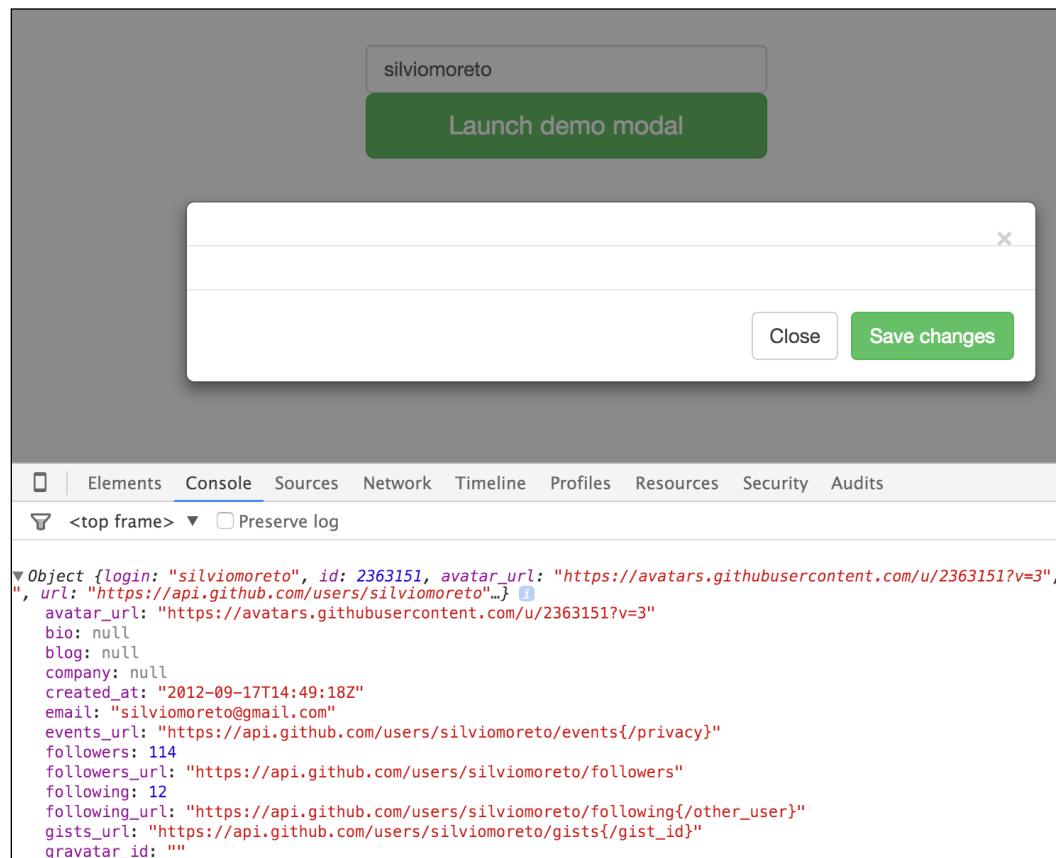
To make it clear, JSON is an open standard format to transmit data as a set of key-values. It is widely used to transfer data from web services and APIs, such as GitHub.

Moving on, to make the request to the server, we use the function `$.get` from jQuery. Pass a URL and a callback function with the JSON data object returned from the server:

```
$( '#githubModal' ).on( 'show.bs.modal', function (e) {
    var $element = $(this),
        url = 'https://api.github.com/users/{username}';

    $.get(url, function(data) {
        console.log(data);
    });
}) ;
```

If everything is working so far, refresh your web browser, type your username on the input, and click on the button. After the modal opens, check your console terminal and you must see the data from the request, as shown in the following screenshot:



Now, it would be good if we parse the data and displayed some information on the modal. For that, let's use the same principle for replace the `url` variable. Along with the variables, let's add other ones related to the template.

We want to create a template with two columns, the left one the user avatar image from the object and some basic info on the right. So, add the highlighted lines in your JavaScript:

```
$('#githubModal').on('show.bs.modal', function (e) {
    var $element = $(this),
        url = 'https://api.github.com/users/{username}',
        title = 'Hi, my name is {name}',
        content = '' +
            '<div class="row">' +
            '  ' +
            '  <p class="col-sm-9" id="bio">{bio}</p>' +
            '</div>',

        bio = '' +
            'At moment I have {publicRepo} public repos ' +
            'and {followers} followers.\n' +
            'I joined Github on {dateJoin}';

    $.get(url, function(data) {
        console.log(data);
    });
});
```

Here, we created three template variables that we will replace with the data from the get request. Inside the get function, let's replace the variables and create our final template.

The principle is the same as what we applied to the url, just replace the key, which is surrounded by curly brackets, with the value on data:

```
$('#githubModal').on('show.bs.modal', function (e) {
    var $element = $(this),
        url = 'https://api.github.com/users/{username}',
        title = 'Hi, my name is {name}',
        content = '' +
            '<div class="row">' +
            '  ' +
            '  <p class="col-sm-9" id="bio">{bio}</p>' +
            '</div>',

        bio = '' +
            'At moment I have {publicRepo} public repos ' +
            'and {followers} followers.\n' +
            'I joined Github on {dateJoin}';

    $.get(url, function(data) {
        title = 'Hi, my name is ' + data.name;
        bio = 'At moment I have ' + data.public_repos + ' public repos ' +
            'and ' + data.followers + ' followers.\n' +
            'I joined Github on ' + data.created_at;
        $element.html(content);
    });
});
```

```
bio = '' +
    'At moment I have {publicRepo} public repos ' +
    'and {followers} followers.\n' +
    'I joined Github on {dateJoin}';

url = url.replace("/{username}/", $('#github-username').val());

$.get(url, function(data) {
    title = title.replace("/{name}/", data.name);

    bio = bio.replace("/{publicRepo}/", data.public_repos)
        .replace("/{followers}/", data.followers)
        .replace("/{dateJoin}/", data.created_at.split('T')
[0]);

    content = content.replace("/{img}/", data.avatar_url)
        .replace("/{bio}/", bio);

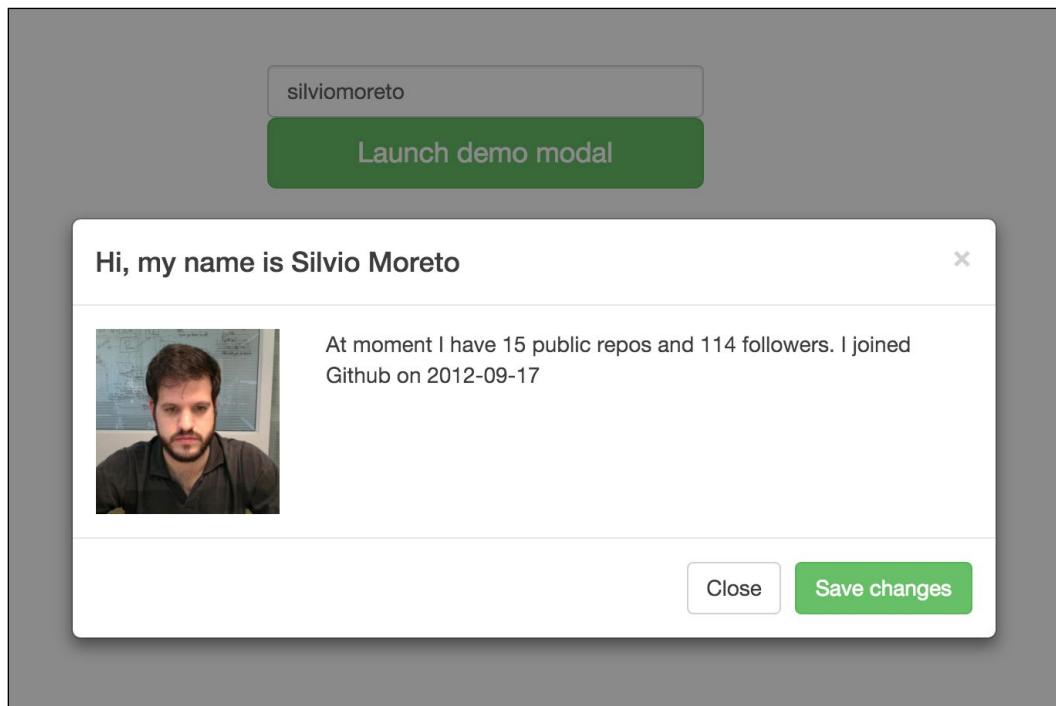
    $element.find('.modal-title').text(title);
    $element.find('.modal-body').html(content);
});

});
```

After all the replacements, we set the parsed template variables to the modal. We query the title to find the `.modal-title` and insert the text inside, while we insert the HTML for `.modal-body`.

The difference here is that we can pass an HTML or a simple text to jQuery. Take care when you pass an HTML to ensure that your HTML is not degenerated. That might cause issues for your client. So, pay attention when you want to set just a text, like for `.modal-title`, or a valid `html`, like for `.modal-body`.

On the browser, type your GitHub username on the input, press the button, and you should see a nice modal, such as the one in the next screenshot:



So, we saw how to interact more with the Bootstrap plugins while customizing it for our own tasty.

Remember that the Bootstrap events exist for every Bootstrap plugin. They are friendly and can be very handy while interacting with the plugins, like in this case, to execute some action when the Modal shows.

The additional Bootstrap plugins

Bootstrap has plugins for almost anything. However, there are some missing components and plugins that would be nice to have in our web pages, for example, a data picker, or a color picker, or a select component. Bootstrap does not incorporate these plugins into the framework because they are not that generic for any application, so you should add it if you need.

Knowing that, the Bootstrap developers provide a list of additional Bootstrap resources that can be found at <http://expو.getbootstrap.com/resources/>.

Creating our Bootstrap plugin

In the previous chapter, we discussed the Carousel Bootstrap plugin. Do you remember the HTML markup to use the plugin? It is a big markup as you can see from the following code:

```
<div id="carousel-notification" class="carousel slide" data-ride="carousel">

    <!-- Wrapper for slides -->
    <div class="carousel-inner" role="listbox">
        <div class="item active">
            
            <div class="carousel-caption">
                <h3>Doge said:</h3>
                <p>What are you doing? So scare. It's alright now.</p>
            </div>
        </div>
        <div class="item">
            
            <div class="carousel-caption">
                <h3>Crazy cat said:</h3>
                <p>I will never forgive you...</p>
            </div>
        </div>
        <div class="item">
            
            <div class="carousel-caption">
                <h3>Laika said:</h3>
                <p>Hey! How are you?</p>
            </div>
        </div>
    </div>

    <!-- Indicators -->
    <ol class="carousel-indicators">
        <li data-target="#carousel-notification" data-slide-to="0" class="active"></li>
        <li data-target="#carousel-notification" data-slide-to="1"></li>
        <li data-target="#carousel-notification" data-slide-to="2"></li>
    </ol>
```

```
<!-- Controls -->
<a class="left carousel-control" href="#carousel-notification"
role="button" data-slide="prev">
    <span class="glyphicon glyphicon-chevron-left" aria-
hidden="true"></span>
    <span class="sr-only">Previous</span>
</a>
<a class="right carousel-control" href="#carousel-notification"
role="button" data-slide="next">
    <span class="glyphicon glyphicon-chevron-right" aria-
hidden="true"></span>
    <span class="sr-only">Next</span>
</a>
</div>
```

There is a reason why the plugin has all these lines of code. With all of that, you are able to customize the plugin for your own use. However, it would be nice if we had a simple carousel with fewer lines of code. Can we do that?

The template that we are trying to create for new plugin will have only the HTML that will reflect the same action as the preceding code:

```
<div id="carousel-notification" class="bootstrap-carousel">
    
    
    
</div>
```

In the plugin, we will have only one `div` wrapping up everything. Inside that, we will have a sequence of `img` elements, each one containing the image source, the title via `data-title`, and the slide content via `data-content`.

Building a plugin from scratch is quite difficult, but we will be able to learn the concepts behind Bootstrap and master it when we finish the plugin.

Creating the plugin scaffold

First of all, let's define the directories and files that we are using. For the HTML, we will start a new one that will have the same base template that was used in all the other examples.

Inside the `imgs` directory, we will keep the pet images that we used in the previous chapter. In this chapter, we will not use any CSS, so do not mind that.

Create a file named `bootstrap-carousel.js` inside the `js` folder and import it in the HTML just below the `bootstrap.js` load (bottom of the page):

```
<script src="js/jquery-1.11.3.js"></script>
<script src="js/bootstrap.js"></script>
<script src="js/bootstrap-carousel.js"></script>
```

Let's create the plugin base. Inside the `bootstrap-carousel.js` file, add the following lines:

```
+function ($) {
  'use strict';

  // BOOTSTRAP CAROUSEL CLASS DEFINITION
  // =====
  var BootstrapCarousel = function (element, options) {
    this.$element = $(element);
    this.options = $.extend({}, BootstrapCarousel.DEFAULTS,
options);
  }

  BootstrapCarousel.VERSION = '1.0.0'
  BootstrapCarousel.DEFAULTS = {
  };

  BootstrapCarousel.prototype = {
  };

}(jQuery);
```

Here, we define a new function for jQuery. First, we define a class called `BootstrapCarousel` that will be our plugin. The function receives the element that will be applied the carousel and options that will be passed through data attributes or JavaScript initialization.



Why the plus symbol in the beginning of the function?

The plus (+) symbol forces to treat it as an expression so that any function after it should be called immediately. Instead of this symbol, we could use others unary operators to have the same effect (such as `!`, `~`, or `()`). Without the initial symbol, the function can be seen as the declaration of a function rather than an expression, which can create a syntax error.

The variable `options` are then extended from the `BootstrapCarousel.DEFAULT` options. So, if an option is not provided, a default value will be used.

Let's define the `VERSION` of the plugin, the `DEFAULT` values, and the `prototype` that contains all the properties and methods for the class. Inside `prototype`, we will create the plugin methods and classes, and this is where the core logic will be stored.

Before creating the Bootstrap carousel logic, we must finish some tasks for plugin initialization. After `prototype`, let's create our plugin initialization:

```
+function ($) {
  'use strict';

  // BOOTSTRAP CAROUSEL CLASS DEFINITION
  // =====
  var BootstrapCarousel = function (element, options) {
    this.$element = $(element);
    this.options = $.extend({}, BootstrapCarousel.DEFAULTS,
options);
  }

  BootstrapCarousel.VERSION = '1.0.0'
  BootstrapCarousel.DEFAULTS = {
  };

  BootstrapCarousel.prototype = {
  };

  // BOOTSTRAP CAROUSEL PLUGIN DEFINITION
  // =====
  function Plugin(option) {

    var args = arguments;
    [].shift.apply(args);

    return this.each(function () {
      var $this = $(this),
        data = $this.data('bootstrap-carousel'),
        options = $.extend({}, BootstrapCarousel.DEFAULTS,
$this.data(), typeof option == 'object' && option),
        value;
```

```

        if (!data) {
            $this.data('bootstrap-carousel', (data = new
BootstrapCarousel(this, options)));
        }

        if (typeof option == 'string') {
            if (data[option] instanceof Function) {
                value = data[option].apply(data, args);
            } else {
                value = data.options[option];
            }
        }
    })
}

})(jQuery);

```

The class `Plugin` will receive the option called and arguments for the element and call it. Do not worry about this part. This is quite a common plugin initialization that is replicated over almost all plugins.

To end the plugin initialization, add the following highlighted code after the `Plugin` class:

```

+function ($) {
    'use strict';

    // BOOTSTRAP CAROUSEL CLASS DEFINITION
    // =====
    var BootstrapCarousel = function (element, options) {
        this.$element = $(element);
        this.options = $.extend({}, BootstrapCarousel.DEFAULTS,
options);
    }

    BootstrapCarousel.VERSION = '1.0.0'
    BootstrapCarousel.DEFAULTS = {
    };

    BootstrapCarousel.prototype = {
    };
}

```

```
// BOOTSTRAP CAROUSEL PLUGIN DEFINITION
// =====
function Plugin(option) {
    ... // the plugin definition
}

var old = $.fn.bCarousel;
$.fn.bCarousel = Plugin;
$.fn.bCarousel.Constructor = BootstrapCarousel;

// BOOTSTRAP CAROUSEL NO CONFLICT
// =====
$.fn.bCarousel.noConflict = function () {
    $.fn.bCarousel = old;
    return this;
}

// BOOTSTRAP CAROUSEL CLASS LOAD
// =====
$(window).on('load', function () {
    $('.bootstrap-carousel').each(function () {
        var $carousel = $(this);
        Plugin.call($carousel, $carousel.data());
    })
})

})(jQuery);
```

First, we associate the plugin with jQuery by in the line `$.fn.bCarousel = Plugin;`. Then, set that the constructor for the class initialization will be called for `$.fn.bCarousel.Constructor = BootstrapCarousel;`. Here, we named our plugin `bCarousel`, so we will can the plugin via JavaScript:

```
($('some element selected').bCarousel());
```

Then, we add the plugin again for conflict cases where you have more than one plugin with the same name.

In the last part of code, we initialize the plugin via data class. So, for each element identified by the class `.bootstrap-carousel`, the plugin will be initialized passing the data attributes related to it automatically.

Defining the plugin methods

Now that we have our plugin well declared, we must fill the logic for it. We will create methods inside the `prototype` to create this behavior. We will only show this portion of the plugin code here.

The first method that we will create is `init()`. We will call it later to start the plugin. Before that, we have a few steps:

- Initial verifications
- Assigning the plugin elements and prerequisites
- Loading the original Bootstrap template
- Starting the Bootstrap plugin

The initialize method and plugin verifications

Actually, we have only one requirement from the Bootstrap original carousel plugin: the outmost `div` must have an `id`. Let's create the `init` function while making this assertion:

```
BootstrapCarousel.prototype = {
  init: function () {
    if (!this.$element.attr('id')) {
      throw 'You must provide an id for the Bootstrap Carousel element.';
    }

    this.$element.addClass('slide carousel');
  }
};
```

Therefore, we check if the element has the attribute `id` using `this.$element.attr('id')`. If not, we throw an error to the console and the developer will properly fix this issue. Note that we can access the plugin element using `this.$element` because we made this assignment at the start of the plugin.

In the last line of the function, we added some classes needed for the Bootstrap Carousel, in case we do not have it in the `$element` such as `.slide` and `.carousel`.

Adding the Bootstrap template

To load the Bootstrap Carousel template, let's create another function called `load` inside the `init` method to start it:

```
BootstrapCarousel.prototype = {
    init: function () {
        if (!this.$element.attr('id')) {
            throw 'You must provide an id for the Bootstrap Carousel
element.';
        }

        this.$slides = this.$element.find('> img');
        this.$element.addClass('slide carousel');
        this.load();
    }

    load: function() {
    },
};
```

First, we must remove any Carousel elements that could be already present inside our `$element`. The elements that we must remove are the ones with the `.carousel-inner`, `.carousel-indicators`, and `.carousel-control` classes. Also, we have to load and hide the slide images in the variable `this.$slides`:

```
load: function() {
    // removing Carousel elements
    this.$element.find('.carousel-inner, .carousel-indicators,
.carousel-control').remove();

    // loading and hiding the slide images
    this.$slides = this.$element.find('> img');
    this.$slides.hide();
},
```

Next, we must make sure that there are not any other associations of Bootstrap Carousel in our plugin element. Append the following lines in the function:

```
this.$element.carousel('pause');
this.$element.removeData('bs.carousel');
```

First, we will pause the Carousel to stop any interaction and after use the function `removeData` in the `bs.carousel`, which is the name of the Carousel plugin.

To continue, we must load the Bootstrap Carousel template. Inside the class prototype, we have to create a variable to hold the original template. The variable will have the following format:

```
template: {
  slide: '...',
  carouselInner: '...',
  carouselItem: '...',
  carouselIndicator: '...',
  carouselIndicatorItem: '...',
  carouselControls: '...',
},
}
```

We are not going to place the full code of each template because it is quite extensive, and it would be better for you to check the full code attached with the book and see each template. Although there are no secrets in the templates, they are just a big string with some marked parts that we will replace. The marked parts are defined as a string around curly brackets, for example, `{keyName}`. When creating the template, we just need to replace these parts of the string by calling `.replace(/{keyName}/, 'value')`.

Each key inside the template correspond to a certain part of the template. Let's explain each one:

- `slide`: This is the slide template of the new plugin and it is used to add slides via JavaScript
- `carouselInner`: This is the element inside the carousel that is parent for the items
- `carouselItem`: This is the item that contains the image and the caption of a slide
- `carouselIndicator`: This is the set of bullets at the bottom of the carousel
- `carouselIndicatorItem`: This represents each bullet of the indicator
- `carouselControls`: This is the controls to switch between left and right the carousel slides

At the end of the `load` method, add two more lines:

```
load: function() {
  this.$element.find('.carousel-inner, .carousel-indicators,
.carousel-control').remove();
  this.$slides = this.$element.find('> img');
  this.$slides.hide();
```

```
this.$element.carousel('pause');
this.$element.removeData('bs.carousel');

this.$element.append(this.createCarousel());
this.initPlugin();
},
```

So, we will append in the `this.$element` the template generated in the function `createCarousel`. After that, we just need to initialize the Bootstrap original Carousel plugin.

Creating the original template

The original template will be created in the function `createCarousel`. It is composed of two steps. The steps are as follows:

- We create the slide deck for the `.carousel-inner` element
- Then, we create the indicator and the controls, if needed

Thus, the `createCarousel` method is composed of the call of these three functions that will append the string template to a variable:

```
createCarousel: function() {
    var template = '';

    // create slides
    template += this.createSlideDeck();

    // create indicators
    if(this.options.indicators) {
        template += this.createIndicators();
    }

    // create controls
    if(this.options.controls) {
        template += this.createControls();
    }

    return template
},
```

Note that for the indicator and the controls we made, check before creating the template. We performed a check in the `this.options` variable to see if the developer passed the argument to add these components or not.

So, we are defining the first two variables of our plugin. They can be passed through data attributes in the element, like `data-indicators` and `data-controls`. It defines whether the template will have these elements or not.

The slide deck

The slide deck will be created by the iterating of each `this.$slide` and loading the image source, the `data-title` and the `data-content` in this case. Also, for the first item, we must apply the class `.active`. The code is as follows:

```
createSlideDeck: function() {
    var slideTemplate = '',
        slide;

    for (var i = 0; i < this.$slides.length; i++) {
        slide = this.$slides.get(i);

        slideTemplate += this.createSlide(
            i == 0 ? 'active' : '',
            slide.src,
            slide.dataset.title,
            slide.dataset.content
        );
    }

    return this.template.carouselInner.replace(/{{innerContent}}/,
        slideTemplate);
},
```

In each iteration, we are calling another function named `createSlide`, where we are passing, if the slide is active, the image source, the item title, and the item content. This function will then replace the template using these arguments:

```
createSlide: function(active, itemImg, itemTitle, itemContent) {
    return this.template.carouselItem
        .replace('{{activeClass}}', active)
        .replace('{{itemImg}}', itemImg)
        .replace('{{itemTitle}}', itemTitle ||
this.options.defaultTitle)
        .replace('{{itemContent}}', itemContent ||
this.options.defaultContent);
}
```

We performed a check for the title and the content. If there is no title or content provided, a default value will be assigned from `this.options`. Just like the indicators and controls, these options can be passed through data attributes such as `data-default-title` and `data-default-content` in the plugin HTML element.



Do not forget that these options can be also provided in the plugin initialization through JavaScript by calling `.bCarousel({ defaultTitle: 'default title' })`.

The carousel indicators

The function `createIndicators` is used to create the indicators. In this function, we will perform the same method of the one to create the slide deck. We will create each bullet and then wrap it in the list of `.carousel-indicators`:

```
createIndicators: function() {
    var indicatorTemplate = '',
        slide,
        elementId = this.$element.attr('id');

    for (var i = 0; i < this.$slides.length; i++) {
        slide = this.$slides.get(i);

        indicatorTemplate += this.template.carouselIndicatorItem
            .replace("/{elementId}/", elementId)
            .replace("/{slideNumber}/", i)
            .replace("/{activeClass}/", i == 0 ? 'class="active"' : '');
    }

    return this.template.carouselIndicator.replace("/{indicators}/",
        indicatorTemplate);
},
```

The only trick here is that each bullet must be enumerated and have a reference to the parent element id. Thus, we made the replacements for each `this.$slides` and returned the indicator template.



Why are replacing the key and surrounding with slashes?

Surrounding with slashes on JavaScript performs a regex search on the pattern provided. This can be useful for custom replaces and specific searches.

The carousel controls

The controls create the arrows to switch slides from left to right. They follow the same methodology as the other templates. Just get a template and replace the keys. This method must be implemented like this:

```
createControls: function() {
    var elementId = this.$element.attr('id');

    return this.template.carouselControls
        .replace(/{{elementId}}/g, elementId)
        .replace(/{{previousIcon}}/, this.options.previousIcon)
        .replace(/{{previousText}}/, this.options.previousText)
        .replace(/{{nextIcon}}/, this.options.nextIcon)
        .replace(/{{nextText}}/, this.options.nextText);
},
```

Note that in the first replacement for the {{elementId}}, our regex has an append g. The g on the regex is used to replace all occurrences of the following pattern. If we do not use g, JavaScript will only replace the first attempt. In this template we have two {{elementId}} keys, using which we replace both at once.

We also have some options passed through plugin initialization for the previous and next icon and the text corresponding to that.

Initializing the original plugin

After creating the original template, we must start the original Carousel plugin. We defined a function called initPlugin with the following implementation:

```
initPlugin: function() {
    this.$element.carousel({
        interval: this.options.interval,
        pause: this.options.pause,
        wrap: this.options.wrap,
        keyboard: this.options.keyboard
    });
},
```

It just starts the plugin by calling `this.$element.carousel` while passing the carousel options on start. The options are loaded just like the others that we presented before. As shown, the options are loaded in the plugin class definition in the following line:

```
this.options = $.extend({}, BootstrapCarousel.DEFAULTS, options);
```

If any option is passed, it will override the default options present in `BootstrapCarousel.DEFAULTS`. We must create like this:

```
BootstrapCarousel.DEFAULTS = {  
    indicators: true,  
    controls: true,  
    defaultTitle: '',  
    defaultContent: '',  
    nextIcon: 'glyphicon glyphicon-chevron-right',  
    nextText: 'Next',  
    previousIcon: 'glyphicon glyphicon-chevron-left',  
    previousText: 'Previous',  
    interval: 5000,  
    pause: 'hover',  
    wrap: true,  
    keyboard: true,  
};
```

Making the plugin alive

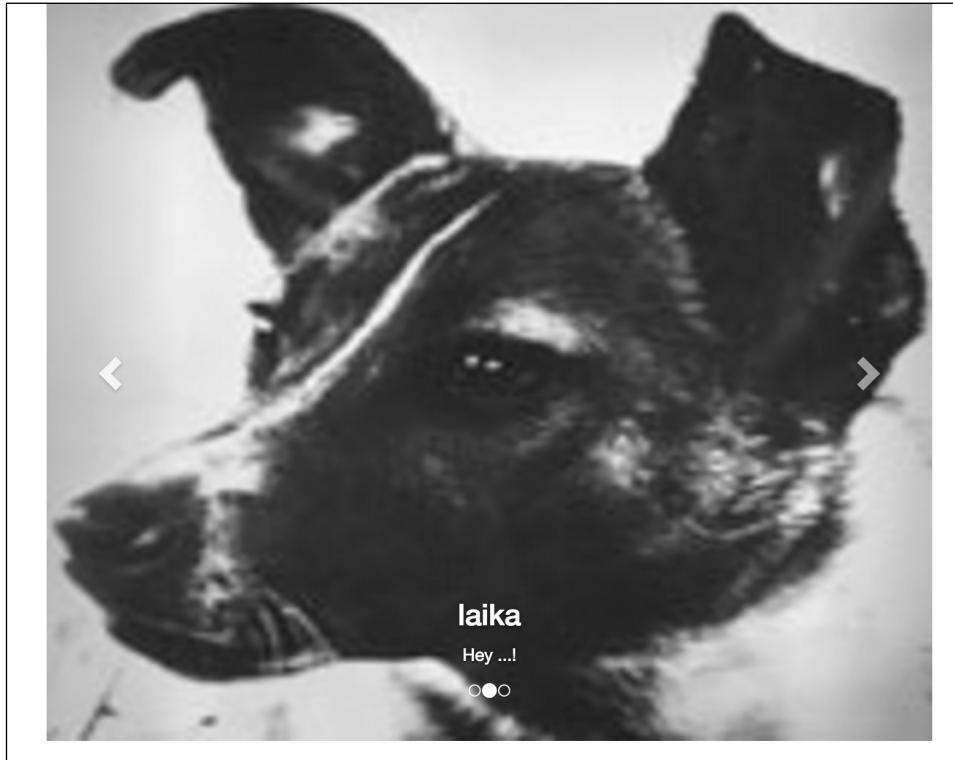
We are one step away from loading the plugin. To do so, create the following code in the HTML:

```
<div id="carousel-notification" class="bootstrap-carousel" data-  
indicators="true" data-controls="true">  
      
      
      
</div>
```

In our plugin JavaScript, we have to ignite the prototype by calling the `init` function like this:

```
var BootstrapCarousel = function (element, options) {  
    this.$element = $(element);  
    this.options = $.extend({}, BootstrapCarousel.DEFAULTS, options);  
  
    this.init();  
}
```

Hooray! Open the HTML file in our browser and see the plugin in action, as shown in the next screenshot. In the DOM, you can now perfectly mime the Bootstrap Carousel plugin, reducing the declaration in almost 35 lines of code:



Creating additional plugin methods

We are almost finishing our plugin. Now, it's time to add some methods to be called in the plugin, just like you can call `.carousel('pause')` on Bootstrap Carousel for instance.

When we were creating the plugin base, we created a class `Plugin`, which is the definition of the plugin. This part of the code is pretty common across the plugins and it is used on every native Bootstrap plugin:

```
function Plugin(option) {  
  
    var args = arguments;  
    [].shift.apply(args);  
  
    return this.each(function () {  
        var $this = $(this),  
            data = $this.data('bootstrap-carousel'),  
            options = $.extend({}, BootstrapCarousel.DEFAULTS, $this.  
data(), typeof option == 'object' && option),  
            value;  
  
        if (!data) {  
            $this.data('bootstrap-carousel', (data = new  
BootstrapCarousel(this, options)));  
        }  
  
        if (typeof option == 'string') {  
            if (data[option] instanceof Function) {  
                value = data[option].apply(data, args);  
            } else {  
                value = data.options[option];  
            }  
        }  
    })  
}
```

If you take a look at the highlighted lines of code, here we check the option variable that is passed. If it is a string, we apply the function, calling the option function on the plugin.

After that, we need to expose the function of the BootstrapCarousel class definition. So let's add two options, one to reload the plugin and another to add a slide to the carousel:

```
var BootstrapCarousel = function (element, options) {
    this.$element = $(element);
    this.options = $.extend({}, BootstrapCarousel.DEFAULTS, options);

    // Expose public methods
    this.addSlide = BootstrapCarousel.prototype.addSlide;
    this.reload = BootstrapCarousel.prototype.load;

    this.init();
}
```

The highlighted lines represent the exposed methods. Now we need to implement them on the prototype.

Although one of the methods has already been implemented, the `BootstrapCarousel.prototype.load` when exposing it we renamed the expose from load to reload. Calling this method will erase all the Bootstrap Carousel original plugin, create the template again based on the images passed through our plugin, and generate the plugin again.

We need to implement the method `BootstrapCarousel.prototype.addSlide`. So, inside `Bootstrap.prototype`, create the following function:

```
addSlide: function(itemImg, itemTitle, itemContent) {
    var newSlide = this.template.slide
        .replace("/{itemImg}/", itemImg)
        .replace("/{itemTitle}/", itemTitle)
        .replace("/{itemContent}/", itemContent);
    this.$element.append(newSlide);
    this.load();
},
```

This function will receive `itemImg`, which is the source of an image; `itemTitle`, for the slide title caption; and `itemContent` for the paragraph on the caption as well.

To create a new slide, we first use the template for a new one that can be found in the template variable `this.template.slide`:

```
template: {
    slide: '',
    ... // others template variable
}
```

Like creating the slide deck, indicators, and controls, we set a multiple keys identified around curly brackets and do a replace of them in the function.

After the replacements, the new slide is appended to `this.$element`, which also contains the others slides. Finally, we need to call the `load` function, which will do all the hard work to assign variables, hide elements, and start the original plugin.

Then, when you want to add a slide to the plugin, you just need to call:

```
$('.bootstrap-carousel').bCarousel('addSlide', 'imgs/jon.png',
'New title image', 'This is awesome!');
```

With this plugin function, we are done! See, it is not too difficult to create a new plugin. We can now start incrementing it with more options for automation and customization.

Summary

In my opinion, this last chapter was awesome! We saw more about Bootstrap customization in terms of both components style and plugin interaction. Bootstrap is a great framework, but what makes it great is the extensibility potential that it has. It matches the perfect world where premade components and customization live in symbiosis.

To finish the book with a flourish, we developed a new Bootstrap plugin, the wrapper for Bootstrap Carousel. The plugin contemplates almost every pattern for the Bootstrap plugin, and it has been very helpful in creating a simple carousel with minimal verbosity.

The plugin is available on GitHub at github.com/silviomoreto/bootstrap-carousel. Take a look at it and create a pull-request! There are a bunch of improvements and new features that could be added to the plugin—perhaps a method to remove slides?

Also, the goal of creating a plugin is to make you able to create a new one in the future or understand a Bootstrap plugin if you need to adjust some part of it. I think you can now see the plugin's code with more familiarity and improve them.

I would like to congratulate you for reaching the end of the book. Understanding a complete framework such as Bootstrap is not a simple task and it is completed by just a small group of developers in the world. Be proud of your achievement.

The understanding of the plugin goes from the basic usage of the scaffolding from the creation of a plugin to the components, elements, and more. All of that was achieved using very useful examples that will be useful some day in your work.

The cherry on top of the pie is that you also learned about Bootstrap 4, which was released recently. This means you are one of the few people who are completely ready to use the new version of the Bootstrap framework.

I hope you liked the journey through the world of Bootstrap and were able to learn a lot from this book. Now it is your turn! You must go and nail every frontend Bootstrap task that you face. I believe that with all the knowledge acquired from the examples covered in this book, you are more than ready to be a true Bootstrap master.

Module 2: Learning Bootstrap 4 - Second Edition

Chapter 1: Introducing Bootstrap 4

7

Introducing Bootstrap	7
Bootstrap 4 advantages	8
Improved grid system and flexbox	8
Card component	8
Rebooting normalize.css	9
Internet Explorer 8 support dropped	9
Other updates	9
Implementing framework files	9
Inserting the JavaScript files	10
The starter template	10
HTML5 DOCTYPE	11
Structuring the responsive meta tag	12
Normalizing and Rebooting	12
Taking the starter template further	12
Using a static site generator	12
Converting the base template to a generator	13
Installing Harp.js	13
Adding Sass in Harp	14
Setting up the project	14
Inserting the CSS	14
Inserting the JavaScript	14
Other directories	14
Setting up the layout	15
Compiling your project	16
Previewing your project	16
Deploying your project	16
Installing Surge	17
Using Surge to deploy your project	17
Summary	17

Chapter 2: Using Bootstrap Build Tools

18

Different types of tools	18
Installing Node.js	19
Updating npm	19
Installing Grunt	21

Download the Bootstrap source files	22
Installing Ruby	22
Installing the Bundler gem	24
Running the documentation	25
Setting up the static site generator	25
Why use Harp.js	26
Installing Harp.js	26
Setting up the blog project	27
css	27
fonts	27
img	27
js	28
partial	28
EJS files	28
Setting up the JSON files	28
Creating the data JSON file	29
Setting up the layout	30
Setting up the header	31
Setting up the footer	33
Creating our first page template	34
Compiling your project	34
Running your project	35
Viewing your project	35
A note about Sass	37
Summary	37
Chapter 3: Jumping into Flexbox	38
Flexbox basics and terminology	38
Ordering your Flexbox	39
Stretching your child sections to fit the parent container	41
Changing the direction of the boxes	41
Wrapping your Flexbox	43
Creating equal-height columns	45
Setting up the Bootstrap Flexbox layout grid	51
Updating the Sass variable	51
Setting up a Flexbox project	52
Adding a custom theme	53
Creating a basic three-column grid	54
Creating full-width layouts	56
Designing a single blog post	57

Summary	60
Chapter 4: Working with Layouts	61
Working with containers	61
Creating a layout without a container	63
Using multiple containers on a single page	63
Inserting rows into your layout	64
Adding columns to your layout	65
Extra small	65
Small	65
Medium	65
Large	66
Extra large	66
Choosing a column class	66
Creating a simple three-column layout	66
Mixing column classes for different devices	68
What if I want to offset a column?	69
Coding the blog home page	70
Writing the index.ejs template	70
Using spacing CSS classes	71
Testing out the blog home page layout	72
Adding some content	73
What about mobile devices?	75
Using responsive utility classes	77
Coding the additional blog project page grids	77
Updating _data.json for our new pages	77
Creating the new page templates	78
Coding the contact page template	78
Adding the contact page body	79
Coding the blog post template	82
Adding the blog post feature	82
Adding the blog post body	83
Converting the mailing list section to a partial	84
Summary	87
Chapter 5: Working with Content	88
Reboot defaults and basics	88
Headings and paragraphs	89
Lists	89
Preformatted text	89
Tables	89
Forms	89

Learning to use typography	90
Using display headings	90
Customizing headings	91
Using the lead class	91
Working with lists	92
Coding an unstyled list	92
Creating inline lists	94
Using description lists	94
How to style images	95
Making images responsive	95
Using image shapes	96
Aligning images with CSS	96
Coding tables	98
Setting up the basic table	98
Inversing a table	99
Inversing the table header	100
Adding striped rows	101
Adding borders to a table	101
Adding a hover state to rows	102
Color-coating table rows	103
Making tables responsive	104
Summary	104
Chapter 6: Playing with Components	105
Using the button component	105
Basic button examples	105
Creating outlined buttons	106
Checkbox and radio buttons	107
Creating a radio button group	108
Using button groups	110
Creating vertical button groups	110
Coding a button dropdown	111
Creating a pop-up menu	112
Creating different size drop-down buttons	113
Coding forms in Bootstrap 4	114
Setting up a form	114
Adding a select dropdown	115
Inserting a textarea tag into your form	117
Adding a file input form field	117
Inserting radio buttons and checkboxes to a form	118
Adding a form to the blog contact page	120
Updating your project	120

Additional form fields	122
Creating an inline form	
Hiding the labels in an inline form	123
Adding inline checkboxes and radio buttons	123
Changing the size of inputs	124
Controlling the width of form fields	125
Adding validation to inputs	126
Using the Jumbotron component	128
Adding the Label component	130
Using the Alerts component	131
Adding a dismiss button to alerts	132
Using Cards for layout	133
Moving the Card title	135
Changing text alignment in cards	136
Adding a header to a Card	137
Inverting the color scheme of a Card	139
Adding a location card to the Contact page	142
Updating the Blog index page	144
Adding the sidebar	147
Setting up the Blog post page	150
How to use the Navs component	153
Creating tabs with the Nav component	154
Creating a pill navigation	155
Using the Bootstrap Navbar component	156
Changing the color of the Navbar	157
Making the Navbar responsive	158
Adding Breadcrumbs to a page	160
Adding Breadcrumbs to the Blog post page	160
Using the Pagination component	161
Adding the Pager to the Blog post template	162
How to use the List Group component	162
Summary	164
Chapter 7: Extending Bootstrap with JavaScript Plugins	165
Coding a Modal dialog	165
Coding the Modal dialog	166
Coding Tooltips	168
Updating the project layout	168
How to use Tooltips	168
How to position Tooltips	170

Adding Tooltips to buttons	171
Updating the layout for buttons	171
Avoiding collisions with our components	172
Using Popover components	173
Updating the JavaScript	173
Positioning Popover components	174
Adding a Popover to a button	175
Adding our Popover button in JavaScript	175
Using the Collapse component	176
Coding the collapsable content container	176
Coding an Accordion with the Collapse component	177
Coding a Bootstrap Carousel	180
Adding the Carousel bullet navigation	181
Including Carousel slides	182
Adding Carousel arrow navigation	183
Summary	184
Chapter 8: Throwing in Some Sass	185
Learning the basics of Sass	185
Using Sass in the blog project	186
Updating the blog project	186
Using variables	187
Using the variables in CSS	188
Using other variables as variable values	189
Importing partials in Sass	190
Using mixins	192
How to use operators	193
Creating a collection of variables	194
Importing the variables to your custom style sheet	194
Adding a color palette	195
Adding some background colors	195
Setting up variables for typography	196
Coding the text color variables	197
Coding variables for links	198
Setting up border variables	198
Adding variables for margin and padding	199
Adding mixins to the variables file	199
Coding a border-radius mixin	200
Customizing components	202
Customizing the button component	202

Extending the button component to use our color palette	203
Writing a theme	
Common components that need to be customized	206
Theming the drop-down component	207
Customizing the alerts component	209
Customizing the typography component	211
Summary	212
Chapter 9: Migrating from Version 3	213
Browser support	213
Big changes in version 4	213
Switching to Sass	213
Updating your variables	214
Updating @import statements	214
Updating mixins	215
Additional global changes	216
Using REM units	216
Other font updates	217
New grid size	217
Migrating components	217
Migrating to the Cards component	218
Using icon fonts	218
Migrating JavaScript	218
Miscellaneous migration changes	219
Migrating typography	219
Migrating images	219
Migrating tables	219
Migrating forms	220
Migrating buttons	220
Summary	220
Index	221

Module 2

Learning Bootstrap 4

Second Edition

*Unearth the potential of Bootstrap 4 to create highly responsive and beautiful websites
using modern web techniques*

1

Introducing Bootstrap 4

Bootstrap is the most popular **HTML, CSS**, and JavaScript framework on the planet. Whether you are new to web development or an experienced master, Bootstrap is a powerful tool for whatever type of web application you are building. With the release of version 4, Bootstrap is more relevant than ever and brings a complete set of components that are easy to learn to use. In this book, I'll jump right into using Bootstrap, what's new in version 4, and strategies you can use to get the most out of the framework. In my opinion, the best way to learn to code is through real-world examples. As we progress through the book, we'll build a blog and portfolio website so that you will have a fully functional template once you're done. In this chapter, I'll cover the following topics:

- Why should you use Bootstrap?
- What's new in Bootstrap 4?
- The basic files and template required to start a project

Introducing Bootstrap

There are several reasons to use Bootstrap but let me boil it down to a few of the key reasons I recommend it. If you're like me, you're constantly starting new web projects. One of the most frustrating parts of getting a project off the ground is to reinvent the base HTML, CSS, and JavaScript for each project. It makes much more sense to reuse the same base code and then build on top of it. Some developers may prefer to write their own framework, and in some cases this may make sense. However, with most projects, I've found that it is easier to just use an existing framework. On top of the components that Bootstrap provides out-of-the-box, there are hundreds of other third-party components you can integrate it with, with a large community of other developers to help you.

Bootstrap is also a powerful prototyping tool in the start-up world. Often, you will want to vet an idea without investing tons of time into it. Bootstrap allows you to quickly build a prototype to prove out your idea without a large time commitment to build out a frontend that you might not end up using. Even better, if you're working in a team of developers, it is very likely everyone will be familiar with the framework. This will allow for code consistency from day one. No arguing over how to name the selectors or the best way to structure a CSS file. Most of the configuration is already set up for you and you can get on with creating your project faster.

Bootstrap 4 advantages

With the release of Bootstrap 4, there are a number of key updates to the framework. One of the biggest changes is the move from **Less**, which is a CSS preprocessor, to **Sass**. When Bootstrap first started out, there was no clear favorite when it came to preprocessors. Over the last couple of years, Sass has gained a bit of an edge, so this switch should come as no surprise. If you haven't used Sass before, don't worry; it is similar to Less and really easy to learn. In later chapters, I will cover Sass in greater depth.

Improved grid system and flexbox

Another big new feature in version 4 is the improved grid system and the inclusion of flexbox. For the regular grid, another grid level has been added to better target mobile devices, and media queries have been reworked too. Flexbox is the grid of the future and it's really exciting that it's been included. By default, the regular grid will work out-of-the-box but you can switch to the flexbox grid by switching a simple Sass variable to take advantage of this new layout component.

Card component

Bootstrap 4 sees the deprecation of components such as wells, thumbnails, and panels, and the introduction of the new card component. This is a good thing for a couple of reasons. First of all, it removes a few components that were similar and replaces them with a single card component. This makes the framework a little lighter and easier to learn for the new user. The card component has also seen an increase in popularity lately, so it makes sense to include it here. All one has to do is to look at the popularity of Google's **Material Design** to see how cards are a great component to use in a web application.

Rebooting normalize.css

One change that you might not notice immediately but is great nonetheless is the improvements to the built-in CSS reset. Bootstrap has taken `normalize.css` and extended it with a new module called **Reboot**. Reboot improves on Normalize and tightens up the default browser styling that needs to be reset for all web-based projects.

Internet Explorer 8 support dropped

I couldn't be happier to see that Bootstrap has dropped support for Internet Explorer 8 (IE8). The time has come to leave this browser in the past! If you need IE8 support, the recommendation is to continue using Bootstrap 3.

Other updates

All of the JavaScript plugins that come with Bootstrap have been rewritten in ES6, which allows for the use of the latest JavaScript functionality. The tooltip and popover components have been extended to use the Tether library. This is just scratching the surface, as there are a ton of other minor updates that have been built into the framework.

Implementing framework files

Before we get into building the basic template for a Bootstrap project, we should review the files that we need to include to make the framework run properly. At the very minimum, we require one CSS file and two JavaScript files. These files can either be served from the Bootstrap **Content Delivery Network (CDN)** or downloaded and included directly in our project. If you are using the CDN, simply include this line of code in the head of your file:

```
<link rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-alpha.2/css/bootstrap.min.css" integrity="sha384-yTfxAZXuh4HwSYylfB+J125MxIs6mR5FOHamPBG064zB+AFeWH94NdvaCBm8qnd"
      crossorigin="anonymous">
```

If you would like to include the CSS file yourself, go to <http://getbootstrap.com/> and download the framework. Extract the resultant ZIP file and locate the /css directory. Within this directory will be a number of CSS files. The only one you need to worry about is bootstrap.min.css. Locate that file and copy it to the /css directory of your own project. Once there, link it into the head of your document, which will look something like this:

```
<link rel="stylesheet" href="/path/to/your/file/bootstrap.min.css">
```

Inserting the JavaScript files

As I mentioned earlier, we need to include two JavaScript files to implement the framework properly. The files are the **jQuery** and **Bootstrap JavaScript** framework files. As with the CSS file, you can either do this through the use of a CDN or download and insert the files manually. The JavaScript files should be inserted at the bottom of your page right before the closing </body> tag. If you choose to use the CDN, insert the following lines of code:

```
<script  
src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js"></sc  
ript>  
<script  
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-alpha.2/js/bootstrap.m  
in.js" integrity="sha384-  
vZ2WRJMwsjRMW/8U7i6PWi6AlO1L79snBrmgidpgIWJ82z8eA5lenwvx  
bMV1PAh7"  
crossorigin="anonymous"></script>
```

If you prefer to insert the files yourself, go back to the Bootstrap package you downloaded earlier and locate the /js directory. There will be a few files here but the one you want is bootstrap.min.js. You'll need to also head to <http://jquery.com> to download the jQuery framework file. Once you've done that, drop both files into the /js directory for your own project. Next, enter the following lines of code at the bottom of your page template. Make sure jQuery is loaded before bootstrap.min.js. This is critical; if you load them in the opposite order, the framework won't work properly:

```
<script src="/path/to/your/files/jquery.min.js"></script>  
<script src="/path/to/your/files/bootstrap.min.js"></script>
```

That concludes the explanation of the key Bootstrap framework files you need to include to get your project started. The next step will be to set up the basic starter template so you can begin coding your project.

The starter template

The basic starter template is the bare bones of what you'll need to get a page going using Bootstrap. Let's start by reviewing the code for the entire template and then I'll break down each critical part:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- Required meta tags always come first -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,
shrink-to-fit=no">
    <meta http-equiv="x-ua-compatible" content="ie=edge">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-alpha.2/css/bootstrap
.min.css" integrity="sha384-
y3tfxAZXuh4HwSYylfB+J125MxIs6mR5FOHamPBG064zB+AFeWH94NdvaCBm8qnd"
crossorigin="anonymous">
  </head>
  <body>
    <h1>Hello, world!</h1>

    <!-- jQuery first, then Bootstrap JS. -->
    <script
      src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js"></sc
ript>
    <script
      src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-alpha.2/js/bootstrap.m
in.js" integrity="sha384-
vZ2WRJMwsjRMW/8U7i6PWi6AlO1L79snBrmgidpgIWJ82z8eA5lenwvx
bMV1PAh7"
crossorigin="anonymous"></script>
  </body>
</html>
```

HTML5 DOCTYPE

Like most projects nowadays, Bootstrap uses the HTML5 DOCTYPE for its template. That is represented by the following line of code:

```
<!DOCTYPE html>
```

Avoid using other DOCTYPES such as **XHTML** strict or transitional or unexpected issues will arise with your components and layouts.

Structuring the responsive meta tag

Bootstrap is a mobile-first framework so the following meta tag needs to be included to allow for responsive web design. To make sure your project renders properly on all types of devices, you must include this meta tag in the `<head>` of your project:

```
<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
```

If you're interested in learning more about how responsive web design works in Bootstrap, you should check out the documentation at: <http://v4-alpha.getbootstrap.com/layout/responsive-utilities/>.

That brings to a close the most important parts of the template that you need to be aware of. The remainder of the code in the starter template should be straightforward and easy to understand.

Normalizing and Rebooting

As I mentioned earlier, Bootstrap uses `normalize.css` as the base CSS reset. With the addition of the Reboot reset, Bootstrap extends Normalize and allows for styling to only be done using CSS classes. This is a much safer pattern to follow, as it's much easier to deal with CSS specificity if you are NOT using CSS IDs for styling purposes. The CSS reset code is baked right into `bootstrap.min.css` so there is no need to include any further CSS files for the reset.

Taking the starter template further

Although we have our template set up, one of the main problems with static websites is when things change. If your project grew to 50, 100, or 500 pages and you wanted to possibly update to a new version of Bootstrap, you might be looking at having to update all of those files. This is extremely painful, to put it mildly. Now we enter static site generators.

Using a static site generator

One of the hottest trends right now in web development is the use of static site generators. What exactly does that mean? Instead of having several static files that require updating every time something changes globally, you can use a series of base templates then load your body content into them. This is sometimes called includes or partials. This way, you only have one or two layout files that include the header and footer code.

Then, when something changes, you only have to update a few files instead of 500. Once your website is complete, you then generate a version that is plain HTML, CSS, and JavaScript, and deploy it to your server. This is what I would call creating your own frontend web development environment. This is also how most people work on larger projects nowadays to keep them manageable.

Converting the base template to a generator

Why don't we integrate the basic template into a generator so I can show you what I'm talking about? My generator of choice is called **Harp.js** and you can install it over at <http://harpjs.com/>.

Before we get too far ahead of ourselves, we need to install **Node.js**. Harp runs off Node.js so it's a dependency you'll need to use. If this feels too advanced for you, feel free to skip ahead to [Chapter 2, Using Bootstrap Build Tools](#). This section is totally optional. Head to the following URL to install Node.js if you don't already have it on your computer: <https://nodejs.org/download/>.

Follow the instructions on the Node.js website and, once you've finished installing it, run the following command in a command-line application such as Terminal or Cygwin:

```
$ node -v
```

This should spit out the version number of Node.js that you just installed and will also confirm that the installation worked. You should see something like this:

```
$ v0.10.33
```

Perfect, now let's move on to installing Harp.

Installing Harp.js

If you closed your command-line app, open it back up. If you are on a Mac, run the following command to install Harp:

```
$ sudo npm install -g harp
```

If you happen to be on a Windows machine, you'll need to enter a slightly different command, which is as follows:

```
$ npm install -g harp
```

After the installation completes, run the following command to get the Harp version number, which will also confirm that the installation was successful:

```
$ harp version
```

Adding Sass in Harp

I should also mention that most static site generators will also have built-in CSS preprocessors. This avoids you having to compile your Sass code somewhere else when working on your project. Harp includes both Sass and Less, so this will save you some time in upcoming chapters when we cover Sass in more detail.

Setting up the project

Before we convert our template to a Harp template, we need to set up the project structure. Create a new folder on your computer for the project then create the following subdirectories:

- css
- js
- img (if you plan on using images)
- partial
- fonts

Inserting the CSS

If you're storing the CSS files locally, copy `bootstrap.min.css` from your original project and add that into the new `/css` folder. In a future chapter, I'll show you how to code a custom Bootstrap theme. That file would also be included within this directory.

Inserting the JavaScript

The same pattern for the CSS will also apply to the JavaScript files. If you are storing `jquery.min.js` and `bootstrap.min.js` locally, then copy them into the new `/js` directory.

Other directories

The `/img` directory is optional and only applies if you plan to use images in your project. Ignore the `/partial` directory for now and I'll cover that a bit later. In the `/fonts` directory, you should drop in the Glyphicons icon set that comes with Bootstrap. If you downloaded Bootstrap, go back to the downloaded package and locate the font files. Copy them into this directory. Now that we have the project structure set up, we can start to break the basic page template down into a few different pieces.

Setting up the layout

In the root of your new Harp project, create a new file called `_layout.ejs`. **EJS** stands for **Embeddable JavaScript** and it's a type of template file that allows us to do more than just standard HTML. Within that file, copy and paste the code from our basic starter template. After you've inserted the code, we're going to make one change:

1. Locate the following line in the template and cut and paste it into a new file. Save the file and call it `index.ejs`:

```
<h1>Hello, world!</h1>
```

2. Return to the layout file and insert the following line of code immediately after the `<body>` tag:

```
<%- yield %>
```

3. Save both files then let me explain what is happening. The `yield` tag is a variable. Any page template such as `index.ejs` that lives in the same directory as the layout will be loaded in wherever you place the `yield` in the layout. So the `Hello, world!` line we inserted in the `index.ejs` file will load in here once you compile and launch your project.

Are you starting to see the advantage to this method? You could then go on and create other page templates so that all use this layout. In the future, if you need to make a change to the `<head>` of the layout, you only have to edit the one template file and it will be compiled into all of your final HTML files.

Compiling your project

Now that the template files are ready, we need to compile the project before we can preview it in the browser. Head back to your command-line app and make sure you are in the root of your project directory. Once there, run the following command to compile the project:

```
$ harp compile
```

Assuming you see no errors, your project was compiled successfully and can now be previewed in the browser. Before we move onto that step, though, take a look at your project directory and you'll see a `/www` folder. On compiling, Harp creates this directory and inserts the plain HTML, CSS, and JavaScript files. Assuming the website looks good when you preview, you then deploy the contents of the `/www` directory to your web server. More on deployment shortly.

Previewing your project

Harp has a built-in node web server that you can use to preview your project locally before deploying it. Open up your command-line app and run the following command from the root of your Harp project:

```
$ harp server
```

After doing so, you should see a message in the Terminal telling you that the server is successfully running. Open a web browser and navigate to the following URL to preview your project: <http://localhost:9000>.

Your project will load up in the browser and you should see the `Hello, world!` message that was inserted on compile. This is only a fraction of what you can do with Harp. To learn more about how Harp works, visit their website at <https://harpjs.com/>.

Deploying your project

If you're looking for a simple way to quickly deploy your project for testing, there is a tool called **Surge** from the same people that developed Harp. Surge is a free deployment solution; visit their website to learn more at <http://surge.sh/>.

Installing Surge

To install Surge, you'll need to open up your Terminal again. Once you have done this, run the following command:

```
$ npm install --global surge
```

This will install Surge and make it available anywhere on your computer.

Using Surge to deploy your project

To deploy your new project, navigate back to the root directory in the Terminal then run the following command:

```
$ surge
```

You'll now be prompted to log in or create a new account. Surge is free but you need to register an account to use it. You'll also notice in the Terminal that there is an autogenerated URL. This is the URL you can use to view your project live on the Internet. Once you've finished registering or logging in, visit the URL in your browser. If all went well, you should see the basic hello world page live.

Surge is a great tool if you're looking for a quick way to test your project on a live web server. If all goes well, you can then manually deploy your project to your own web server. Surge does offer a paid plan allowing for the use of a custom domain. So you could actually use it for your production deployment if that seems like a good idea.

Summary

That brings the first chapter to a close. I hope this chapter has proved to be a good introduction to Bootstrap 4 and provided you with a few advanced techniques for setting up your Bootstrap projects. In the next chapter, we'll take what we've learned here a step further by covering Bootstrap build tools. This will include a deeper explanation of how to use Harp, as well as other tools that are commonly used with Bootstrap.

2

Using Bootstrap Build Tools

In the previous chapter, we reviewed the process for setting up a basic Bootstrap template with the compiled framework files. What if you need to customize your Bootstrap build or you want to use additional development tools to make your life easier? This is possible through the use of a number of great tools. In this chapter, I'll show you how to install, set up, and use a number of build tools such as Node.js, Grunt.js, and Harp.js to extend Bootstrap and reveal the real power of the framework.

Different types of tools

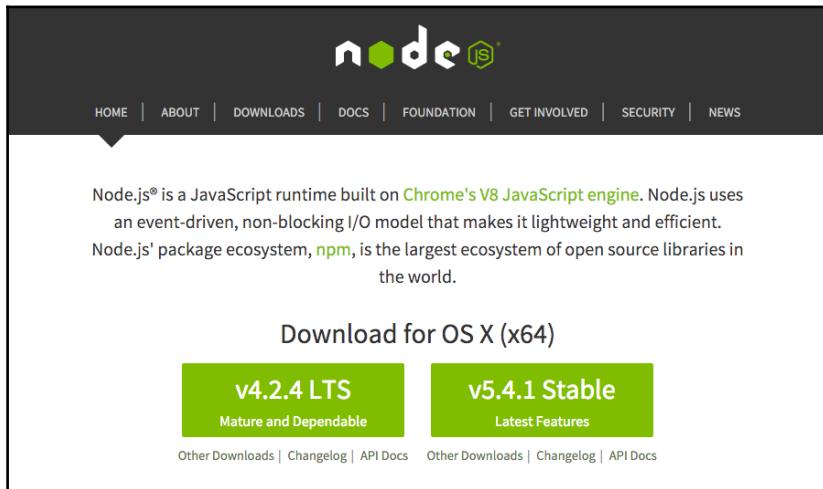
When working with Bootstrap, there are really three types of tools you need to be aware of. The first two are `Node.js` and `Grunt.js`. These are build tools and they take the development framework files and build them into the final files that you want to include in the production version of your projects. You wouldn't include development files on your actual production web server, as they are tools. You want to compile your files into production-ready HTML, CSS, and JavaScript that a web server can read and a browser can translate into a website.

The second type of tool you might want to use is a static website generator such as Harp.js. I talked a little bit about Harp in the first chapter but I will review it again in a little more detail. The main advantages of using Harp are things such as variables and partials in HTML, and a reusable-template-based system for your pages that allows you to reuse code.

The final types of tool you can use with Bootstrap are CSS preprocessors. In Bootstrap 4, the only option is Sass and we'll cover that in more depth later. However, before you can really start to learn to use Sass, you need to learn how to compile it into regular CSS files. Once you do, you can also start to use things such as variables and mixins in your CSS to make your files cleaner and easier to write.

Installing Node.js

If you skipped installing Node.js in Chapter 1, *Introducing Bootstrap 4* then now is the time to follow along and install and configure all your build tools. Let's start by heading to its website, <https://nodejs.org>, and downloading Node.js:



Node is a JavaScript runtime that uses Google Chrome's V8 JavaScript engine. What that means is that Node is a JavaScript-based web server that you can run locally or in production. It includes an event-driven, non-blocking I/O model which is easy to use and lightweight. Node comes with a built-in package manager called npm which includes the largest ecosystem of open source libraries on the Web.

Follow the installation instructions on the Node.js website and once you're done, open up a command-line application such as Terminal or Cygwin. Run the following command:

```
$ node -v
```

This will print out the Node.js version number that you installed and will confirm that it worked. It should look something like this if successful:

```
$ v0.10.33
```

Updating npm

Now that Node is installed, let's ensure that the latest version of npm is also installed. npm is a package manager for Node and allows you to install useful tools such as Grunt, which we'll do in our next step.

The screenshot shows the npm homepage with the following data:

- Total packages: 228,343
- Downloads in the last day: 118,795,177
- Downloads in the last week: 769,866,206
- Downloads in the last month: 2,542,752,182

Below this, a section titled "packages people 'npm install' a lot" lists several popular packages:

Package	Description	Last Published
browserify	browser-side require() the node...	10.2.6 published 6 months ago by...
express	Fast, unopinionated, minimali...	4.13.1 published 6 months ago by...
pm2	Production process manager f...	0.14.3 published 7 months ago by...
grunt-cli	The grunt command line interf...	0.1.13 published 2 years ago by...
npm	a package manager for JavaSc...	2.13.0 published 7 months ago by...
karma	Spectacular Test Runner for Ja...	0.13.1 published 6 months ago by...
bower	The browser package manager	1.4.1 published 10 months ago by...
cordova	Cordova command line interfa...	5.1.1 published 7 months ago by...
coffee-script	Unfancy JavaScript	1.9.3 published 8 months ago by j...

To make sure the latest version of npm is installed, run the following command in the Terminal:

```
npm update -g npm
```

You may need to include sudo before this command in some cases.



Once the update is complete, we can safely start to install the other packages we'll need for our Bootstrap development environment.

Installing Grunt

Grunt is a JavaScript task runner and it's the tool that will do the actual compiling and building of the development Bootstrap files into the production versions.

The screenshot shows the official Grunt.js website. At the top, there's a navigation bar with links for "Getting Started", "Plugins", "Documentation", and "API". The main feature is a large orange bull logo on the left, with the word "GRUNT" in large, bold, brown letters and "The JavaScript Task Runner" in a smaller, brown sans-serif font below it. To the right of the logo, there are two sections: "Why use a task runner?" and "Why use Grunt?". Below these are "Latest Version" and "Latest News" sections. A sidebar on the left contains a "Free screencasts" section with a video thumbnail of a rooster and a link to "Ads by Bocoup".

Latest Version

- Stable: v0.4.5 (npm)
- Development: v0.4.6 (github)

Free screencasts about JavaScript, Flexbox, Node.js and more from the experts at Bocoup.

Latest News

Grunt 0.4.5 released
May 12, 2014

Why use a task runner?

In one word: automation. The less work you have to do when performing repetitive tasks like minification, compilation, unit testing, linting, etc., the easier your job becomes. After you've configured it through a [Gruntfile](#), a task runner can do most of that mundane work for you—and your team—with basically zero effort.

Why use Grunt?

The Grunt ecosystem is huge and it's growing every day. With literally hundreds of plugins to choose from, you can use Grunt to automate just about anything with a minimum of effort. If someone hasn't already built what you need, authoring and publishing your own Grunt plugin to npm is a breeze. See how to [get started](#).

Available Grunt plugins

Grunt provides automation and allows you to chain together repetitive tasks such as compiling, minification, linting, and unit testing. Therefore, it's commonly used in frameworks such as Bootstrap to build the source files into production. To install Grunt, run the following command in the Terminal:

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

If you receive any errors, you may need to add `sudo` to the beginning of the above command. After finishing your installation, run the following command to check the Grunt version number and confirm that everything is working properly:

```
$ grunt -v
```

You should expect to see something like this printed out in the Terminal:

```
grunt-cli v0.1.13
```

Download the Bootstrap source files

To allow us to compile source files into production, we now need to download the Bootstrap source files and install them on our local machine. Head to the following URL and download the Bootstrap source files:

<http://v4-alpha.getbootstrap.com/getting-started/download/>.

Once you've download the files, unzip the package and move the directory to where you want it to live on your computer. If you just want to leave it on the desktop for now, that is fine. You can safely move the project around before or after editing it. The next thing you need to do is install the project dependencies. First, navigate to the root of the download package in the Terminal. It will likely be called something like `bootstrap-4.0.0-alpha.2`. Once you are there, run the following command to install the files:

```
$ npm install
```

If you get any type of error, try including `sudo` at the beginning of the command.



If you are using `sudo`, you'll likely be prompted for your system password. Type it in then hit *Enter* to execute the command.

Installing Ruby

Another tool you need to work with the Bootstrap source files is Ruby. Ruby is an object-oriented programming language that was designed in the 1990s in Japan. If you are familiar with Perl, you will likely enjoy Ruby, as Perl was the main inspiration for the language.

The screenshot shows the official Ruby website. At the top, there's a navigation bar with links for Downloads, Documentation, Libraries, Community, News, Security, and About Ruby. Below the navigation is a banner with the text "Ruby is..." and a description of Ruby as a dynamic, open-source programming language. A red "Download Ruby" button is prominently displayed. To the right, there's a code snippet in a box:

```
# Ruby knows what you
# mean, even if you
# want to do math on
# an entire Array
cities = %w[ London
Oslo
Paris
Amsterdam
Berlin ]
visited = %w[Berlin Oslo]

puts "I still need " +
"to visit the " +
"following cities:",
cities - visited
```

Below the banner, there's a news announcement for "Ruby 2.3.0 Released" with a link to "Continue Reading...". On the right side, there's a sidebar with a "Get Started, it's easy!" heading and links to "Try Ruby!", "Ruby in Twenty Minutes", and "Ruby from Other Languages".

In Bootstrap, Ruby is used to run the documentation website and to compile the core Sass files into regular CSS. For the Bootstrap documentation, you can always visit <http://getbootstrap.com/>. However, in some cases, you may find yourself offline, so you might want to install a local version of the docs that you can use. Let's first start by installing Ruby before we get to the documentation.

Good news! If you're on a Mac, Ruby comes pre-installed with OS X. Run the following command to check the Ruby version number and verify that it's available:

```
$ ruby -v
```

If Ruby is installed, you should see something like this in the Terminal:

```
$ ruby 1.9.2p320 (2012-04-20 revision 35421) [x86_64-
darwin12.3.0]
```

If you're on a Windows machine, you may need to manually install Ruby. If that's the case, check out the following website to learn how to install it: <http://rubyinstaller.org/>.

Installing the Bundler gem

After Ruby is ready to roll, you need to install a Ruby gem called Bundler. In the words of the developers of Bundler: *Bundler provides a consistent environment for Ruby projects by tracking and installing the exact gems and versions that are needed.* For more info on Bundler, please visit <http://bundler.io/>.

Don't worry too much about what Bundler does. The important thing is to just install it and move on.

1. To do this, we need to run the following command in the Terminal in your Bootstrap source file root directory:

```
$ gem install bundler
```

2. Again, if you get any errors, just begin the command with `sudo`. To confirm your installation of Bundler, run the following command to view the version number as in our previous examples:

```
$ bundler -v
```

3. If all is good, you should see something like this printed out in the Terminal:

```
$ Bundler version 1.11.2
```

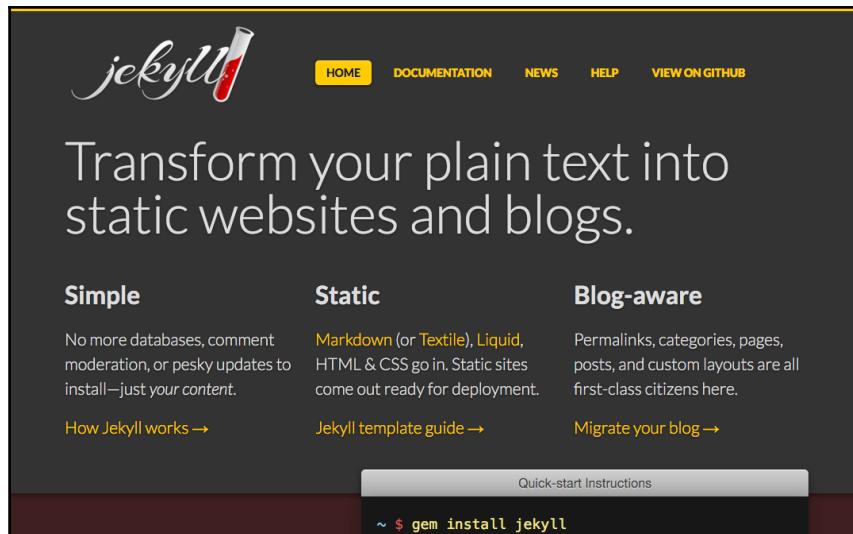
4. The last step you need to do is install the actual documentation bundle of files. Do this by running the following command in the Terminal from your root Bootstrap directory:

```
$ bundle install
```

5. This will install all Ruby dependencies, such as **Jekyll**, which is used for the documentation, and the Sass compiler we'll need a little later in the book. If you're a Windows user and you want to run the Jekyll documentation locally then you should check this out: <http://jekyll-windows.juthilo.com/>.



Jekyll is a database-independent static site generator that will convert plain text into a static website or blog. You can write templates in Markdown, Textile, Liquid, or HTML and CSS. On deployment, the code will be compiled into production-ready files that can be uploaded to a web server or run locally. That completes the setup for the first part of the Bootstrap build tools. Before we move onto the static site generator portion, let me show you how to run the documentation locally.



Running the documentation

Getting the documentation running locally is actually pretty easy. From the root of the Bootstrap source file directory, run the below command in the terminal:

```
bundle exec jekyll serve
```

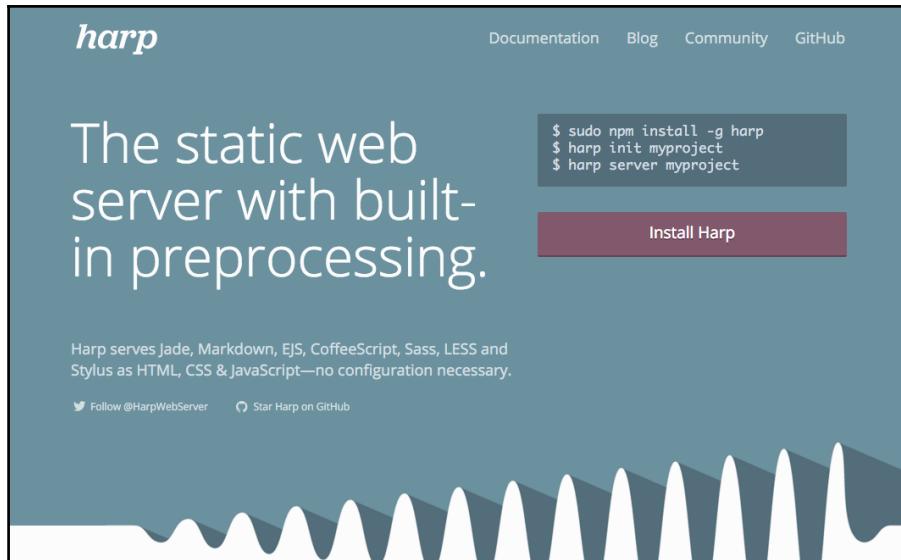
In the Terminal, you'll see that the server is running. The next step is to open up a web browser and enter the following address:

`http://localhost:9001/`

The Bootstrap documentation website will load up and now you have a local version of the documentation! To quit out of the server, hit `Ctrl + C` and you will exit.

Setting up the static site generator

In Chapter 1, *Introducing Bootstrap 4* I gave you a quick overview of setting up Harp.js, which is a static site generator. In this chapter, I'll go into more depth on how to properly set up your website, CSS, HTML, and JavaScript files. Before I do that though, we should talk about why you might want to use Harp.js.



Why use Harp.js

There are a number of great arguments for using a static site generator such as Harp.js: cleaner code, modern best practices, and more. However, the best reason is that it will just make your life simple. Instead of having to update a header on all 50 pages of the website, you can simply update the header partially and then have that compiled into all your templates. You can also take advantage of using variables to insert content and configuration.

Installing Harp.js

Harp is another project that runs on Node.js so we can use `npm` to install it with the following command:

```
$ sudo npm install -g harp
```



If you did this in Chapter 1, *Introducing Bootstrap 4*, you can skip down to the next part of this chapter.

To confirm that Harp was successfully installed, let's use our version-checking trick by entering the following command into the Terminal:

```
$ harp version
```

If all is good, you should see something like this printed out in the Terminal:

```
$ 0.14.0
```

Harp is now installed and we can move on to setting up our project for the book.

Setting up the blog project

As I mentioned earlier, we're going to be building a blog throughout this book as we learn to use Bootstrap 4. Let's start by creating a new directory and call it something like `Bootstrap Blog`. Open up that folder and create the following sub-directories inside it:

- css
- fonts
- img
- js
- partial

CSS

The `css` directory will hold the Bootstrap framework's CSS file and a custom theme file which we'll build later on. Go to the Bootstrap source file directory and locate the `dist/css` folder. From there, copy `bootstrap.min.css` to our new blog project's `css` directory.

fonts

The `fonts` directory will hold either a font icon library such as Glyphicon or Font Awesome. Previously, Bootstrap shipped with Glyphicon but they have dropped it in version 4. If you wish to use it, you'll need to download the icon font set and then drop the files into this directory. You could also include a web font that you may want to use on your project in this directory. If you are looking for web fonts, a good place to start is Google Web Fonts.

img

The `img` directory will hold any images used in the blog.

js

The `js` or JavaScript directory will hold the Bootstrap framework JavaScript files. If you add any other third-party libraries, they should also be included in this directory. Go back to the Bootstrap source files one last time and locate the `dist/js` folder. From there, copy `bootstrap.min.js` to the `js` directory in the blog project.

partial

The `partial` directory will hold any reusable snippets of code that we want to use in multiple locations throughout our templates or web pages, for example, the header and footer for our project. It's important to note you can have as many partial files as you like or use none at all.

Within this folder, create two new files and name them `_header.ejs` and `footer.ejs`. For now, you can leave them blank.

EJS files

EJS stands for **Embeddable JavaScript**. This is a type of template file that allows us to use things such as partials and variables in our templates. Harp also supports Jade if you prefer that language. However, I prefer to use EJS because it is very similar to HTML and therefore really easy to learn. If you've ever used WordPress, it is very similar to using template tags to insert bits of content or components into your design.

Setting up the JSON files

Each Harp project has at least two JSON files that are used for configuring a project. JSON stands for JavaScript Object Notation and it's a lightweight format for data interchange. If that sounds complicated, don't worry about it. The actual coding of a JSON file is actually really straightforward, as I will show you now.

The first is called `_harp.json` and it's used for configuring global settings and variables that will be used across the entire blog. In this case, we're going to set up a global variable for the name of our project that will be inserted into every page template. Start by creating a new file in the root of blog project and call it `_harp.json`. Within the file, insert the following code:

```
{  
  "globals": {  
    "siteTitle": "Learning Bootstrap 4"  
  }  
}
```

Here's what's happening in this code:

- We're using the `globals` keyword so any variables under this will be available across all of our templates
- I've created a new variable called `siteTitle` which will be the title of the project
- I've inserted the name of the book, `Learning Bootstrap 4`, as the title for the project

That completes the setup of the global `_harp.json` file. In a little bit, I'll show you how to add the variable we set up to the main layout file.

Creating the data JSON file

The next thing we need to do is set up the `_data.json` file that can hold template-specific variables and settings. For our project, we'll set up one variable for each page template which will hold the name of the page. Create another file in the root of the blog project and name it `_data.json`. In that file, insert the following code:

```
{  
  "index": {  
    "pageTitle": "Home"  
  }  
}
```

Let me break down this code for you:

- `index` refers to a filename. In this case, it will be our home page. We haven't actually created this file yet but that is okay as we will in the next steps.
- I've created a variable called `pageTitle` which will refer to the title of each page template in our project

- Since this is the `index` template, I've assigned a value or name of `Home` to it

That completes the setup of the `_data.json` file for now. Later on, we'll need to update this file once we add more page templates. For now, this will give us the minimum resources that we need to get our project going.

Setting up the layout

Let's go ahead and set up the layout file for our project. The layout is a separate file that will be a wrapper for the content of all of our pages. It contains things such as the `<head>` of our page, a header partial, and a footer partial. This is one of the advantages to using a static site generator. We don't have to define this on every page so if we want to change something in our header, we only change it in the layout. On the next compile, all of the page templates' headers will be updated with the new code.

Create a new file in the root of the blog project called `_layout.ejs`. Since this is technically a type of layout file, we'll be creating it as an EJS template file. Once you've created the file, insert the following code into it:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- Required meta tags always come first -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,
shrink-to-fit=no">
    <meta http-equiv="x-ua-compatible" content="ie=edge">

    <title><%- pageTitle %> | <%- siteTitle %></title>

    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="css/bootstrap.min.css">
  </head>
  <body>

    <%- partial("partial/_header") %>

    <%- yield %>

    <%- partial("partial/_footer") %>

    <!-- jQuery first, then Bootstrap JS. -->
    <script
      src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js"></sc
```

```
ript>
  <script src="js/bootstrap.min.js"></script>
</body>
</html>
```

There are a few things going on here, so let me explain everything that you need to know:

- The top is your standard `<head>` section that matches the basic Bootstrap template we covered in the first chapter, with, however, a few differences.
- Note the `<title>` tag and that it includes the two variables we set up previously. One for the `pageTitle` variable which will print out **Home** if we are on the index page. The second `siteTitle` variable will always print out **Learning Bootstrap 4** as that is what we set it to in `_harp.json`.
- Skip down to the `<body>` section and you'll see some new lines of code. The first partial is for our header. This line will include a snippet of code that we'll set up later that contains the markup for our header. Since this will be the same on all pages, we only need to include it here once instead of on every page.
- The second section in the `<body>` is the `<%- yield %>` tag. This is a Harp template tag and here is where the contents of our page template files will load. In the case of our index page, any code that we enter into `index.ejs` (that we need to create still) will be loaded in at this place in the layout.
- The final line of code is a partial for the footer and works exactly the same as the header. At a minimum, you should have a header and footer partial in your projects. However, you are free to add as many partials as you like to make your project more modular.

That completes the setup of the layout. Next, let's move on to coding the header and footer partials.

Setting up the header

Let's set up our first partial by coding the header. We'll use the Bootstrap navbar component here for our global navigation for the blog. In the partial directory, open up the `_header.ejs` file that you created a little earlier and insert the following code:

```
<nav class="navbar navbar-light bg-faded">
  <a class="navbar-brand" href="#">Learning Bootstrap 4</a>
  <ul class="nav navbar-nav">
    <li class="nav-item active">
      <a class="nav-link" href="index.html">Home</a>
    </li>
```

```
<li class="nav-item">
    <a class="nav-link" href="about.html">About</a>
</li>
<li class="nav-item">
    <a class="nav-link" href="contact.html">Contact</a>
</li>
</ul>
<form class="form-inline pull-xs-right">
    <input class="form-control" type="text" placeholder="Search">
    <button class="btn btn-primary" type="submit">Search</button>
</form>
</nav>
```

If you're a Bootstrap 3 user, you'll likely notice that the code to render a navbar in version 4 is much cleaner. This will make the navbar much easier to use and explain. Let me break down the code for you:

- On the `<nav>` tag, we have a few classes we need to include. `.navbar` is the standard class need for this component. `.navbar-light` will render a light-colored navbar for us. There are some other color options you can check out in the Bootstrap documents. Finally, the `.bg-faded` class is optional but I like to include it as it makes the background of the navbar a little more subtle.
- The `.navbar-brand` class is unchanged from Bootstrap 3 and I've inserted the name of the book for this tag. Feel free to name it whatever you want.
- Next, we have our navigation list of links. The `` tag needs to have the two required classes here: `.nav` and `.navbar-nav`.
- Within the list, you'll notice three pages: Home, About and Contact. These are going to be the pages we'll build out through later chapters so please fill them in now.



Note the `.active` class on the index page link. This is optional and you may not want to include it in this manner as this is a global navigation.

- Finally, I've included a search form and used the `.pull-xs-right` to align it to the right of the navbar. If you're familiar with Bootstrap 3, this class used to simply be called `.pull-right`. In Bootstrap 4, you have more control of the alignment based on the viewport size of your device. If you always want the search bar to be aligned to the right then use the `-xs` value in the class.

Save the file and that will complete the setup of the header partial. Let's move on to setting up the footer.

Setting up the footer

The footer partial works exactly like the header. Open up the `_footer.ejs` file in the partial directory that we created earlier and paste in the following code:

```
<!-- footer //-->
<div class="container">
    <div class="row">
        <div class="col-lg-12">
            Learning Bootstrap 4 2016
        </div>
    </div>
</div>
```

The footer content is going to be quite basic for our blog. Here's a breakdown of the code:

- I'm using the `.container` class to wrap the entire footer, which will set a max width of 1140 px for the layout. The navbar wasn't placed into a container so it will stretch to the full width of the page. The `.container` class will also set a left and right padding of .9375rem to the block. It's important to note that Bootstrap 4 uses REMs for the main unit of measure. EMs has been deprecated with the upgrade from version 3. If you're interested in learning more about REMs, you should read this [blog post](http://snook.ca/archives/html_and_css/font-size-with-rem).
http://snook.ca/archives/html_and_css/font-size-with-rem.
- It's also important to note that the column classes have NOT changed from Bootstrap 3 to 4. This is actually a good thing if you are porting over a project, as it will make the migration process much easier. I've set the width of the footer to be the full width of the container by using the `.col-lg-12` class.
- Finally I've entered some simple content for the footer, which is the book name and the year of writing. Feel free to change this up to whatever you want.
- Save the file and the footer setup will be complete.

We're getting closer to having our Harp development environment set up. The last thing we need to do is set up our index page template and then we can compile and view our project.

Creating our first page template

For our first page template, we're going to create our Home or index page. In the root of the blog project, create a new file called `index.ejs`. Note this file is not prepended with an underscore like the previous files. With Harp, any file that has the underscore will be compiled into another and ignored when the files are copied into the production directory. For example, you don't want the compiler to spit out `layout.html` because it's fairly useless with the content of the Home page. You only want to get `index.html`, which you can deploy to your web server. The basic thing you need to remember is to *not* include an underscore at the beginning of your page template files. Once you've created the file, insert the following code:

```
<div class="container">
  <div class="row">
    <div class="col-lg-12">
      <h1>hello world!</h1>
    </div>
  </div>
</div>
```

To get us started, I'm going to keep this really simple. Here's a quick breakdown of what is happening:

- I've created another `.container` which will hold the content for the Home page
- Within the container, there is a full-width column. In that column, I've inserted an `<h1>` with a `hello world!` message

That will be it for now. Later on, we'll build this page out further. Save the file and close it. We've now completed setting up all the basic files for our Harp development environment. The last step is to compile the project and test it out.

Compiling your project

When we compile a project in Harp, it will find all the different partial, layout, and template files and combine them into regular HTML, CSS, and JavaScript files. We haven't used any Sass yet but, as with the template files, you can have multiple Sass files that are compiled into a single CSS file that can be used on a production web server. To compile your project, navigate to the root of the blog project in the Terminal. Once you are there, run the following command:

```
$ harp compile
```

If everything worked, a new blank line in the terminal will appear. This is good! If the compiler spits out an error, read what it has to say and make the appropriate changes to your template files. A couple of common errors that you might run into are the following:

- Syntax errors in `_harp.json` or `_data.json`
- Syntax errors for variable or partial names in `_layout.ejs`
- If you have created additional page templates in the root of your project, and *not* included them in `_data.json`, the compile will fail

Once your compile is successful, head back to the root of the blog project and notice that there is a new `www` directory. This directory holds all the compiled HTML, CSS, and JavaScript files for your project. When you are ready to deploy your project to a production web server, you would copy these files up with FTP or using another means of file transfer. Every time you run the `harp compile` command in your project, these files will be updated with any new or edited code.

Running your project

Harp has a built-in web server that is backed by `Node.js`. This means you don't need a web hosting account or web server to actually test your project. With a simple command, you can fire up the built-in server and view your project locally. This is also really great if you are working on a project somewhere with no Internet connection. It will allow you to continue building your projects Internet-free. To run the server, head back to the Terminal and make sure you are still in the root directory of your blog project. From there, enter the following command:

```
$ harp server
```

In the Terminal, you should see a message that the server is running. You are now free to visit the project in a browser.

Viewing your project

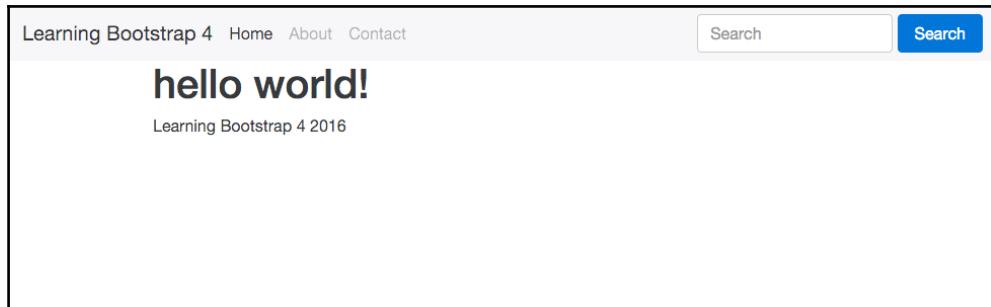
Now that the project is up and running on the web server, simply navigate to the following URL to view it: `http://localhost:9000`.

By default, Harp runs on port `9000` but you can specify a different port by modifying the last command. Go back to the terminal and quit the server by hitting `Ctrl + C`. Now enter the following command:

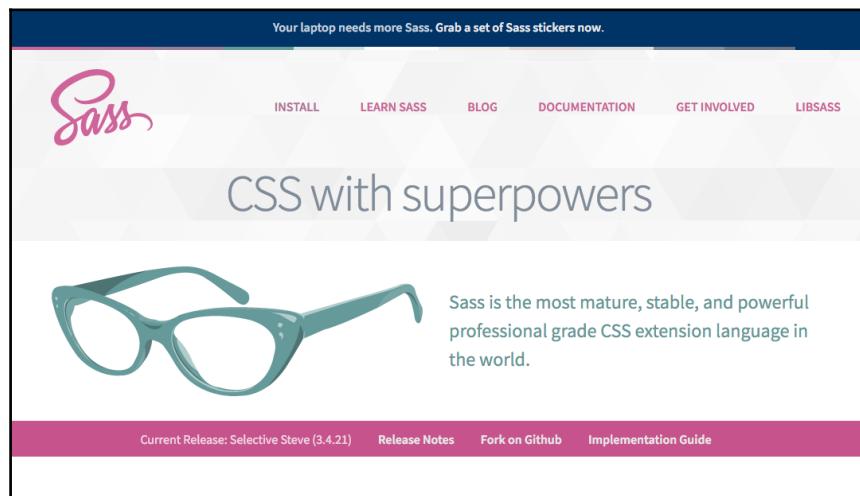
```
$ harp server --port 9001
```

Using this command, you can invoke any port you would like to use. Head back to the web browser again and change the URL slightly to read `http://localhost:9001`.

Your project should load for you and look something like this:



It might not be much to look at right now but it works! Your project is successfully set up and running. In future chapters, we'll add to this page and build some more using additional Bootstrap 4 components.



A note about Sass

When building a project with Bootstrap 4, there are two ways you can work with Sass. The first would be by editing the actual Bootstrap framework files and then recompiling them using Grunt. This is useful if you'd like to use something such as Flexbox for your grid layout. I'll discuss this in greater depth in the next chapter. The other way you might want to use Sass is to craft a unique theme that applies a custom look and feel to Bootstrap. This is done in the actual Harp project. Within the `css` directory, you can include Sass files; when you compile your Harp project, they will be converted to regular CSS, as Harp has a built-in Sass compiler. Then it is just a simple exercise of including those additional files in your layout template. I'll also get into that a little later in the book but I wanted to point out the difference now.

Summary

That brings the second chapter to a close. We've covered how to use a number of Bootstrap build tools. Let's review what we learned:

- How to install and run `Node.js` and `npm`
- How to install Grunt
- How to install Ruby
- How to navigate the Bootstrap source files
- How to install the Harp static site generator
- How to set up a basic project with Harp and run it locally
- Some important notes about how you can use Sass in Bootstrap 4

Now that our environment is set up and ready to go, we'll start coding the blog in the next chapter. To get us started, we'll jump right into learning about how to use a Flexbox layout in Bootstrap.

3

Jumping into Flexbox

Alright, now that we have finished setting up all the Bootstrap build tools, let's jump into an actual great new feature of Bootstrap 4. The latest version of the framework comes with CSS Flexbox support. The goal of the Flexbox layout module is to create a more effective way of designing a layout for a website or web application. The grid of boxes is aligned in a way that distributes them across their container even if their size is unknown. This is where the "Flex" in Flexbox comes from.

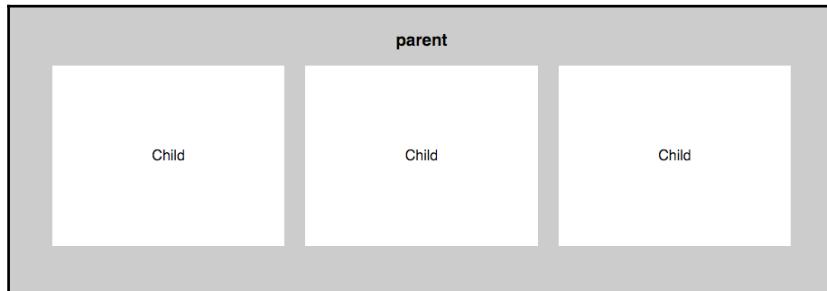
The motivation for a flexible box arose from a web design for mobiles. A way to have a section grow or shrink to best fill the available space was needed when building responsive web applications or websites. Flexbox is the opposite of block layouts that are either vertically or horizontally driven. It's important to note that Flexbox is generally best suited for use when designing web applications. The traditional grid method still works best for larger websites.

In our blog project, we're going to use Flexbox to create a homepage. There will be several rows of blocks, each being a post. I'll show you a few ways to lay the blocks out and different ways you can customize the contents of each block, all using the new Flexbox layout in Bootstrap.

Flexbox basics and terminology

Before we go too far, we should define a few Flexbox basics and some terminology that I'll use throughout the chapter. Every Flexbox layout is dependent on an outer container. As we move through the chapter, I'll refer to this container as the **parent**. Within the parent container there will always be a collection of boxes or blocks. I'll refer to these boxes as **children** or **child** elements of the parent. Why don't we start by talking a little bit more about why you would want to use Flexbox? The main purpose of Flexbox is to allow for the dynamic resizing of child boxes within their parent container.

This works for the resizing of both width and height properties on-the-fly. Many designers and developers prefer this technique as it allows for easier layouts with less code:



Ordering your Flexbox

Flexbox is a really powerful module as it comes with several properties that you can customize. Let's quickly go over some more basics before we fully take the plunge and use Flexbox in Bootstrap. Let's start by talking about the order of child boxes. By default, they will appear in the order that you insert them in the HTML file. Consider the following code:

```
<div class="parent">
  <div class="child">
    1
  </div>
  <div class="child">
    2
  </div>
  <div class="child">
    3
  </div>
</div>
```

A proper CSS will produce a layout that looks like this:



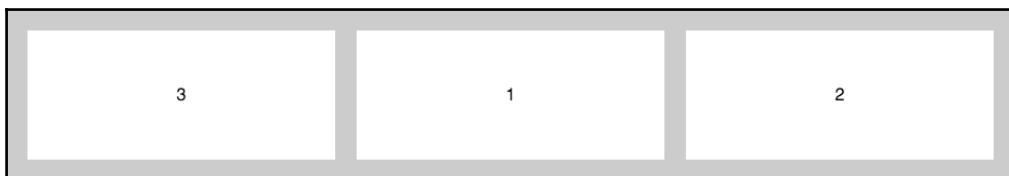
Here's the CSS to produce this layout if you are following along at home:

```
.parent {  
  display: flex;  
  background: #ccc;  
  padding: 10px;  
  font-family: helvetica;  
}  
  
.child {  
  padding: 10px;  
  margin: 10px;  
  background: #fff;  
  flex-grow: 1;  
  text-align:center;  
  height: 100px;  
  line-height: 100px;  
}
```

Now using an `order` property we can reorder the children using some CSS. Let's put the third box at the beginning. If you are reordering some blocks, you need to define the position for each one; you can't simply enter the value for a single block. Add the following CSS to your style sheet:

```
.child:nth-of-type(1) {  
  order: 2;  
}  
.child:nth-of-type(2) {  
  order: 3;  
}  
.child:nth-of-type(3) {  
  order: 1;  
}
```

I'm using the `nth-of-type` pseudo selector to target each of the three boxes. I've then used the `order` property to set the third box to the first position. I've also adjusted the other two boxes to move them over one space. Save the file and your boxes should now look like this:



As you can see, the third box has moved to the first position. It's as easy as that to rearrange blocks on boxes on a page. I think you'll likely see how this could be useful for coding up a web application dashboard.

Stretching your child sections to fit the parent container

Another important Flexbox feature is the ability to stretch the width of the child boxes to fit the full-width of the containing parent. If you look at the preceding CSS you'll notice a `flex-grow` property on the `.child` class. The property is set to 1 which means that the child boxes will stretch to equally fill their parent. You could also do something where one box is set to a different value, using the `nth-of-type` selector, and then it would be wider than the others. Here's the code to create equal-width columns as that is what you'll likely do in most cases:

```
.child {  
    flex-grow: 1;  
}
```

Changing the direction of the boxes

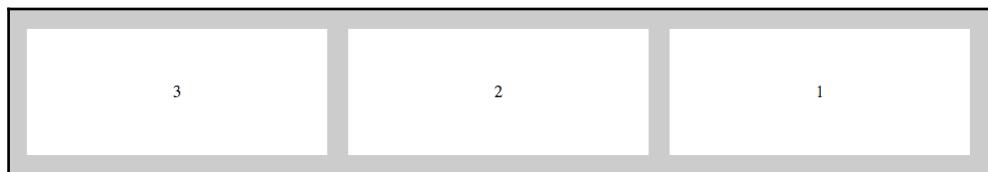
By default in Flexbox, the child boxes will be in a row going left to right. If you like, you can change the direction using the `flex-direction` property. Let's try out a few different directions. First let's review our base HTML code again:

```
<div class="parent">  
    <div class="child">  
        1  
    </div>  
    <div class="child">  
        2  
    </div>  
    <div class="child">  
        3  
    </div>  
</div>
```

Here's the base CSS we wrote a little earlier. However, this time we'll add the `flex-direction` property (with a value of `row-reverse`) to the `.parent` class. This will reverse the order of the boxes:

```
.parent {  
  display: flex;  
  flex-direction: row-reverse;  
  background: #ccc;  
  padding: 10px;  
}  
  
.child {  
  padding: 10px;  
  margin: 10px;  
  background: #fff;  
  flex-grow: 1;  
  text-align:center;  
  height: 100px;  
  line-height: 100px;  
}
```

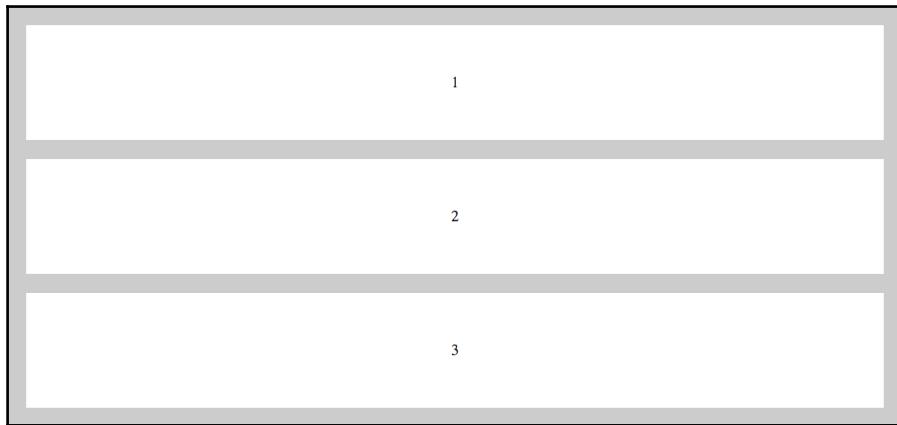
If you save the file and view it in a browser it should now look like this:



What if we wanted to order the boxes vertically so they were stacked on top of each other in descending order? We can do that by changing the `flex-direction` property to `column`:

```
.parent {  
  ...  
  flex-direction: column;  
}
```

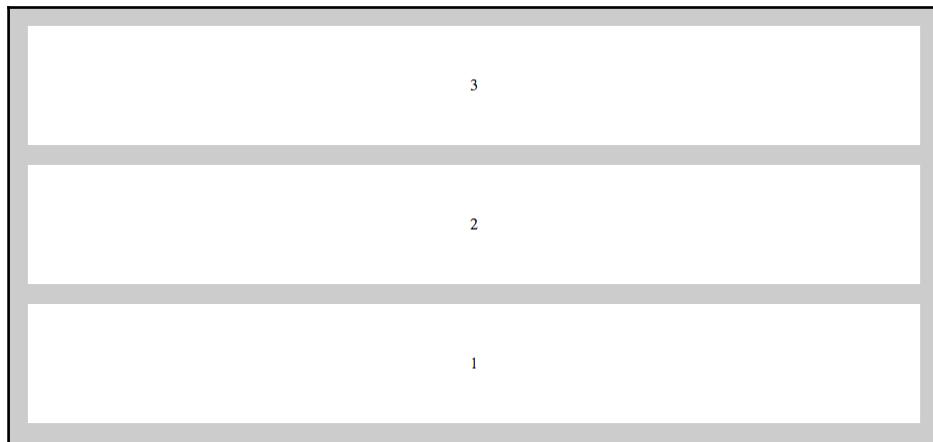
That configuration will produce a grid that looks like this:



Finally there is one more direction we can try. Let's do the same vertically stacked grid but this time we'll reverse it. We do that by switching the `flex-direction` property to `column-reverse`:

```
.parent {  
  ...  
  flex-direction: column-reverse;  
}
```

That will produce a grid that looks like this:



Wrapping your Flexbox

By default all of your child boxes will try to fit onto one line. If you have a layout with several boxes, this may not be the look you want. If this is the case, you can use the `flex-wrap` property to wrap the child boxes as needed. Let's add more boxes to our original code with the following HTML:

```
<div class="parent">
  <div class="child">
    1
  </div>
  <div class="child">
    2
  </div>
  <div class="child">
    3
  </div>
  <div class="child">
    4
  </div>
  <div class="child">
    5
  </div>
  <div class="child">
    6
  </div>
  <div class="child">
    7
  </div>
  <div class="child">
    8
  </div>
  <div class="child">
    9
  </div>
</div>
```

We now have nine boxes in our parent container. That should give us enough to work with to create a nice wrapping effect. Before we see what this looks like, we need to add some more CSS. Add the following properties to your CSS file:

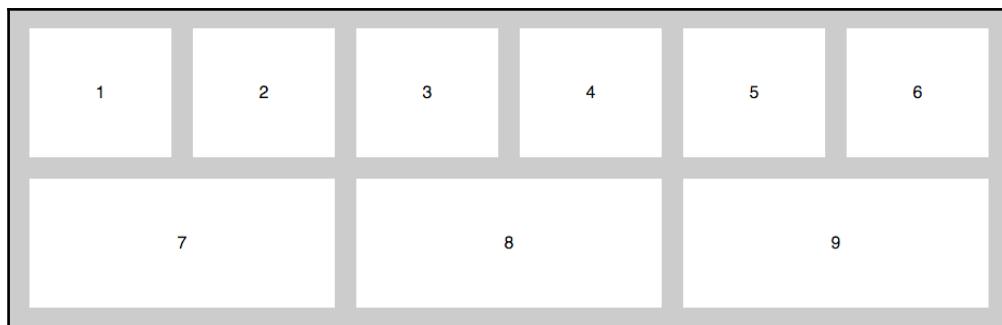
```
.parent {
  ...
  flex-wrap: wrap;
}
```

```
.child {  
  ...  
  min-width: 100px;  
}
```

I've added two new properties to achieve the layout we want. Let me break-down what is happening:

- I've added the `flex-wrap` property to the `.parent` class and set the value to `wrap`. This will wrap the boxes when it's appropriate.
- On the `.child` class I added a `min-width` of `100px`. I've done this so we can have some control on when the child boxes will break. If we don't add this, the width of the columns may get too thin.

Once you've added those properties to the existing code, save the file and test it. Your layout should now look something like this:



As you can see, we now have a two-row layout with six boxes on top and three below. Remember we added the `flex-grow` property previously, so the second row is stretching or growing to fit. If you want your boxes to always be equal you should use an even number, in this case 12. You could also remove the `flex-grow` property; then all the boxes would be the same width but they would not fill the layout the same way.

Creating equal-height columns

One of the best features of Flexbox is the ability to easily create equal height columns. In a regular horizontal layout, if your content is not the exact same length, each column will be a different height. This can be problematic for a web application layout because you usually want your boxes to be more uniform. Let's check out some regular layout code and what it looks like in the browser:

```
<div class="parent">
  <div class="child">
    Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
    tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
    veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
    commodo consequat.
  </div>
  <div class="child">
    Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
    tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
    veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
    commodo consequat. Lorem ipsum dolor sit amet, consectetur adipiscing elit,
    sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim
    ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip
    ex ea commodo consequat.
  </div>
  <div class="child">
    Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
    tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
    veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
    commodo consequat. Lorem ipsum dolor sit amet, consectetur adipiscing elit,
    sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim
    ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip
    ex ea commodo consequat. Lorem ipsum dolor sit amet, consectetur adipiscing
    elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut
    enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut
    aliquip ex ea commodo consequat. Lorem ipsum dolor sit amet, consectetur
    adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna
    aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris
    nisi ut aliquip ex ea commodo consequat. Lorem ipsum dolor sit amet,
    consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et
    dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation
    ullamco laboris nisi ut aliquip ex ea commodo consequat.
  </div>
</div>
```

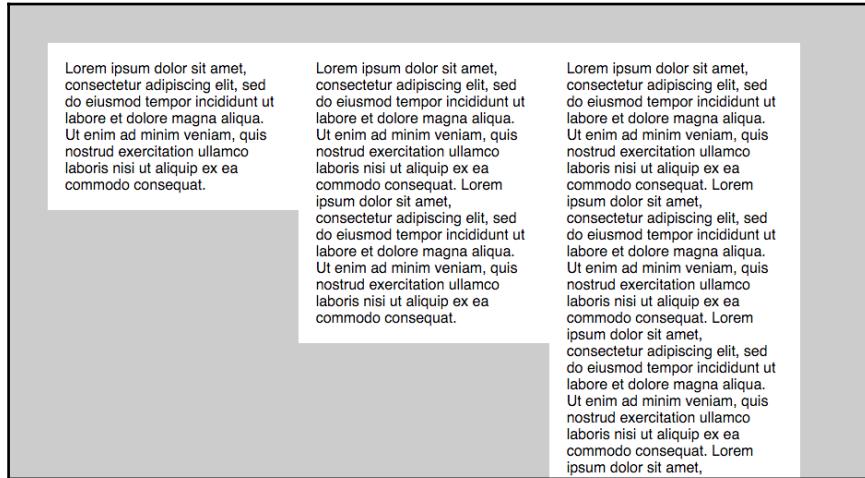
I've created three columns with different amounts of text in each of them. Let's add some basic styling to these columns:

```
.parent {
  width: 100%;
  background: #ccc;
  font-family: helvetica;
  padding: 5%;
  float: left;
}

.child {
```

```
padding: 2%;  
background: white;  
width: 25%;  
display: inline-block;  
float: left;  
}
```

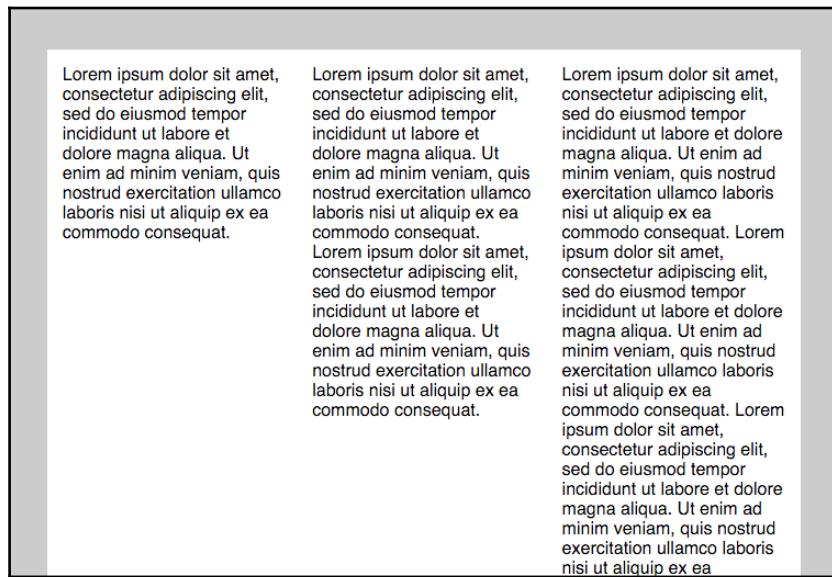
I've created a similar look and feel for this regular layout like our Flexbox. Let's see what this looks like if we view it in a browser:



That doesn't look very good does it? What would be better is if the two smaller columns stretched vertically to match the height of the longest column. The good news is this is really easy to do with Flexbox. Leave the HTML as it is but let's go and change our CSS to use a Flexbox approach:

```
.parent {  
  display: flex;  
  background: #ccc;  
  font-family: helvetica;  
  padding: 5%;  
}  
  
.child {  
  padding: 2%;  
  background: white;  
  flex-grow: 1;  
  min-width: 200px;  
}
```

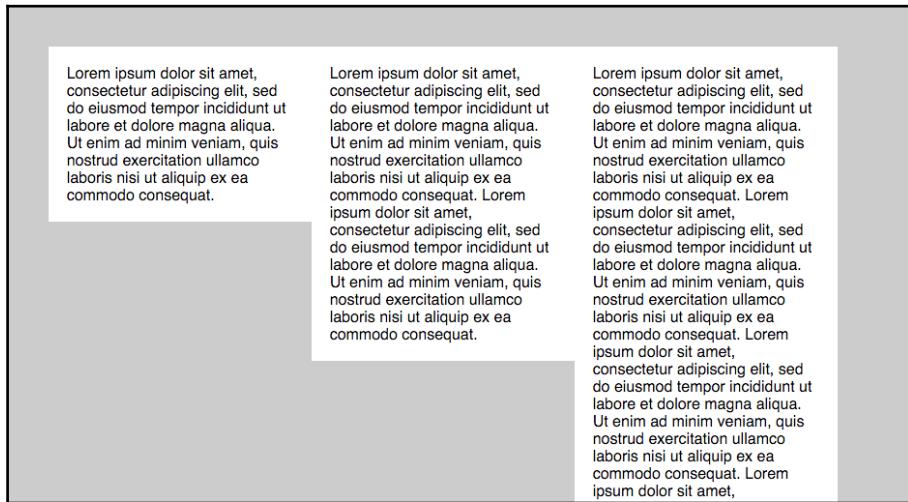
The preceding code is actually very similar to one of the first examples. Therefore, an equal height column comes standard right out of the Flexbox. I have added a `min-width` of `200px` to each column so that the text is readable. With the preceding CSS our layout will now look like this:



Perfect! Now the white background of each column has extended vertically to match the height of the tallest child. This looks much better and will allow for nicer horizontal alignment if you add additional rows of content. What's happening here is that the `align-items` property is defaulting to the `stretch` value. This value is what stretches the height of the columns to fit. There are some additional alignment values you can also try out. To continue, let's try out the `flex-start` value. Add the following CSS to the `.parent` class:

```
.parent {  
    ...  
    align-items: flex-start;  
}
```

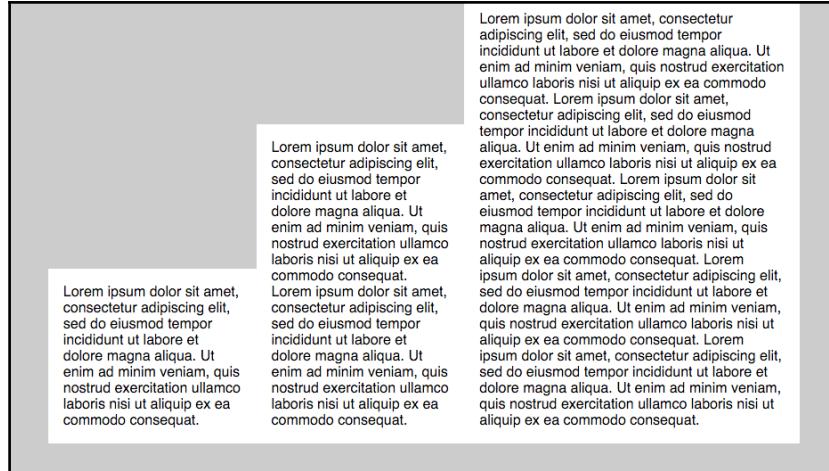
This configuration will actually undo the equal height columns and appear like a regular grid. Here's the image to refresh your memory:



A more useful value is the `flex-end` option, which will align the boxes to the bottom of the browser window. Change your CSS to:

```
.parent {  
  ...  
  align-items: flex-end;  
}
```

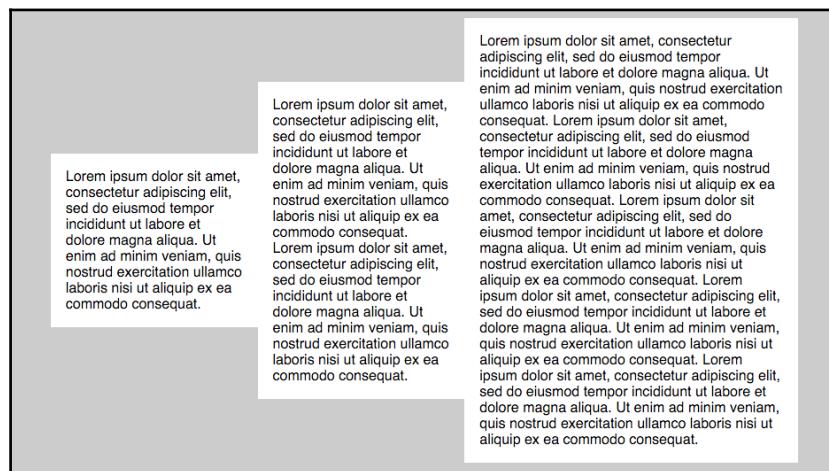
This setup will produce a grid that looks like this:



If you'd like to center your columns vertically in the layout, you can do that with the `center` value:

```
.parent {  
  ...  
  align-items: center;  
}
```

If you go for this setup, your grid will look like this:



This is just a taste of the properties you can use to customize the Flexbox grid. As I mentioned previously, I just wanted to give you a quick introduction to using Flexbox and some of the terminology that is needed. Let's take what we've learned and build on that by building a Flexbox grid in Bootstrap.

Setting up the Bootstrap Flexbox layout grid

Whether you are using Flexbox or not, the grid is based on Bootstrap's regular row and column classes. If you are familiar with the Bootstrap grid, this will work exactly as you expect it to. Before you start any Bootstrap project, you need to decide if you want to use a Flexbox or regular grid. Unfortunately, you can't use both at the same time in a Bootstrap project. Since the focus of this chapter is on Flexbox, we'll be using the appropriate grid configuration. By default Bootstrap is set up to use the regular grid. Therefore, we are going to need to edit the source files to activate the Flexbox grid. Let's start by downloading the source files again from <http://v4-alpha.getbootstrap.com/>.

Once you've downloaded the ZIP file, expand it and rename it so you don't get confused. Call it something like `Flexbox Bootstrap`. Next we'll need to edit a file and recompile the source files to apply the changes.

Updating the Sass variable

To use the Flexbox grid, we need to edit a Sass variable in the `_variables.scss` file. The way Sass variables work is that you set a single value in the `_variables.scss` file. When you run the built-in compiler, that value is written into every component of the Bootstrap framework where it is needed. You can then grab the compiled `bootstrap.min.css` file and it will have all the required code you need to use the Flexbox grid:

1. In your new source file directory, using the Terminal, navigate to:

```
$ scss/_variables.scss
```

2. Open the file in a text editor such as Sublime Text 2 or Notepad and find the following line of code:

```
$enable-flex: false !default;
```

3. Change the `false` value to `true`. The line of code should now read:

```
$enable-flex: true !default;
```

4. Save the file and close it. Before this change is applied, we need to recompile the source files. Since we downloaded a new version of the source files, we'll need to reinstall the project dependencies. Navigate to the root of the new Flexbox source files in the Terminal and run the following command:

```
$ npm install
```

5. This will likely take a couple minutes and you can follow the progress in the Terminal. Once it's done we need to compile the project. To do this we use Grunt. To run the compiler, simply enter the following command into the Terminal:

```
$ grunt
```

Again this will take a minute or two and you can follow the progress in the Terminal. Once it completes, the source files will have been compiled into the `/dist` directory. If it isn't clear, the production files that you want to use in your actual project will be compiled into the `/dist` directory.

Before we move onto our project, it would be a good idea to confirm that everything worked. Go back to your text editor and open the `dist/css/bootstrap.css` file from the root of your source files.

This is the un-minified version of the compiled Bootstrap CSS framework file. Once it's open do a quick find (`cmd + f` on Mac or `Ctrl + f` on Windows) and search for `flex`. If everything worked, it should quickly find an instance of `flex` in the file. This confirms that your compile worked.

Setting up a Flexbox project

A Flexbox project is structured exactly like a regular one. You just have to be sure to replace the `bootstrap.min.css` file in the `/css` directory with the new Flexbox version. Copy the project we made in the last chapter and paste it wherever you want on your computer. Rename the project to something like `Flexbox project`. Now open up that project and navigate to the `/css` directory. In a new window, open up the Flexbox sources files directory and navigate to the `/dist/css/` directory. Copy the `bootstrap.min.css` file from `/dist/css` into the `/css` directory in your new `Flexbox project`. You'll be prompted to overwrite the file and you should choose **Yes**. That's it, your new Flexbox project is ready to roll.

It would be a good idea to keep the Flexbox source files somewhere on your computer. In future projects, you can simply copy the compiled Flexbox version of the Bootstrap CSS over, saving you the trouble of having to recompile the source files each time you want a Flexbox layout.

Adding a custom theme

Before we code our first Flexbox grid, we need to add a custom CSS theme to our project. We're going to do this to add any custom look and feel styles on top of Bootstrap. In Bootstrap you never want to edit the actual framework CSS. You should use the cascading power of CSS to insert a theme for additional custom CSS or to overwrite existing Bootstrap styles. In a later chapter, I'll go into more depth on custom themes but for now let's set up a basic one that we can use for our Flexbox grid. First, let's start by creating a new file in the /css directory of our project called `theme.css`. For now, the file can be blank; just make sure you create it and save it.

Next we need to update our `_layout.ejs` file to include the theme file in our page. Open up `_layout.ejs` in a text editor and make sure it matches the following code:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- Required meta tags always come first -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,
shrink-to-fit=no">
    <meta http-equiv="x-ua-compatible" content="ie=edge">

    <title><%- pageTitle %> | <%- siteTitle %></title>

    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="css/bootstrap.min.css">
    <link rel="stylesheet" href="css/theme.css">
  </head>
  <body>

    <%- partial("partial/_header") %>

    <%- yield %>

    <%- partial("partial/_footer") %>

    <!-- jQuery first, then Bootstrap JS. -->
    <script
```

```
src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js"></script>
<script src="js/bootstrap.min.js"></script>
</body>
</html>
```

I've added one line of code to the template that loads in `theme.css`:

```
<link rel="stylesheet" href="css/theme.css">
```



Note that this line of code is after `bootstrap.min.css`. This is important as our theme needs to be loaded last so that we can overwrite Bootstrap default styles if we want to. Our template is now up-to-date and we are ready to start with our first grid. Feel free to keep `theme.css` open as we'll be adding some styles to it in the next step.

Creating a basic three-column grid

Now that we've set up our project, let's go ahead and start doing some Bootstrap coding. The good news is that the Bootstrap column classes used with the Flexbox grid are exactly the same as the ones used in a regular grid. There is no need to learn any new class names. In your project folder, create a new file and name it `flexbox.ejs`.

Before you go any further, you need to add an instance for this page to `_data.json`. Otherwise your `harp compile` command will fail. Open up `_data.json` and add the following code:

```
{
  "index": {
    "pageTitle": "Home"
  },
  "flexbox": {
    "pageTitle": "Flexbox"
  }
}
```

I've added a second entry for `flexbox.ejs` and given it this page title: Flexbox. Now we can safely start working on `flexbox.ejs` and the compile will work. Let's start with a simple three-column grid. Enter the following HTML code into `flexbox.ejs`:

```
<div class="container">
  <div class="row">
    <div class="col-md-4">Lorem ipsum dolor sit amet, consectetur
    adipiscing elit. Nullam eget ornare lacus. Nulla sed vulputate mauris. Nunc
```

```
nec urna vel sapien mattis consectetur sit amet eu tellus.</div>
<div class="col-md-4">Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Nullam eget ornare lacus. Nulla sed vulputate mauris. Nunc
nec urna vel sapien mattis consectetur sit amet eu tellus. Suspendisse
tempus, justo sed posuere maximus, velit purus dictum lacus, nec vulputate
arcu neque et elit. Aliquam viverra vitae est eu suscipit. Donec nec neque
eu sapien blandit pretium et quis est.</div>
<div class="col-md-4">Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Nullam eget ornare lacus. Nulla sed vulputate mauris. Nunc
nec urna vel sapien mattis consectetur sit amet eu tellus. Suspendisse
tempus, justo sed posuere maximus, velit purus dictum lacus, nec vulputate
arcu neque et elit. Aliquam viverra vitae est eu suscipit. Donec nec neque
eu sapien blandit pretium et quis est. Sed malesuada sit amet mi eget
pulvinar. Mauris posuere ac elit in dapibus. Duis ut nunc at diam volutpat
ultrices non sit amet nulla. Aenean non diam tellus.</div>
</div>
</div>
```

Let me breakdown what is happening here:

- Like in the earlier example, I've created three equal columns. Each one has a different amount of text in it.
- I'm using the `col-md-4` column class, as I want the three-column horizontal layout to be used for medium-size devices and upwards. Smaller devices will default to a single column width layout.
- I've also added a `.child` class to each of the column `<div>`s so that I can style them.

Now let's add a little CSS to `theme.css` so we can more easily see what is going on:

```
.child {
  background: #ccc;
  padding: 20px;
}
```

Here's what is happening with the `.child` class:

- I've added a light gray background color so we can easily see the child box.
- I've added some padding. Note that you can add padding to a Flexbox grid without worrying about breaking the grid. In a regular layout, this would break your box model and add extra width to the layout.

Here's what the finished layout should look like:

The screenshot shows a website layout with a header containing 'Learning Bootstrap 4' and links for Home, About, Contact, and a search bar. Below the header, there are three columns of text. The first column contains the first half of a long paragraph. The second column contains the second half of the same paragraph. The third column contains the third half of the paragraph. All three columns have equal height and are aligned vertically. The background of the page is white, and the text columns have a light gray background. At the bottom left, there is a footer with the text 'Learning Bootstrap 4 2016'.

As you can see the light gray background has stretched to fit the height of the tallest column. An equal height column with almost no effort is awesome! You'll also notice that there is some padding on each column but our layout is not broken.

You may have noticed that I used the regular `.container` class to wrap this entire page layout. What if we want the layout to stretch the entire width of the browser?

Creating full-width layouts

Creating a full-width layout with no horizontal padding is actually really easy. Just remove the `container` class. The HTML for that type of layout would look like this:

```
<div class="row">
  <div class="col-md-4 child">Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Nullam eget ornare lacus. Nulla sed vulputate mauris. Nunc
nec urna vel sapien mattis consectetur sit amet eu tellus.</div>
  <div class="col-md-4 child">Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Nullam eget ornare lacus. Nulla sed vulputate mauris. Nunc
nec urna vel sapien mattis consectetur sit amet eu tellus. Suspendisse
tempus, justo sed posuere maximus, velit purus dictum lacus, nec vulputate
arcu neque et elit. Aliquam viverra vitae est eu suscipit. Donec nec neque
eu sapien blandit pretium et quis est.</div>
  <div class="col-md-4 child">Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Nullam eget ornare lacus. Nulla sed vulputate mauris. Nunc
nec urna vel sapien mattis consectetur sit amet eu tellus. Suspendisse
tempus, justo sed posuere maximus, velit purus dictum lacus, nec vulputate
arcu neque et elit. Aliquam viverra vitae est eu suscipit. Donec nec neque
```

```
eu sapien blandit pretium et quis est. Sed malesuada sit amet mi eget  
pulvinar. Mauris posuere ac elit in dapibus. Duis ut nunc at diam volutpat  
ultrices non sit amet nulla. Aenean non diam tellus.</div>  
</div>
```

As you can see, I've simply removed the `<div>` with the `.container` class on it. Let's take a look at what the layout looks like now:

Learning Bootstrap 4 Home About Contact Search **Search**

Learning Bootstrap 4 2016

Three columns of placeholder text (Lorem ipsum) are displayed. The first column is white, while the second and third columns are a darker shade of gray. Each column contains three paragraphs of text.

There we go, the columns are stretching right to the edges of the browser now. We've easily created a full-width layout that has equal height columns. Let's improve on this design by making each column an actual blog post and we'll also add more rows of posts.

Designing a single blog post

Let's start by designing the layout and content for a single blog post. At the very least, a blog post should have: a title, post-meta, description, and a read more link. Open up the `flexbox.ejs` file and replace the first column's code with this new code:

```
<div class="col-md-4 child">  
  <h3><a href="#">Blog Post Title</a></h3>  
  <p><small>Posted by <a href="#">Admin</a> on January 1, 2016</small></p>  
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eget  
  ornare lacus. Nulla sed vulputate mauris. Nunc nec urna vel sapien mattis  
  consectetur sit amet eu tellus.</p>  
  <p><a href="#">Read More</a></p>  
</div>
```

Let me breakdown what is happening here:

- I've added an `<h3>` tag with a link for the post title
- I've added some post-meta and wrapped it in a `<small>` tag so it is subtle
- I've left our description and added a read more link at the bottom

Now go ahead and copy and paste this code into the other two columns. If you want to play around with the length of the description text, feel free. For this example I'm going to keep it the same. When you're done, the entire page code should look like this. Note, I added the container `<div>` back in:

```
<div class="container">
  <div class="row">
    <div class="col-md-4 child">
      <h3><a href="#">Blog Post Title</a></h3>
      <p><small>Posted by <a href="#">Admin</a> on January 1, 2016</small></p>
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eget ornare lacus. Nulla sed vulputate mauris. Nunc nec urna vel sapien mattis consectetur sit amet eu tellus.</p>
      <p><a href="#">Read More</a></p>
    </div>
    <div class="col-md-4 child">
      <h3><a href="#">Blog Post Title</a></h3>
      <p><small>Posted by <a href="#">Admin</a> on January 1, 2016</small></p>
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eget ornare lacus. Nulla sed vulputate mauris. Nunc nec urna vel sapien mattis consectetur sit amet eu tellus.</p>
      <p><a href="#">Read More</a></p>
    </div>
    <div class="col-md-4 child">
      <h3><a href="#">Blog Post Title</a></h3>
      <p><small>Posted by <a href="#">Admin</a> on January 1, 2016</small></p>
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eget ornare lacus. Nulla sed vulputate mauris. Nunc nec urna vel sapien mattis consectetur sit amet eu tellus.</p>
      <p><a href="#">Read More</a></p>
    </div>
  </div>
</div>
```

Save your file, do a harp compile if you haven't done so for in a while. Then do a harp server to launch the web server and head to `http://localhost:9000` to preview the page. It should look like this:

The screenshot shows a blog homepage with a header containing navigation links (Learning Bootstrap 4, Home, About, Contact) and a search bar. Below the header, there are three columns, each representing a blog post. Each post includes a title, a short bio, a preview of the content, and a "Read More" link.

Blog Post Title	Blog Post Title	Blog Post Title
Posted by Admin on January 1, 2016	Posted by Admin on January 1, 2016	Posted by Admin on January 1, 2016
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eget ornare lacus. Nulla sed vulputate mauris. Nunc nec urna vel sapien mattis consectetur sit amet eu tellus.	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eget ornare lacus. Nulla sed vulputate mauris. Nunc nec urna vel sapien mattis consectetur sit amet eu tellus.	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eget ornare lacus. Nulla sed vulputate mauris. Nunc nec urna vel sapien mattis consectetur sit amet eu tellus.
Read More	Read More	Read More

Learning Bootstrap 4 2016

Great, now we have a decent-looking blog homepage. However, we need to add more posts to fill this out. Let's go ahead and add more column `<div>`s inside the same row. Since this is Flexbox, we don't need to start a new `<div>` with a row class for each row of posts. Let's add three more posts in then see what it looks like:

The screenshot shows the same blog homepage after adding three more posts, resulting in a 2x3 grid of posts. The layout remains consistent with the previous screenshot, featuring a header with navigation and a search bar, followed by a grid of six blog posts arranged in two rows of three.

Blog Post Title	Blog Post Title	Blog Post Title
Posted by Admin on January 1, 2016	Posted by Admin on January 1, 2016	Posted by Admin on January 1, 2016
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eget ornare lacus. Nulla sed vulputate mauris. Nunc nec urna vel sapien mattis consectetur sit amet eu tellus.	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eget ornare lacus. Nulla sed vulputate mauris. Nunc nec urna vel sapien mattis consectetur sit amet eu tellus.	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eget ornare lacus. Nulla sed vulputate mauris. Nunc nec urna vel sapien mattis consectetur sit amet eu tellus.
Read More	Read More	Read More
Blog Post Title	Blog Post Title	Blog Post Title
Posted by Admin on January 1, 2016	Posted by Admin on January 1, 2016	Posted by Admin on January 1, 2016
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eget ornare lacus. Nulla sed vulputate mauris. Nunc nec urna vel sapien mattis consectetur sit amet eu tellus.	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eget ornare lacus. Nulla sed vulputate mauris. Nunc nec urna vel sapien mattis consectetur sit amet eu tellus.	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eget ornare lacus. Nulla sed vulputate mauris. Nunc nec urna vel sapien mattis consectetur sit amet eu tellus.
Read More	Read More	Read More

Perfect. Now our homepage is starting to take shape. Continue adding more posts until you have a number that you are happy with. At this point you should have a decent understanding of the Flexbox grid.

Summary

We started by reviewing the basic functionality of the Flexbox module and the terminology that goes along with it. Next, I showed you how to activate the Flexbox grid in Bootstrap by editing the Sass variable and recompiling the source files. Finally we got our hands dirty by learning how to build a blog homepage and feed using the Bootstrap Flexbox grid. In the next chapter, we'll move further into layouts and how you can set up your pages with Bootstrap.

4

Working with Layouts

The core of any Bootstrap website is the layout or grid component. Without a grid, a website is pretty much useless. One of the biggest layout challenges we face as web developers nowadays is dealing with a large array of screen resolutions, from desktop to tablets, mobile phones, and even Apple watches. It is not easy to lay out a website and we rely on responsive web design and media queries to take a mobile-first approach. Perhaps the best feature of the Bootstrap layout grid is that it's mobile-first and built using media queries. This takes the hardest part out of constructing a grid and lets us concentrate on the actual design of our projects. Before we start to layout the next part of our blog project, let's review the lay out grid basics in Bootstrap 4.

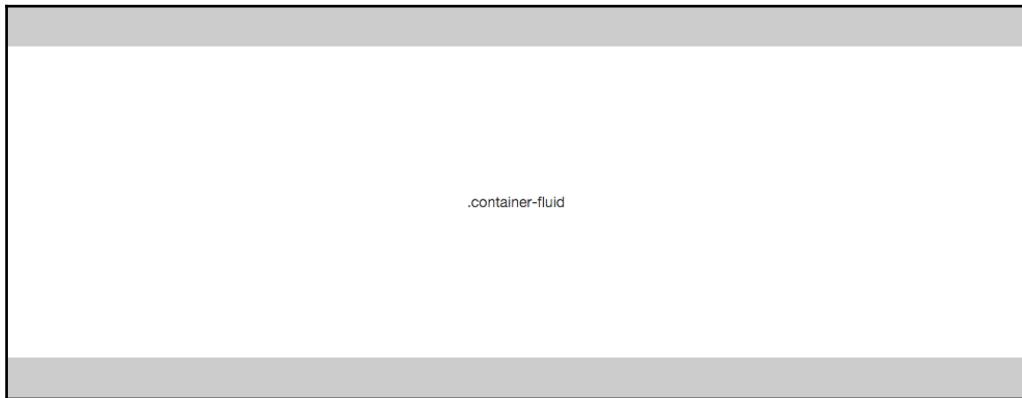
In this chapter, we are going to discuss the following listed topics briefly:

- Working with containers
- Adding columns to your layout
- Creating a simple three-column layout
- Coding the blog home page
- Using responsive utility classes

Working with containers

The base of any Bootstrap layout is a container class. There are two types of containers you can choose to use. The first is `.container-fluid`, which is a full-width box and will stretch the layout to fit the entire width of the browser window.

There is some left and right padding added so the content doesn't bump right up against the browser edge:



The second option is the basic `.container` class, which will have a fixed width based on the size of your device's viewport. There are five different sizes in Bootstrap, with the following width values:

- Extra small <544px
- Small >544px
- Medium >768px
- Large >992px
- Extra large >1140px



Let's take a look at the markup for both container types. I'll start with the basic `.container` class:

```
<div class="container">  
  ...  
</div>
```

That's pretty easy. Now let's look at the code for the fluid container:

```
<div class="container-fluid">  
  ...  
</div>
```

Again, that is straightforward and is all that you need to know about using the container classes in Bootstrap 4.

Creating a layout without a container

Now, in some cases, you may not want to use a container and that is totally fine. An example of this would be if you want a full width layout but you don't want the default left and right padding. You may have an image banner that you want to stretch to the full width of the browser with no padding. In this case, just remove the `<div>` with the `container` class on it.

Using multiple containers on a single page

It is perfectly fine to use multiple containers on a single page template. The containers are CSS classes, so they are reusable. You may want to do this on longer page layouts, perhaps a landing page design, where you have multiple large regions. Another example is using a container to wrap your footer. If you are using a template system like Harp, you'll want to create a footer partial. You can make the footer component more self contained by giving it its own container. Then you don't have to worry about closing a container `<div>` in the footer that was opened in a page template or even the header. I would recommend using multiple containers to make your designs more modular and easier to manage by multiple users. Let's take a quick look at how you would structure a basic page with multiple containers:

```
<div class="container">  
  <!-- header code goes here, harp partial name would be _header.ejs //-->  
</div>  
  
<div class="container">
```

```
<!-- template code goes here, harp file name would be index.ejs //-->
</div>

<div class="container">
    <!-- footer code goes here, harp partial name would be _footer.ejs //-->
</div>
```

We have three separate files there and using a container class for each makes each section more modular. You also don't have to worry about opening a `<div>` in one file then closing it in another. This is a good way to avoid orphan closing `</div>` tags.

Inserting rows into your layout

The next step in creating a layout is to insert at least a single row of columns. Each container class can have one or more rows nested inside of it. A row defines a collection of horizontal columns that can be broken up to twelve times. The magic number when it comes to columns in Bootstrap is twelve, and you can sub-divide them any way you like. Before we get into columns though, let's review the markup for rows. First let's look at an example of a container with a single row:

```
<div class="container">
    <div class="row">
        <!-- insert column code here //-->
    </div>
</div>
```

As you can see, this is an easy next step in setting up your layout. Like I mentioned, you can have as many rows within a container as you like. Here's how you would code a five-row layout:

```
<div class="container">
    <div class="row">
        <!-- insert column code here //-->
    </div>
    <div class="row">
        <!-- insert column code here //-->
    </div>
    <div class="row">
        <!-- insert column code here //-->
    </div>
    <div class="row">
        <!-- insert column code here //-->
    </div>
    <div class="row">
        <!-- insert column code here //-->
    </div>
```

```
</div>  
</div>
```

Like the container class, rows are also a CSS class, so they can be reused as many times as you like on a single page template.



You should never include actual contents inside a row `<div>`. All content should be included with column `<div>`s.

Adding columns to your layout

Before we jump into actually adding the columns, let's talk a little bit about the different column classes you have at your disposal with Bootstrap. In Bootstrap 3 there were four different column class widths to choose from: extra small, small, medium, and large. With Bootstrap 4, they have also introduced a new extra large column class. This is likely to allow for extra large monitors, like you would find on an iMac. Let's go over the fine points of each column class in Bootstrap 4.

Extra small

The smallest of the grid classes uses the naming pattern `.col-xs-#`, where `-#` is equal to a number from 1 to 12. Remember, in Bootstrap, your row must be divided into a number of columns that adds up to 12. A couple of examples of this would be `.col-xs-6` or `.col-xs-3`. The extra smallcolumn class is for targeting mobile devices that have a max-width of 544 pixels.

Small

The smallcolumn class uses the syntax pattern `.col-sm-#`, and some examples of that would be `.col-sm-4` or `.col-sm-6`. It is targeted for devices with a resolution greater than 544 pixels but smaller than 720 pixels.

Medium

The mediumcolumn class uses a similar naming pattern of `.col-md-#` and some examples would be `.col-md-3` or `.col-md-12`. This column class is for devices greater than 720 pixels and smaller than 940 pixels.

Large

The largecolumn class again uses the naming pattern of .col-lg-# and some examples would be .col-lg-6 or .col-lg-2. This column class targets devices that are larger than 940 pixels but smaller than 1140 pixels.

Extra large

The final and new column class is extra large and the syntax for it is .col-xl-# with examples being .col-xl-1 or .col-xl-10. This column class option is for all resolutions greater than or equal to 1140 pixels.

Choosing a column class

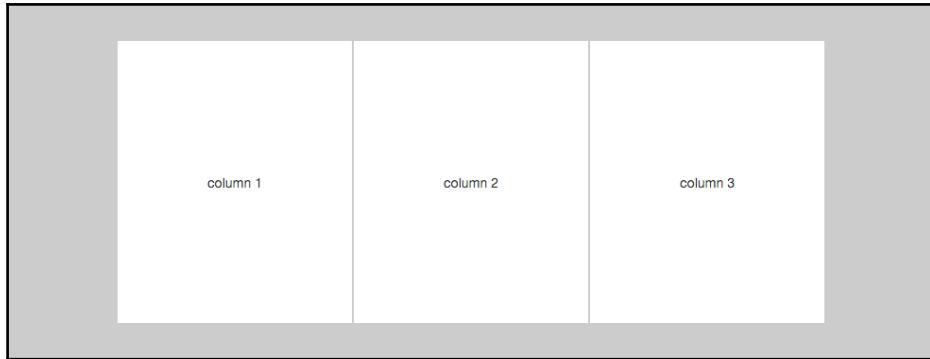
This is a good question. With all of the class options, it can be hard to decide which ones to use. If you are building a mobile app, then you would likely want to stick to the extra small or small column classes. For a tablet, you might want to use medium. If the primary user for your application will be on a desktop computer, then use the large or extra large classes. But what if you are building a responsive website and you need to target multiple devices? If that is the case, I usually recommend using either the medium or large column classes. Then you can adjust to use larger or smaller classes where necessary if you have a component that needs some extra attention for specific resolutions.

Creating a simple three-column layout

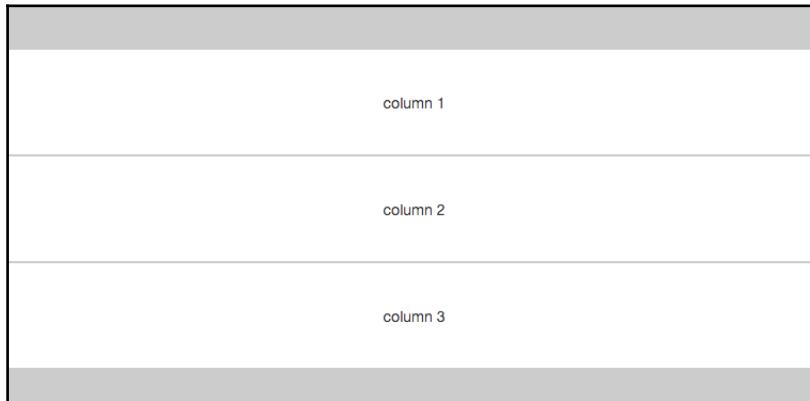
Let's assume that we are building a simple responsive website and we need a three-column layout for our template. Here's what your markup should look like:

```
<div class="container">
  <div class="row">
    <div class="col-md-4">
      <!-- column 1 //-->
    </div>
    <div class="col-md-4">
      <!-- column 2 //-->
    </div>
    <div class="col-md-4">
      <!-- column 3 //-->
    </div>
  </div>
</div>
```

As you can see, I've inserted three `<div>`s inside my row `<div>`, each with a class of `.col-md-4`. For devices that have a resolution of 768 pixels or greater, you'll see a three-column layout like this:



Now, if you were to view this same layout on a device with resolution smaller than 768 pixels, each column's width would change to 100% and the columns would be stacked on top of each other. That variation of the layout for smaller screens would look like this:



That's all well and good, but what if we wanted to have a different layout for the columns on smaller devices that didn't set them all to 100% width? That can be done by mixing column classes.

Mixing column classes for different devices

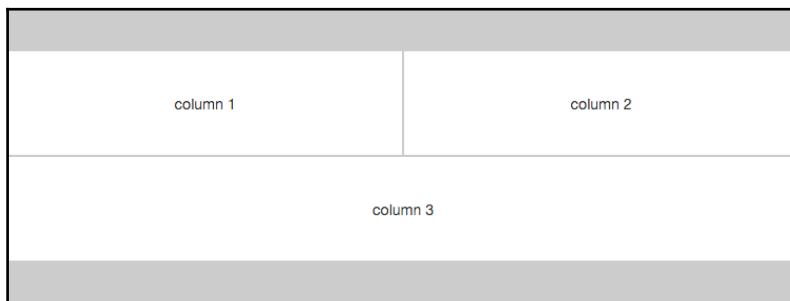
Adding additional classes to each of our column `<div>`s will allow us to target the grid layout for different devices. Let's consider our three-column layout from before, but this time, we want to lay it out like this:

- The first two columns should be 50% of the layout
- The third column should stretch 100% of the layout and be below the first two

To achieve this layout, we'll mix some different column classes. Here's what the markup should look like:

```
<div class="container">
  <div class="row">
    <div class="col-md-4 col-xs-6">
      <!-- column 1 //-->
    </div>
    <div class="col-md-4 col-xs-6">
      <!-- column 2 //-->
    </div>
    <div class="col-md-4 col-xs-12">
      <!-- column 3 //-->
    </div>
  </div>
</div>
```

I've added the `.col-xs-6` class to the first two column `<div>`s. Now, if our device resolution is less than 768 pixels, the first two columns will be set to a width of 50%. For the third column, I've used the `.col-xs-12` class, which will wrap the third column onto a new line and set it to 100% of the width of the layout. The resulting layout will look like this on smaller devices:



This will only apply to devices with a layout of less than 768 pixels. If you were to view this grid, using the latest code, on a larger device, it will still appear as three equal columns laid out horizontally.

What if I want to offset a column?

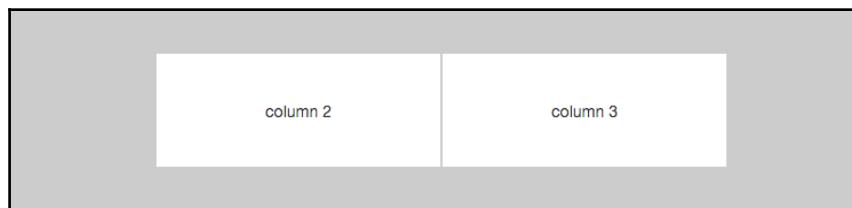
Perhaps your layout requires you to offset some columns and leave some horizontal blank space before your content starts. This can easily be used with Bootstrap's offset grid classes. The naming convention is similar to the regular column classes, but we need to add the offset keyword, like this: `.col-lg-offset-#`. A couple examples of this would be `.col-lg-offset-3` or `.col-md-offset-6`. Let's take our three-column grid from before but remove the first column. However, we want the second and third columns to remain where they are in the layout. This will require us to use the offset grid class. Here's what your markup should look like:

```
<div class="container">
  <div class="row">
    <div class="col-md-4 col-md-offset-4">
      <!-- column 2 //-->
    </div>
    <div class="col-md-4">
      <!-- column 3 //-->
    </div>
  </div>
</div>
```



Note how I removed the first column `<div>`. I also added the class `.col-md-offset-4` to the second column's `<div>`. Once you've done this, your layout should appear like this.

There you go; you've successfully offset your column by removing the first column and then sliding over the other two columns:



That concludes the basics of the Bootstrap grid that you'll need to know for the remainder of this chapter. Now that you have a good understanding of how the grid works, let's move onto coding up our blog home page layout using the grid classes.

Coding the blog home page

Now that you have a good grasp of how to use the Bootstrap 4 grid, we're going to code up our blog home page. This page will include a feed of posts, a sidebar, and a newsletter sign-up form section at the bottom of the page. Let's start by taking the code we wrote in Chapter 2, *Using Bootstrap Build Tools* for our `hello world!` template and duplicating the entire directory. Rename the folder Chapter 4: Working with Layouts or Bootstrap Layout.



Remember, we are using the regular grid moving forward, not the Flexbox grid. Make sure you are using the default build of `bootstrap.min.css`. If you carry out a simple duplication of the second chapter's code then you'll have the right file configuration.

Writing the index.ejs template

Good news! Since we set up our Harp project in Chapter 2, *Using Bootstrap Build Tools*, we can reuse a bunch of that code now for our blog home page. There's no need to make any updates to the JSON files and header or footer partials. The only file we need to make changes to is `index.ejs`. Open the file up in a text editor and paste the following code to get started:

```
<div class="container">
  <!-- page title //-->
  <div class="row m-t-3">
    <div class="col-md-12">
      <h1>Blog</h1>
    </div>
  </div>
  <!-- page body //-->
  <div class="row m-t-3">
    <div class="col-md-8">
      <!-- blog posts //-->
    </div>
    <div class="col-md-4">
      <!-- sidebar //-->
    </div>
```

```
</div>
<!-- mailing list //-->
<div class="row m-t-3">
  <div class="col-md-12">
    <!-- form //-->
  </div>
</div>
</div>
```

There are a few different things going on here so let me break them all down for you:

- I don't want a full width layout, so I've decided to use the `.container` class to wrap my templates layout.
- I've created three different rows, one for our page title, one for the page content (blog feed and sidebar), and one for the mailing list section.
- There are some classes on the row `<div>`s that we haven't seen before, like `m-t-3`. I'll cover what those do in the next section.
- Since I want my blog to be readable on devices of all sizes, I decided to use the medium-sized column classes.
- The page title column is set to `.col-md-12`, so it will stretch to 100% of the layout width.
- I've divided the second row, which holds most of our page content, into a two-column grid. The first column will take up 2/3 of the layout width with the `col-md-8` class. The second column, our sidebar, will take up 1/3 of the layout width with the `col-md-4` class.
- Finally, the third row will hold the mailing list and it is also using the `col-md-12` class and will stretch to fill the entire width of the layout.

The basic layout of the grid for our blog home page is now complete. However, let's revisit those new CSS classes from our layout that I added to the row `<div>`s.

Using spacing CSS classes

One of the new utilities that has been added in Bootstrap 4 is spacing classes. These are great as they add an easy, modular way to add extra vertical spacing to your layouts without having to write custom CSS classes for each region. Spacing classes can be applied to both the CSS margin and padding properties. The basic pattern for defining the class is as follows:

{property}-{sides}-{size}

Let's break down how this works in more detail:

- `property` is equal to either `margin` or `padding`.
- `sides` is equal to the side of a box you want to add either `margin` or `padding` to. This is written using a single letter: `t` for top, `b` for bottom, `l` for left, and `r` for right.
- `size` is equal to the amount of margin or padding you want to add. The scale is 0 to 3. Setting the size value to 0 will actually remove any existing margin or padding on an element.

To better understand this concept, let's construct a few spacer classes. Let's say that we want to add some top margin to a row with a size value of 1. Our class would look like this:

`.m-t-1`

Applied to the actual row, `<div>`, the class would look like this:

```
<div class="row m-t-1">
```

For a second example, let's say we want to add some left padding to a div with a value of 2. That combination would look like this when combined with a row `<div>`:

```
<div class="row p-l-2">
```

Are you starting to see how easy it is to add some spacing around your layout and components?



Spacing classes can be used on any type of element, not just the Bootstrap grid.

Now that you understand how these classes work, let's take a look at our blog home page template again. In that case, our `<div>`s looks like this:

```
<div class="row m-t-3">
```

On three sections of the template, I've decided to use these classes and they are all top margin with a size value of three. It's a good idea to try and keep these consistent as it will result in a visually appealing layout when you are done. It also makes it a little easier to do the math when you are setting up your page. Now that we've gone over the entire home page layout, we need to test it.

Testing out the blog home page layout

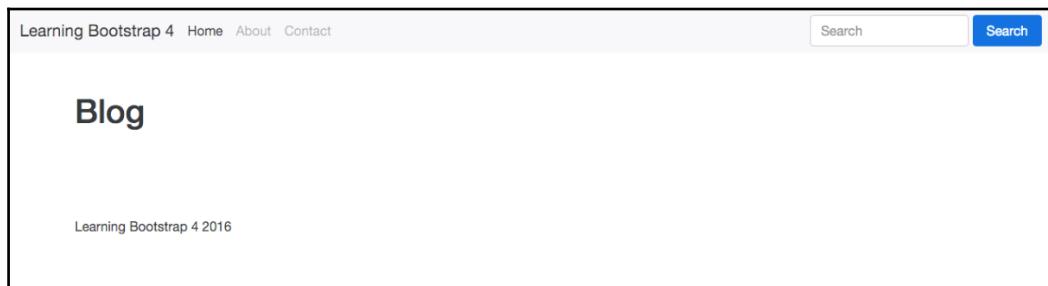
Let's test it out in the browser to make sure it's looking the way we want. Before we can do that we'll need to compile our code with Harp. Open the Terminal back up and navigate to the project directory for this chapter's code that we created. Run the `harp compile` command, here it is again in case you forgot:

```
$ harp compile
```

That should run without any errors; then, we can start-up the web server to view our page. Here's the command again to run the web server:

```
$ harp server
```

Now that the server has launched, head to a web browser and enter `http://localhost:9000` in the URL bar to bring up the blog home page. Here's what your page should look like:



Uh oh, that doesn't look quite right. You can see the page title but we can't see any of our columns. Oh yeah! We need to fill in some content so the columns are revealed. Let's add in some dummy text for demo purposes. In later chapters, I'll get into coding the actual components we want to see on this page. This chapter is just about setting up our layout.

Adding some content

Head back to `index.ejs` in your text editor and let's add some dummy text. Go to the first column of the main content area first and enter something like this:

```
<div class="col-md-8">
  <p>Pellentesque habitant morbi tristique senectus et netus et
  malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae,
  ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam
  egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend
```

```
leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat
wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean
fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci,
sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar
facilisis. Ut felis. Praesent dapibus, neque id cursus faucibus, tortor
neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat
volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus,
metus</p>
</div>
```

If you're looking for a quick way to get filler text in HTML format, visit <http://html-ipsum.com/>.

Next, go to the sidebar column `<div>` and add the same paragraph of text, like so:

```
<div class="col-md-4">
    <p>Pellentesque habitant morbi tristique senectus et netus et
malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae,
ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam
egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend
leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat
wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean
fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci,
sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar
facilisis. Ut felis. Praesent dapibus, neque id cursus faucibus, tortor
neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat
volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus,
metus</p>
</div>
```

Finally, drop down to the mailing list `<div>` and add the same paragraph of content again. It should look like this:

```
<div class="col-md-12">
    <p>Pellentesque habitant morbi tristique senectus et netus et
malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae,
ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam
egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend
leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat
wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean
fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci,
sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar
facilisis. Ut felis. Praesent dapibus, neque id cursus faucibus, tortor
neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat
volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus,
metus</p>
</div>
```

Now that we've added some actual content to our page body, let's recompile the project and launch the web server again:



With Harp, you don't actually have to recompile after every little change you make. You can also make changes to your files while the server is running and they will be picked up by the browser. It's a good habit to compile regularly in case you run into an error on compile. This will make it easier to troubleshoot potential problems.

Once the server is up and running, return to your browser and refresh the page. Now your layout should look like this:

Blog

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci, sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar facilisis. Ut felis. Praesent dapibus, neque id cursus faucibus, tortor neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus, metus

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci, sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar facilisis. Ut felis. Praesent dapibus, neque id cursus faucibus, tortor neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus, metus

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci, sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar facilisis. Ut felis. Praesent dapibus, neque id cursus faucibus, tortor neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus, metus

Learning Bootstrap 4 2016

Yay! We can now see our columns and the dummy text that we just added. The page may not be much to look at right now, but what's important is to verify that your columns are laid out correctly.

What about mobile devices?

We need to consider what will happen to our layout on mobile devices and smaller screen resolutions. I used the medium grid layout class, so any device that is smaller than 720 pixels will have an adjusted layout. Resize your browser window, making it smaller to trigger the media query, and you'll see that all of the columns will be resized to 100% width of the container. Here's what it looks like:

The screenshot shows a blog page with a header containing 'Learning Bootstrap 4' and links for 'Home', 'About', and 'Contact'. A search bar with a 'Search' button is also present. The main content area has a title 'Blog' and two blocks of placeholder text. The text is styled with a sans-serif font and is repeated twice.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci, sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar facilisis. Ut felis. Praesent dapibus, neque id cursus faucibus, tortor neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus, metus

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci, sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar facilisis. Ut felis. Praesent dapibus, neque id cursus faucibus, tortor neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus, metus

I'm going to keep our blog layout pretty minimal so I'm okay with this layout. In this format, the sidebar will slide in under the main blog feed of posts. I'm actually not that crazy about this design, so I'm just going to hide the sidebar altogether when you view the blog on a smaller device.

Using responsive utility classes

Responsive utility classes will allow you to selectively hide `<div>`s or components based on the screen resolution size. This is great for creating a mobile-first web application, because in many cases you'll want to hide some components that don't work well on a phone or tablet. Mobile application design generally means a simpler, more minimal experience so using responsive utility classes will allow you to achieve this. Open up `index.ejs` in a text editor and go down to the sidebar `<div>`, then add the `.hidden-md-down` class to your code:

```
<div class="col-md-4 hidden-md-down">
```

Adding this class will hide the `<div>` from the browser when your screen resolution is smaller than 720 pixels. Make sure your column class, in this case `-md`, matches the hidden class. Now, if you shrink your web browser down again, you'll notice that the sidebar `<div>` will disappear.

There are a number of responsive utility classes you can use in your projects. There is a `-down` version for each of the column class names. You can also use a `-up` version if you wish to hide something when viewing at a larger resolution. Some examples of that class are `.hidden-lg-up` or `.hidden-xl-up`. To learn more about responsive utility classes, check out the page in the Bootstrap docs at <http://v4-alpha.getbootstrap.com/layout/responsive-utilities/>.

That completes the layout of the grid for our blog home page. Before we move onto learning about content classes in Bootstrap 4, let's set up the layout grid for the other pages we'll be building for the blog.

Coding the additional blog project page grids

Before we create new templates for our contact and single blog post pages, we need to update some of the Harp project files. Let's update those files, then we'll move onto the page templates.

Updating `_data.json` for our new pages

Remember a couple chapters back we learned how to code the `_data.json` file and we created a variable for the page title of each of our templates? We need to update this file for our two new pages by providing the `pageTitle` variable value for each of them. Open up `_data.json` in a text editor; you can find the file in the root of your blog project directory.

Once you have the file open, insert the following code. The entire file should read as follows:

```
{  
  "index": {  
    "pageTitle": "Home"  
  },  
  "contact": {  
    "pageTitle": "Contact"  
  },  
  "blog-post": {  
    "pageTitle": "Blog Post"  
  }  
}
```

Originally, we only included the `index` file. I've added two more entries, one for the contact page and one for the `blog-post` page. I've entered a value for each page's `pageTitle` variable. It's as simple as that. Save the JSON file and then you can close it.

Creating the new page templates

Now that `_data.json` has been updated, we need to create the actual page template EJS files like we did with `index`. In your text editor, create two new files and save them as `contact.ejs` and `blog-post.ejs`. For now, just leave them blank and we'll start to fill them in the next steps. The templates are now set up and ready to be coded. For now, both of our new pages will use the same `_layout.ejs` file as the `index` file, so there is no need to create any more layouts. Let's start by coding the contact page template.

Coding the contact page template

Open up the `contact.ejs` file you just created in your text editor. Let's start the template by setting up our page title. Enter the following code into the file:

```
<div class="container">  
  <!-- page title //-->  
  <div class="row m-t-3">  
    <div class="col-lg-12">  
      <h1>Contact</h1>  
    </div>  
  </div>  
</div>
```

Let's breakdown what's happening here in the code:

- I've opened up the file with a `<div>` with a `.container` class on it.
- Next I added `.row` `<div>` and I've added the same `m-t-3` spacing classes so it matches the blog home page.
- I've added a column `<div>` with a class of `.col-md-12`. Since this is our page title, we want it to stretch to the width of our layout.
- Finally, I've added an `<h1>` tag with our contact page title.

Adding the contact page body

Next let's insert our grid layout for the body of the contact page. Following the page title code, insert the following grid code:

```
<!-- page body //-->
<div class="row m-t-3">
  <div class="col-md-12">
    <p>Pellentesque habitant morbi tristique senectus et netus et
malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae,
ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam
egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend
leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat
wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean
fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci,
sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar
facilisis. Ut felis. Praesent dapibus, neque id cursus faucibus, tortor
neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat
volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus,
metus</p>
  </div>
</div>
```

Let's review the code for the page body:

- I've opened up another row `<div>` for the page body. It also has the same `m-t-3` spacing class on it for consistent vertical spacing.
- I've used the `col-md-12` column class again because the contact page layout will fill the whole width of our container.
- I've added some filler text for now so that we can verify that the page is laid out properly.

Before we finish, let's add one more row for our mailing list section. I'd like this to be available on every page of our blog. The grid code for this section will be identical to the markup we did for the page body. Here's what it looks like, for reference:

```
<!-- mailing list //-->
<div class="row m-t-3">
  <div class="col-md-12">
    <p>Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci, sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar facilisis. Ut felis. Praesent dapibus, neque id cursus faucibus, tortor neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus, metus</p>
  </div>
</div>
```

Since this code is identical to the page body, I won't bother breaking it down again. Our layout for the contact page is now complete. Make sure you save the file and let's test it before we move onto the blog post page.

Open your Terminal back up and navigate to the root directory of the blog project. Once there, run the `harp compile` command and then the Harp server command to launch the local web server. Open your web browser and enter the following URL to preview your page: `http://localhost:9000/contact.html`.

Your contact page should load up and you should see a page title and two rows of filler text. Here's what it should look like:

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci, sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar facilisis. Ut felis. Praesent dapibus, neque id cursus faucibus, tortor neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus, metus

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci, sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar facilisis. Ut felis. Praesent dapibus, neque id cursus faucibus, tortor neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus, metus

Learning Bootstrap 4 2016

Our contact page grid is now complete. Before we move onto creating the blog post template, let's take a look at all the code for the contact template:

```
<div class="container">
  <!-- page title //-->
  <div class="row m-t-3">
    <div class="col-md-12">
      <h1>Contact</h1>
    </div>
  </div>
  <!-- page body //-->
  <div class="row m-t-3">
    <div class="col-md-12">
      <p>Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci, sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar facilisis. Ut felis. Praesent dapibus, neque id cursus faucibus, tortor neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus, metus</p>
    </div>
  </div>
  <!-- mailing list //-->
  <div class="row m-t-3">
```

```
<div class="col-md-12">
    <p>Pellentesque habitant morbi tristique senectus et netus et
malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae,
ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam
egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend
leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat
wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean
fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci,
sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar
facilisis. Ut felis. Praesent dapibus, neque id cursus faucibus, tortor
neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat
volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus,
metus</p>
    </div>
</div>
</div>
```

Coding the blog post template

Head back to your text editor and open the file `blog-post.ejs` that you previously created. Like our contact page template, let's start by first setting up the page title section of code. Enter the following code into the blog post template file:

```
<div class="container">
    <!-- page title //-->
    <div class="row m-t-2 m-b-2">
        <div class="col-lg-12">
            <h1>Post Title</h1>
        </div>
    </div>
</div>
```

As you can see, this code is almost identical to the contact page. There are two small differences that I will point out for you:

- I've changed up the spacing classes on the row `<div>`. In a future chapter, we are going to add some different components around the page title, so I've altered the vertical spacing to allow for them. I'm using the same margin top spacer but I've only set it to a value of 2. I've added a second margin bottom spacer with a value of 2 with the `.m-b-2` class. Switching the `-t` to a `-b` will apply a bottom margin instead of a top margin.
- I've changed the page title to `Post Title` in the `<h1>` tag.

Adding the blog post feature

The body of our blog post will have some different elements compared to the blog home template. After the page title, I'm going to insert a feature section that will be used for an image or carousel in a future chapter. For now, let's just lay in the grid column and some filler text for testing purposes. Enter the following code after the page title section:

```
<!-- feature //-->
<div class="row">
  <div class="col-md-12">
    <p>Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci, sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar facilisis. Ut felis. Praesent dapibus, neque id cursus faucibus, tortor neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus, metus</p>
  </div>
</div>
```

This is a very simple section. Notice the `row` `<div>` doesn't have a spacer class on it, since we added the bottom margin to the page title section. I've inserted a full-width `col-md-12` column class so the feature can stretch to the width of the layout.

Adding the blog post body

Now that we've added the blog post feature section, let's add the actual body part of the template. This section will use the same layout as our blog home page. It will be a two-column layout, the first being 2/3 wide, and the sidebar being 1/3 of the layout. Insert the following code after the feature section:

```
<!-- page body //-->
<div class="row m-t-2">
  <div class="col-md-8">
    <p>Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci,
```

```
sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar facilisis. Ut felis. Praesent dapibus, neque id cursus faucibus, tortor neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus, metus</p>
</div>
<!-- sidebar //-->
<div class="col-md-4 hidden-md-down">
<p>sidebar</p>
<p>Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci, sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar facilisis. Ut felis. Praesent dapibus, neque id cursus faucibus, tortor neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus, metus</p>
</div>
</div>
```

Let's break down what's happening here in the code:

- The row `<div>` has a `m-t-2` spacer class added on to provide some vertical spacing
- I'm using the same `col-md-8` and `col-md-4` column classes to set up the layout
- I've also used the `hidden-md-4` class on the sidebar `<div>` so that this section will not be visible on smaller resolution devices
- Finally, I added some temporary filler text for testing purposes

Converting the mailing list section to a partial

As I mentioned earlier, I would like the mailing list section to appear on all pages of the blog. Since this is the case, it would make more sense to make this chunk of code a partial that can be included in each template. It saves us having to reinsert this snippet in every one of our page templates.

From your text editor, create a new file called `_mailing-list.ejs` and save it to the `partial` directory in the root of your blog project. Once the file is saved, insert the following code into it:

```
<div class="row m-t-3">
  <div class="col-md-12">
    <p>Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci, sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar facilisis. Ut felis. Praesent dapibus, neque id cursus faucibus, tortor neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus, metus</p>
  </div>
</div>
```

Now go back to the blog post template file and insert the following line of code where the mailing list section should appear:

```
<%- partial("partial/_mailing-list") %>
```

Remember to do the same thing for the index and contact template. Delete the hardcoded mailing list and replace it with the preceding partial line.

That concludes the setup of the blog post template. Let's test it out before we move onto the next chapter, to make sure the new mailing list partial is working properly. Return to the Terminal and compile your project from the root directory. Run the Harp server command, then visit the following URL: <http://localhost:9000/blog-post.html>.

If all went as planned, your blog post page should look like this:

The screenshot shows a blog post page with the following structure:

- Header:** A navigation bar with links to "Learning Bootstrap 4", "Home", "About", and "Contact". To the right is a search input field with a placeholder "Search" and a blue "Search" button.
- Title:** A large, bold heading "Post Title".
- Main Content:** Two blocks of placeholder text (Lorem ipsum) under the heading "Post Title".
- Sidebar:** A vertical sidebar on the right side of the page containing another block of placeholder text.
- Footer:** At the bottom of the page, there is a footer section containing the text "Learning Bootstrap 4 2016".

Make sure you don't forget to test the index and contact page templates in your browser to make sure the mailing list partial is working properly. That concludes the design layout for the blog post template. All of our templates are now ready to go, but before we move onto the next chapter on content components, let's review what we've learned.

Summary

This chapter has been a detailed explanation of the Bootstrap layout grid, how to use it, and how to build a sample project. We started out by learning the basics of the Bootstrap container, container-fluid, and row classes. Next, we moved onto learning the differences between all the Bootstrap column classes. Following the columns, we covered some more advanced topics, like offsetting columns, spacing, and responsive utilities. Once you had a solid understanding of the grid, we coded up the remaining page layouts that we'll need for the rest of the book. Now that we have everything set up, we'll start to drop some real content into the blog using Bootstrap content classes.

5

Working with Content

Content components in Bootstrap 4 are reserved for the most commonly used HTML elements such as images, tables, typography, and more. In this chapter, I'll teach you how to use all the building blocks of HTML that you'll need to build any type of website or web application. We'll start with a quick overview of each component and then we'll build it into our blog project. Bootstrap 4 comes with a brand new CSS reset called Reboot. It builds on top of `Normalize.css` to make your site look even better out of the box. Before we jump in, let's review some Reboot basics when dealing with content components in Bootstrap 4.

Reboot defaults and basics

Let's start this chapter by reviewing the basics of Reboot when using content components in Bootstrap. One of the main changes for content components in Bootstrap 4 is the switch from `em` to `rem` units of measure. `rem` is short for **root em** and is a little bit different from a regular `em` unit. `em` is a relative unit to the font-size of the parent element it is contained within. This can cause a compounding issue in your code that can be difficult to deal with when you have a highly nested set of selectors. The `rem` unit is not relative to its parent, it is relative to the root or HTML element. This makes it much easier to determine the actual size of text or other content components that will appear on the screen.

The `box-sizing` property is globally set to `border-box` on every element. This is good because it ensures that the declared width of an element doesn't exceed its size due to excess margins or padding.

The base `font-size` for Bootstrap 4 is `16px` and it is declared on the `html` element. On the `body` element, the `font-size` is set to `1rem` for easy responsive type-scaling when using media queries.

There are also global `font-family` and `line-height` values set on the `body` tag for consistency through all components. By default, the `background-color` is set to `#fff` or white on the `body` selector.

Headings and paragraphs

There are no major changes to the styles for headings and paragraphs in Bootstrap 4. All heading elements have been reset to have their `top-margin` removed. Headings have a `margin-bottom` value of `0.5rem`, while paragraphs have a `margin-bottom` value of `1rem`.

Lists

The list component comes in three variations: ``, ``, and `<dl>`. Each list type has had its `top-margin` removed and has a `bottom-margin` of `1rem`. If you are nesting lists inside one another, there is no `bottom-margin` at all.

Preformatted text

This typography style is used for displaying blocks of code on a website using the `<pre>` tag. Like the previous components, its `top-margin` has been removed and it has a `bottom margin` of `1rem`.

Tables

The table component has been adjusted slightly to ensure consistent text alignment in all cells. The styles for the `<caption>` tag have also been adjusted a bit for better legibility.

Forms

The form component is much simpler in Bootstrap 4. Much of the default styling has been removed to make the component easier to use and customize. Here are some of the highlights you should be aware of:

- Most of the styles have been removed on the `<fieldset>` tag. The borders, padding, and margin are no longer there.
- The `<legend>` tag has been simplified and is much more minimal in look now.

- The `<label>` tag is now set to display: `inline-block` to allow margins to be added.
- Default margins have been removed from the following tags: `<input>`, `<select>`, `<textarea>`, and `<button>`.
- `<textarea>`s can now only be resized vertically. They can't be resized horizontally, which often breaks page layouts.

That covers the key elements you need to be aware of with Reboot. If you're interested in learning more, please check out the docs at <http://v4-alpha.getbootstrap.com/content/reboot/>.

Now that we've reviewed the Reboot CSS reset, it's time to actually start covering the content components and adding them to our blog project.



Content classes in Bootstrap 4 are not that different from version 3. If you are fluent in Bootstrap 3, you may want to jump ahead to the next chapter at this point.

Learning to use typography

In Bootstrap 4, there are no major changes with the core typographic HTML tags. Header tags and their supporting CSS classes still work as they always have. However, there are some new utility classes you can use with some type tags to provide further variations for things like headers and titles. Later on in the book we'll be using a number of type tags and styles in our blog project. A couple of quick examples would be header tags for page and post titles, and lists for a number of different components. Let's start by reviewing the new display heading classes in Bootstrap 4.

Using display headings

Regular header tags work great in the flow of a page and are key for setting up the hierarchy of an article. For a landing page or other display-type templates, you may require additional header styles. This is where you can use the new display heading classes to create slightly larger titles with some different styling. There are four different levels of display headings you can use and the markups to render them are as follows:

```
<h1 class="display-1">Display 1</h1>
<h1 class="display-2">Display 2</h1>
<h1 class="display-3">Display 3</h1>
```

```
<h1 class="display-4">Display 4</h1>
```

Keep in mind you can apply these classes to any header tag you like. `display-1` will be the largest and the headers will shrink as you increase their size. For example, `display-4` would be the smallest of the options. Here's what the headers will look like when rendered in the browser:



Keep in mind, you can apply these classes to any header tag you like. `display-1` will be the largest and the headers will shrink as you increase their size. For example, `display-4` would be the smallest of the options.

Customizing headings

You may want to add some additional context to your headers and you can easily do this with some included Bootstrap 4 utility classes. By using a contextual text class, you can tag on a description to a heading like this:

```
<h3>  
  This is the main title  
  <small class="text-muted">this is a description</small>  
</h3>
```

As you can see, I've added a class of `text-muted` to a `<small>` tag that is nested within my header tag. This will style the descriptive part of the text a bit differently, which creates a nice looking effect:

This is the main title this is a description

Using the lead class

Another utility text class that has been added to Bootstrap 4 is the `lead` class. This class is used if you want to make a paragraph of text stand out. It will increase the font size by 25% and set the font-weight of the text to light or 300. It's easy to add, as the following code shows:

```
<p class="lead">  
here's some text with the .lead class to make this paragraph look a bit  
different and standout.  
</p>
```

The output of the preceding code will look like this:

here's some text with the .lead class to make this paragraph look a bit different and standout.

As you can see, this gives the text a unique look. This would be good to use as the first paragraph in a blog post or perhaps to call out some text at the top of a landing page.

Working with lists

Bootstrap 4 comes with a number of list options out of the box. These CSS classes can be applied to the ``, ``, or `<dl>` tags to generate some styling. Let's start with the unstyled list.

Coding an unstyled list

In some cases, you may want to remove the default bullets or numbers that come with ordered or unordered lists. This can be useful when creating a navigation, or perhaps you just want to create a list of items without bullet points. You can do this by using the `list-unstyled` class on the wrapping list tag. Here's an example of a basic unstyled, unordered list:

```
<ul class="list-unstyled">  
  <li>item</li>  
  <li>item</li>  
  <li>item</li>  
  <li>item</li>  
  <li>item</li>  
</ul>
```

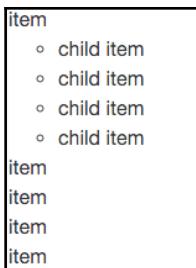
This will produce a list with no bullet points that will look like this:



We can also nest additional lists inside if we want to create multi-level, indented lists. However, keep in mind that the `list-unstyled` class will only work on the first level of your list. Any nested additional lists will have their bullets or numbers. The code for this variation would look something like this:

```
<ul class="list-unstyled">
  <li>item
    <ul>
      <li>child item</li>
      <li>child item</li>
      <li>child item</li>
      <li>child item</li>
    </ul>
  </li>
  <li>item</li>
  <li>item</li>
  <li>item</li>
  <li>item</li>
</ul>
```

The preceding variation will look like the following output:



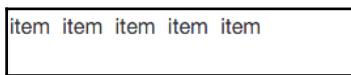
Now, if we check out this code sample in a browser, you'll notice you will see the bullet points for the child list that is nested within the parent.

Creating inline lists

The unstyled list is probably the one you will use the most. The next most useful class is `list-inline`, which will line up each `` in a horizontal line. This is very useful for creating navigations or sub-navigations in a website or application. The code for this list is almost the same as the last, but we change the class name to `list-inline`. We also need to add a class of `list-inline-item` to each `` tag. This is a new change for Bootstrap 4, so make sure you don't miss it in the following code:

```
<ul class="list-inline">
  <li class="list-inline-item">item</li>
  <li class="list-inline-item">item</li>
  <li class="list-inline-item">item</li>
  <li class="list-inline-item">item</li>
  <li class="list-inline-item">item</li>
</ul>
```

As I mentioned, the code is similar to the unstyled list, with a few changes. Here's what it will look like when rendered in the browser:



I think you can see how this would be a lightweight way to set up a horizontal navigation for your project. Let's move onto the last list type, which is a description list.

Using description lists

A description list allows you to create a horizontal display for terms and descriptions. Let's take a look at a basic list's code and then break it down:

```
<dl class="dl-horizontal">
  <dt class="col-sm-3">term 1</dt>
  <dd class="col-sm-9">this is a description</dd>

  <dt class="col-sm-3">term 2</dt>
  <dd class="col-sm-9">this is a different description</dd>

  <dt class="col-sm-3 text-truncate">this is a really long term name</dt>
  <dd class="col-sm-9">this is one last description</dd>
</dl>
```

There are a few things going on here that you need to be aware of, so let me break them all down for you:

- First you start a description list using the `<dl>` tag. It requires a class of `dl-horizontal` to trigger the list component styles.
- Each row is made up of a `<dt>` and `<dd>` tag. `<dt>` stands for term, while `<dd>` stands for description. Each tag should take a column class and is flexible, depending on how you want to lay out your list.
- On the third row, you'll notice a class called `text-truncate`. This class will truncate really long terms or text so they don't run outside the width of the column. This is a good technique to use for long chunks of text.

Now that I've explained all the code for the description list, let's see what this sample should look like in the browser:

term 1	this is a description
term 2	this is a different description
this is a really long term	this is one last description

That completes the typography styles that you need to know about in Bootstrap 4. Next, let me teach you what you can do with images in Bootstrap.

How to style images

Bootstrap allows you to do a few useful things with images through the use of CSS classes. These things include: making images responsive, converting images to shapes, and aligning images. In the next section, I'll show you how to apply all these techniques to your images.

Making images responsive

Bootstrap 4 comes with a new responsive image class that is super-handy when developing websites or web-based applications. When applying the class `img-fluid` to an `` tag, it will automatically set the `max-width` of the image to `100%` and the `height` to `auto`. The result will be an image that scales with the size of the device viewport. Here's what the code looks like:

```

```

It's as easy as adding that class to the image to trigger the responsive controls. A word of advice: I would recommend making your images a little bit bigger than the maximum size you think you will need. That way, the image will look good on all screen sizes.

Using image shapes

Bootstrap allows you to apply three different shape styles to images:

- `img-rounded` will add round corners to your image
- `img-circle` will crop your image into a circle
- `img-thumbnail` will add round corners and a border to make the image look like a thumbnail

As with the responsive images, all you need to do is add a single CSS class to the `` tag to apply these styles. The reason you would want to use these classes is to avoid having to actually create these variations in an app such as Photoshop. It's much easier to apply this simple image formatting using code. Here's the code for each variation:

```
  
  

```

Once you've coded that up, it should look like this in the browser:



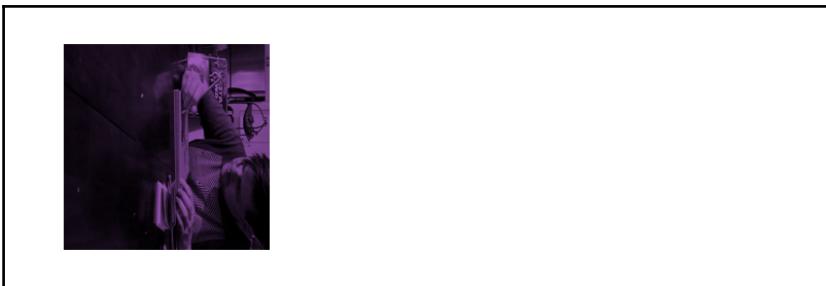
I'm using one of my own images here; you'll need to swap in an image in your code.

Aligning images with CSS

The final Bootstrap classes you can apply to images are the alignment classes. They will allow you to align your image to the left, right, or center of your layout. Like the previous examples, you only need to add a single CSS class to the `` tag to apply the alignment you want. With left and right alignment, you can also provide a column size within the class name. The best policy would be to use the same size as the column the image is contained within. Therefore, if your image is displayed in a column with a class of `col-xs-4`, then use the `-xs` unit in the alignment class name. Here's what the left and right alignment code looks like using the extra small size:

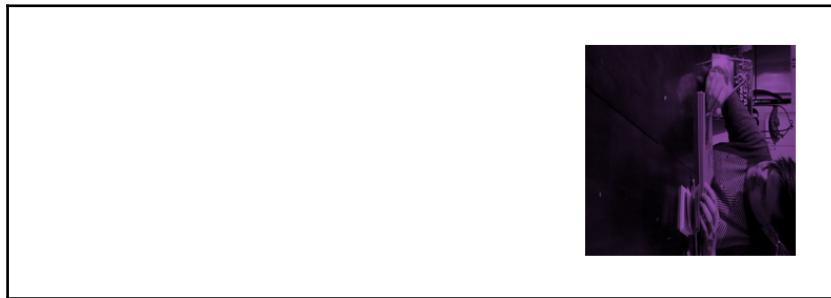
```

```



```

```



The final image alignment class you can use is to center the image in the layout. The class name for this is a little bit different, as you can see here:

```

```



That concludes the section on image classes that you can use in your Bootstrap layouts. Next we will look at writing and rendering tables using Bootstrap 4.

Coding tables

Tables in Bootstrap 4 are largely unchanged from the previous version of the framework. However, there are a few new things, like inverse color table options and responsive tables. Let's start with the basics and we will build in the new features as we go.

Setting up the basic table

The basic table structure in Bootstrap takes advantage of almost all the available HTML table tags. The header is wrapped in `<thead>` and the body `<tbody>` tags. This will allow additional styling as we get into the inverse table layout. For now, let's see how we put together a basic table in Bootstrap:

```
<table class="table">
<thead>
  <tr>
    <th>first name</th>
    <th>last name</th>
    <th>twitter</th>
  </tr>
</thead>
<tbody>
  <tr>
    <td>john</td>
    <td>smtih</td>
    <td>@jsmtih</td>
  </tr>
  <tr>
    <td>steve</td>
```

```
<td>stevens</td>
  <td>@stevens</td>
</tr>
<tr>
  <td>mike</td>
  <td>michaels</td>
  <td>@mandm</td>
</tr>
</tbody>
</table>
```

As you can see, the syntax is fairly straightforward. The only class being applied is the root table class on the `<table>` tag. This needs to be applied to any table variation you are using in Bootstrap. This will produce a table that looks like the following in the browser:

first name	last name	twitter
john	smtih	@jsmtih
steve	stevens	@stevens
mike	michaels	@mandm

As you can see, the syntax is fairly straightforward. The only class being applied is the root table class on the `<table>` tag. This needs to be applied to any table variation you are using in Bootstrap.

Inversing a table

Let me quickly show you one of the new table classes in Bootstrap 4. If we add the class `table-inverse` to the `<table>` tag, the table colors will flip to be a dark background with light text. Here's the code you need to change:

```
<table class="table table-inverse">
  ...
</table>
```

This slight variation in code will produce a table that looks like this:

first name	last name	twitter
john	smtih	@jsmtih
steve	stevens	@stevens
mike	michaels	@mandm

That's a pretty handy trick to know if you need to get a quick variation of the basic table styles going.

Inversing the table header

Perhaps you don't want to inverse the entire table? If that is the case, you can use the `thead-inverse` class on the `<thead>` tag to only inverse that row:

```
<table class="table">
<thead class="thead-inverse">
  ...
</thead>
  ...
</table>
```

If this variation is applied, then your table will look like this:

first name	last name	twitter
john	smtih	@jsmtih
steve	stevens	@stevens
mike	michaels	@mandm

If you're looking for a more subtle design for your project, this approach may be more appealing to you.

Adding striped rows

Although not new to Bootstrap 4, the `table-striped` class is one that I use all the time. Applying this class to the `<table>` tag will add zebra striping to your table, starting with the first row in the body and applying a light grey background color on all the odd numbered rows:

```
<table class="table table-striped">
```

Using this class will produce a table that looks like this:

first name	last name	twitter
john	smtih	@jsmtih
steve	stevens	@stevens
mike	michaels	@mandm

Now our table is starting to come together. With a few classes, we can get an attractive-looking layout. Let's see what else we can do with tables.

Adding borders to a table

Another style that is regularly used is to add borders to your table. This can be done in the same way as stripes. Just change or add another class to the `<table>` tag called `table-bordered`. For this example, I'll remove the stripes and add the borders:

```
<table class="table table-bordered">
```

Now that we've added the borders and taken away the stripes, our table should look like this:

```
<table class="table table-bordered table-striped">
```

first name	last name	twitter
john	smtih	@jsmtih
steve	stevens	@stevens
mike	michaels	@mandm

It's important to know that you can combine the table classes and use more than one. What if you wanted a table with stripes and borders? You can do that easily, by including both of the corresponding classes.

Adding a hover state to rows

It's possible and easy to add a hover state to each of your table rows. To do so, you just need to add the `table-hover` class to the `<table>` tag. When used, if you hover over a row in the table, its background color will change to indicate a state change:

```
<table class="table table-hover">
```

Here I've removed the other table classes to show you the basic hover table option. When viewed in the browser, the table should look like the following when a row is hovered over with the mouse:

first name	last name	twitter
john	smtih	@jsmtih
steve	stevens	@stevens
mike	michaels	@mandm

In some cases you may require a table with smaller text and compressed height. This can be done by adding the `table-sm` class to the `<table>` tag. This will make the look of the table more compact when viewing it:

```
<table class="table table-sm">
```

If you choose to use this class, your table should look like this:

first name	last name	twitter
john	smtih	@jsmtih
steve	stevens	@stevens
mike	michaels	@mandm

Creating smaller tables

That concludes the core table variations that you can apply through a simple CSS class. Before we move on, there are a couple more important points on tables that we should go over.

Color-coating table rows

In some cases, you may want to color the background of a table row in a different color. This can easily be achieved through the use of some included contextual classes. There are five different color variations you can choose from:

- `table-active` is the hover color, light grey by default
- `table-success` is green for a positive action
- `table-info` is blue for an informational highlight
- `table-warning` is yellow to call out something that needs attention
- `table-danger` is red for a negative or error action

The preceding classes can be applied to either a `<tr>` or `<td>` tag. If I apply all of these color variations to a single table, they look like this:

first name	last name	twitter
john	smtih	@jsmtih
steve	stevens	@stevens
mike	michaels	@mandm
steve	stevens	@stevens
mike	michaels	@mandm

As you can see, these classes can be useful for validation or just highlighting a particular row that needs to stand out more.

Making tables responsive

Adding responsiveness to tables has never been very easy to do with CSS. Thankfully, Bootstrap 4 comes with some support built right in that you can easily take advantage of. To make a table responsive, you simply need to wrap a `<div>` around your `<table>` that has a class of `table-responsive` on it:

```
<div class="table-responsive">
  <table class="table">
    ...
  </table>
</div>
```

If you view the table on a viewport that is smaller than 768px, then the table cells will scroll horizontally, so they can all be viewed. If the viewport is larger, you will see no difference in the table compared to a regular one.

Summary

With tables finished off, that brings this chapter to a close. I hope this has been a good introduction to content components in Bootstrap, as well as a good review of what's new for these types of components in Bootstrap 4. To review, we learned about: Reboot, typography, images, and tables. In the next chapter, we'll start to jump into some more complicated components and build them into our blog project.

6

Playing with Components

The real power of Bootstrap lies in the components contained within the framework. In this chapter, we'll go through a number of new and existing components. I'll show you how to use them and then we'll insert them into our sample blog project so you can see them in practice. Let's get right into it by covering one of the most commonly used components, which are buttons.

Using the button component

Buttons are one of the most commonly used components in Bootstrap. In version 4 of Bootstrap, some of the new options for the button component include an outlined variation, toggle states, and button groups with checkboxes and radios. Before we get into that, let's review the basic button options and configuration. Here's a few general points to keep in mind when using buttons:

- No matter what type of button you are creating, it will require the `.btn` CSS class to be included at a minimum
- The `.btn` class can be attached to a number of HTML tags, such as `<button>`, `<a>`, and `<input>`, to render a button
- There are different CSS classes for creating different size and color buttons

Basic button examples

Before we move on to more advanced configuration, let's cover the basics of creating Bootstrap buttons. If you aren't new to Bootstrap, you may want to skip this section. Bootstrap comes with six different button color options out of the box. Here's a breakdown of their names and when to use them:

- **Primary:** The main button used on your website. It is blue by default.
- **Secondary:** The alternate or secondary button used in your website. It is white by default.
- **Success:** Used for positive-based actions. It is green by default.
- **Info:** Used for informational buttons. It is a light blue by default.
- **Warning:** Used for warning-based actions. It is yellow by default.
- **Danger:** Used for error-based actions. It is red by default.

Now that I've explained all the button variations, let's check out the code for a button:

```
<button type="button" class="btn btn-primary">Primary</button>
```

As you can see, I'm using the `<button>` tag and I've added a couple of CSS classes to it. The first is the `.btn` class, which I mentioned you need to include on all buttons. The second is the `.btn-primary` class, which indicates that you want to use the **Primary** button variation. If you want to use a different button style, you simply change up that second class to use the corresponding keyword. Let's take a look at the code for all of the button variations:

```
<button type="button" class="btn btn-primary">Primary</button>

<button type="button" class="btn btn-secondary">Secondary</button>

<button type="button" class="btn btn-success">Success</button>

<button type="button" class="btn btn-info">Info</button>

<button type="button" class="btn btn-warning">Warning</button>

<button type="button" class="btn btn-danger">Danger</button>

<button type="button" class="btn btn-link">Link</button>
```

It's as easy as that. Note that the last line is a **Link** button option that I haven't talked about. This variation will appear as a text link in the browser, but will act as a button when you click or hover over it. I don't often use this variation so I left it out at first. If you view this code in your browser, you should see the following buttons:



Creating outlined buttons

Starting in Bootstrap 4, they've introduced a new button variation which will produce an outlined button instead of a filled one. To apply this look and feel, you need to change up one of the button classes. Let's take a look at the following code for all variations:

```
<button type="button" class="btn btn-primary-outline">Primary</button>
<button type="button" class="btn btn-secondary-outline">Secondary</button>
<button type="button" class="btn btn-success-outline">Success</button>
<button type="button" class="btn btn-info-outline">Info</button>
<button type="button" class="btn btn-warning-outline">Warning</button>
<button type="button" class="btn btn-danger-outline">Danger</button>
```

As you can see, the class names have changed; here's how they map to each button variation:

- btn-primary-outline
- btn-secondary-outline
- btn-success-outline
- btn-info-outline
- btn-warning-outline
- btn-danger-outline

Basically, you just need to append `-outline` to the default button variation class name. Once you do, your buttons should look like this:



Checkbox and radio buttons

A new feature in Bootstrap 4 is the ability to convert checkboxes and radio buttons into regular buttons. This is really handy from a mobile standpoint because it is much easier to touch a button than it is to check a box or tap a radio button. If you are building a mobile app or responsive website, it would be a good idea to use this component. Let's start by taking a look at the code to generate a group of three checkboxes as a button group:

```
<div class="btn-group" data-toggle="buttons">
  <label class="btn btn-primary active">
    <input type="checkbox" checked autocomplete="off"> checkbox 1
```

```
</label>
<label class="btn btn-primary">
  <input type="checkbox" autocomplete="off"> checkbox 2
</label>
<label class="btn btn-primary">
  <input type="checkbox" autocomplete="off"> checkbox 3
</label>
</div>
```

Let me break down the code and explain what is going on here:

- To generate a button group with checkboxes, you need to wrap the boxes in a `<div>` with a class of `.btn-group`.
- To allow the buttons to toggle on and off, you also need to add the data attribute `data-toggle="buttons"` to the `<div>`.
- Next we need to use the button classes on the `<label>` tag to convert each checkbox into a button. Note that on the first button I'm using the `.active` class, which will make this checkbox toggled on by default. This class is totally optional.
- Your basic checkbox `<input>` tag is nested within the label.

Keep in mind since these are checkboxes, you can toggle multiple options on or off. Here's what the button group should look like when rendered in the browser:



As you can see, this renders a nice-looking button group that is optimized for mobile and desktop. Also, notice how the first checkbox has a different background color as it is currently toggled on because of the `.active` class applied to that label. In the same way that we've created a button group with checkboxes, we can do the same thing with radio buttons.

Creating a radio button group

Creating a radio button group is very similar to the checkboxes. Let's start by checking out the code to generate this different variation:

```
<div class="btn-group" data-toggle="buttons">
  <label class="btn btn-primary active">
    <input type="radio" name="options" id="option1" autocomplete="off">
```

```
checked> radio 1
</label>
<label class="btn btn-primary">
  <input type="radio" name="options" id="option2" autocomplete="off">
radio 2
</label>
<label class="btn btn-primary">
  <input type="radio" name="options" id="option3" autocomplete="off">
radio 3
</label>
</div>
```

Let me explain what's happening here with this code:

- Like the checkboxes, you need to wrap your collection of radio buttons in a `<div>` with the same class and data attribute
- The `<label>` tag and button classes also work the same way
- The only difference is that we are swapping the checkbox `<input>` type for radio buttons

Keep in mind that with radio buttons, only one can be selected at a time. In this case, the first one is selected by default, but you could easily remove that. Here's what the buttons should look like in the browser:



As you can see, the button group is rendered the same way as the checkboxes, but in this case we are using radios. This should be the expected result to optimize your group of radio buttons for mobile and desktop. Next we'll build on what we've learned about button groups, but learn how to use them in other ways.

We'll circle back later in this chapter and actually add the components to our blog project.



Using button groups

If you're new to Bootstrap, button groups are exactly as they sound. They are a group of buttons that are connected horizontally or vertically to look like a single component. Let's take a look at the code to render the most basic version of the component:

```
<div class="btn-group" role="group" aria-label="Basic example">
  <button type="button" class="btn btn-secondary">Left</button>
  <button type="button" class="btn btn-secondary">Middle</button>
  <button type="button" class="btn btn-secondary">Right</button>
</div>
```

As you can see, we have a group of regular button tags surrounded by `<div>` with a class of `.btn-group` on it. At the very least, this is all you need to do to render a button group.

There are a couple of other optional attributes on the `<div>` tag, which are `role` and `aria-label`. If you need to worry about accessibility, then you should include those attributes, otherwise they are optional. One other small change in this code is I've decided to use the `.btn-secondary` class to mix things up a bit with the button styles. Let's take a look at how this will appear in the browser:



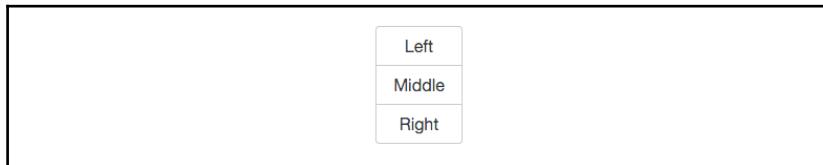
As you can see, we have a single component that is made up of three buttons. This component is commonly used for a secondary navigation, or in a form like I explained in the previous section. If you'd like to display the buttons vertically, that is also possible with a small change.

Creating vertical button groups

If you'd like to arrange the buttons in your group vertically, that is actually quite easy to do. There is no need to change any of the code on the `<button>` tags, you just need to update the CSS class name on the wrapping `<div>` tag. Here's the code you need to change:

```
<div class="btn-group-vertical">
  ...
</div>
```

If you make that alteration to your code, then the same button group will appear like this in the browser:



It would probably have made sense to change the left button label to the top and the right button label to the bottom. However, I left them as they are because I wanted to show you how you can simply shift the alignment of the group by changing one CSS class. That covers the basics of using the button groups component; in the next section, I'll show you how to create button drop-down menus.

Coding a button dropdown

The code to render a button as a dropdown is a little bit more complicated but still fairly easy to get up and running. You'll combine a button tag with `<div>` that has a nested collection of links inside it. Let's take a look at the code required to render a basic drop-down button:

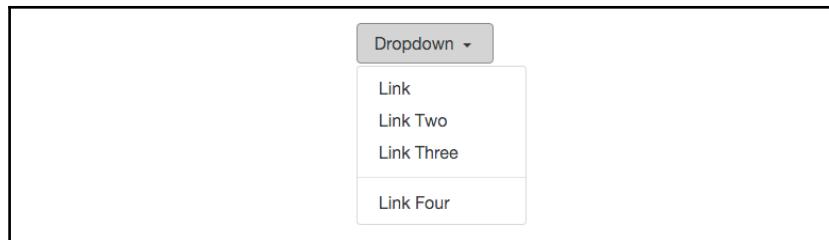
```
<div class="btn-group">
  <button type="button" class="btn btn-secondary dropdown-toggle" data-
  toggle="dropdown" aria-haspopup="true" aria-expanded="false">
    Dropdown
  </button>
  <div class="dropdown-menu">
    <a class="dropdown-item" href="#">Link</a>
    <a class="dropdown-item" href="#">Link Two</a>
    <a class="dropdown-item" href="#">Link Three</a>
    <div class="dropdown-divider"></div>
    <a class="dropdown-item" href="#">Link Four</a>
  </div>
</div>
```

Okay, there are a few things going on here. Let's break them down one by one and explain how the dropdown works:

- The entire component needs to be wrapped in a `<div>` with a class of `.btn-group` on it.

- Next you insert a `<button>` tag with some button CSS classes on it. Like in the previous section, some of the other attributes are optional. However, it is a good idea to include this attribute: `aria-expanded`. This can either be set to `false` or `true` and controls whether the dropdown is open or closed on page load. In most cases, you will want to set this to `false`.
- After the `<button>` tag, insert another `<div>` tag which will hold all the links that appear in the drop-down menu list. Make sure you give this `<div>` a class of `.dropdown-menu`.
- Within the second `<div>` you insert a collection of `<a>` tags, one for each item in your list. Each `<a>` tag requires a class of `.dropdown-item` so that the proper CSS styling is applied.
- You may also want to insert a divider in your drop-down list if you have a large amount of links. This is done by inserting a third `<div>` with a class of `.dropdown-divider` on it.

As I mentioned, this component is a little more complex, but in Bootstrap 4 they have actually simplified it a bit to make it easier to use. Let's take a look at what it should look like in the browser. In the following screenshot, I've showed what the expanded version of the dropdown will look like so you can see the button and the list of links:



As you can see, we have a drop-down button with a list of links nested within it. Keep in mind that if you want to use this component, it does require that you include `jQuery` and `bootstrap.min.js` in your template. There are some other variations of this component you can easily implement, such as the pop-up menu.

Creating a pop-up menu

In some cases, you might want to have your menu pop up above the button instead of below it. You can achieve this by adding one class on the wrapping `<div>` for the component. Check out the code here:

```
<div class="btn-group dropdown">  
  ..  
</div>
```

As you can see, I've added the class `.dropdown` to the `<div>`. This will make the menu appear above the button, and it should look like this:



As you can see, the list appears above the button when it is expanded.

Creating different size drop-down buttons

By adding a single class to the `<button>` tag in the dropdown, you can make the trigger larger or smaller. Let's take a look at the code for the smaller and larger button variations:

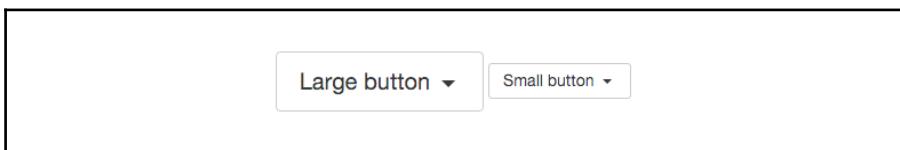
```
<!-- large button //-->  
<div class="btn-group">  
  <button class="btn btn-secondary btn-lg dropdown-toggle" type="button"  
    data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">  
    Large button  
  </button>  
  <div class="dropdown-menu">  
    ...  
  </div>  
</div>  
  
<!-- small button //-->  
<div class="btn-group">  
  <button class="btn btn-secondary btn-sm dropdown-toggle" type="button"  
    data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">  
    Small button  
  </button>  
  <div class="dropdown-menu">  
    ...  
  </div>
```

```
</div>
```

If you find the button tag in the first example, you'll see I've added a class of `.btn-lg` to it. This class will increase the button size to be larger than the default. Take a look at the second chunk of code, find the `<button>` tag again, and you'll see a class of `.btn-sm` on it. This works the same way except the button will now be smaller than the default. Let's see how these buttons will render in the browser.



The `.btn-lg` and `.btn-sm` classes are not exclusive to the button drop-down component. You can use them on any button component variation you like.



That concludes the basics of using the button drop-down component. In the next section, we'll cover a more complicated component, which is forms.

Coding forms in Bootstrap 4

If you are familiar with Bootstrap 3, then you'll notice the CSS form classes are pretty much the same in version 4. The biggest change I see in forms for the new version is that each form group uses a `<fieldset>` tag instead of `<div>`. If you are new to Bootstrap forms, a basic form group is made up of a label and an input. It can also include help text, but that is optional. Let's jump right in by creating a generic form that uses a number of core components.

Setting up a form

At the very least, a form needs to be made up of one input and one button. Let's start with the basics and create a form following those requirements. Here's the code to get you started:

```
<form>
  <fieldset class="form-group">
    <label>Text Label</label>
    <input type="text" class="form-control" placeholder="Enter Text">
```

```
<small class="text-muted">This is some help text.</small>
</fieldset>
<button type="submit" class="btn btn-primary">Submit</button>
</form>
```

Let me explain what is happening here in the code:

- Every form needs to start with a `<form>` tag. However, no special classes are required on this tag.
- I've inserted a `<fieldset>` tag with a class of `.form-group` on it for our single input. This `<fieldset>` pattern will be repeated in the future when you add additional inputs.
- Within the `<fieldset>`, we have a `<label>`. Again, no special CSS classes need to be added to the `<label>`.
- After the label, you need to insert the form `<input>` tag. In this case, I'm using a text input. On this HTML tag, you need to add a class of `.form-control`. All input tags in Bootstrap will require this class. The placeholder text is optional but nice to add for usability.
- In the last line of the `<fieldset>`, I've included a `<small>` tag with a class of `.text-muted`, which will render the text small and light grey. This is the pattern you should use if you want to include some help text with your form input.
- Close the `<fieldset>` tag and then you need to add a `<button>` tag for the form submit button.
- Close the `<form>` and you are done.

After you've finished reviewing the code, fire up your web browser, and your form should look like this:

The image shows a screenshot of a web browser displaying a simple form. The form is contained within a rectangular box. At the top left, there is a label "Text Label". Below the label is an input field with the placeholder text "Enter Text". Underneath the input field is a small text box containing the text "This is some help text.". At the bottom of the form is a blue button with the word "Submit" in white text. The entire form is styled using Bootstrap's CSS framework.

You've successfully coded your first Bootstrap 4 form. Let's continue and I'll explain how to implement other common form components using the latest version of Bootstrap.

Adding a select dropdown

Let's build on our form code by adding a select drop-down menu. Insert the following code after our text input:

```
<fieldset class="form-group">
  <label>Select dropdown</label>
  <select class="form-control">
    <option>one</option>
    <option>two</option>
    <option>three</option>
    <option>four</option>
    <option>five</option>
  </select>
</fieldset>
```

Let's break down the parts of the code you need to be aware of:

- Note that the entire `<select>` is wrapped in a `<fieldset>` with a class of `.form-group`. This pattern should repeat for each type of form input you add.
- On the `<select>` tag, there is a class of `.form-control` that needs to be added.
- Aside from that, you should code the `<select>` as you normally would, following the best HTML syntax practices.

Once you're done, if you view the form in the browser, it should now look like this:

The image shows a screenshot of a web browser displaying a form. The form consists of several elements: a text input field labeled "Text Label" with the placeholder "Enter Text" and some descriptive text below it; a text area labeled "Text Label" containing the placeholder "This is some help text."; a select dropdown menu labeled "Select dropdown" with the value "one" selected; and a blue "Submit" button at the bottom. The entire form is enclosed in a rectangular border.

That completes the setup for `<select>` dropdowns. Next let's check out the `<textarea>` tag.

Inserting a textarea tag into your form

Moving along to the next input type, let's insert a `<textarea>` tag into our form. After the `<select>` menu, add the following code:

```
<fieldset class="form-group">
  <label>Textarea</label>
  <textarea class="form-control" rows="3"></textarea>
</fieldset>
```

Using this input is fairly simple. Like our other examples, you need to use a `<fieldset>` tag with a CSS class of `.form-group` to wrap the entire thing. On the actual `<textarea>` tag, you need to add the `.form-control` class. That's it; once you're done, your form should now look like this:

The form contains the following elements:

- A text input field labeled "Text Label" with placeholder text "Enter Text".
- A dropdown menu labeled "Select dropdown" with the value "one".
- A large text area labeled "Textarea".
- A blue "Submit" button at the bottom.

Now that the `<textarea>` is complete, let's move on to the file input form field.

Adding a file input form field

Historically, the file input form field has been a tricky one to style with CSS. I'm happy to say that in Bootstrap 4 they've created a new approach that's the best I've seen to date. Let's start by inserting the following code after the `<textarea>` in our form:

```
<fieldset class="form-group">
  <label>File input</label>
  <input type="file" class="form-control-file">
  <small class="text-muted">This is some help text. Supported file types
```

```
are: .png</small>
</fieldset>
```

Again, this form field is constructed in the same manner as the previous ones. However, there is one small change you need to be aware of with the **File input** field. On the `<input>` tag, you need to change the CSS class to `.form-control-file`. There are some specific styles being applied to clean up the look and feel of this form field. Once you're done, your form should look like this:

The form consists of several input fields:

- Text Label**: A text input field with the placeholder "Enter Text". Below it is help text: "This is some help text."
- Select dropdown**: A dropdown menu showing "one".
- Textarea**: A text area with a blank content area.
- File input**: A file upload field with the button "Choose File" and the message "No file chosen". Below it is help text: "This is some help text. Supported file types are: .png".

At the bottom is a blue "Submit" button.

That completes the **File input** field which leaves us with two more basic form field inputs to go over. They are radio buttons and checkboxes. Let's learn how to add them next.

Inserting radio buttons and checkboxes to a form

These fields are pretty similar so I'm going to group them together in their own section. The code for these two fields differs a little bit from the other inputs, as I'll outline now. First, let's insert the following code after the `File input` field in our form:

```
<div class="radio">
  <label>
    <input type="radio" name="optionsRadios" id="optionsRadios1"
value="option1" checked>
      Option 1
    </label>
  </div>
<div class="radio">
  <label>
    <input type="radio" name="optionsRadios" id="optionsRadios2"
value="option2">
      Option 2
    </label>
  </div>

<div class="checkbox">
  <label>
    <input type="checkbox"> Checkbox
  </label>
</div>
```

Let's start by going over the radio button code first, then we'll move on to the checkbox:

- The fields don't use the `<fieldset>` tag as the wrapper. In this case, you should use a `<div>` and give it a class of either `.radio` or `.checkbox`, depending on what type you want to use.
- For these fields, the `<label>` tag will also wrap around the `<input>` tag so that everything is displayed in a horizontal line. We don't want the text label to drop down below the radio button or checkbox.
- You don't need a special class on the `<input>` for either of these fields.

As you can see, the code for these fields is a bit different from what we've learned about the other form inputs. Not to worry, as they are pretty easy to use and there aren't a bunch of CSS classes you have to memorize. One of the nicest changes with forms in Bootstrap 4 is that they require less HTML markup, so are easier to write. Finally, if you view our form in the browser, it should look like this:

The form consists of several input fields:

- Text Label:** A text input field with placeholder "Enter Text" and help text "This is some help text."
- Select dropdown:** A dropdown menu showing the option "one".
- Textarea:** A large text area for entering text.
- File input:** A file selection button with placeholder "Choose File" and help text "No file chosen. This is some help text. Supported file types are: .png".
- Radio buttons:** Two radio buttons labeled "Option 1" and "Option 2", where "Option 1" is checked.
- Checkbox:** A checkbox labeled "Checkbox".

A blue "Submit" button is located at the bottom of the form.

That completes the explanation of all the core form fields that you need to know how to use in Bootstrap 4. Before we move on to some more advanced form fields and variations, why don't we add a form to our blog project?

Adding a form to the blog contact page

I know, I know. I said we would wait till the end of the chapter to build components into the blog project. However, I'm thinking you might like a break from learning and actually add some of what you've learned to your project. Let's go ahead and do just that by filling in a form on the **Contact** page.

Updating your project

Let's start by opening up our project directory and finding the file named `contact.ejs`. Open up that file in your text editor and we are going to add some new form code and remove some filler code. To start, find the body section of the page that is wrapped in the following column `<div>`:

```
<div class="col-md-12">
```

Within that `<div>` is currently some filler text. Remove that text and replace it with the following form code:

```
<form>
  <fieldset class="form-group">
    <label>Email</label>
    <input type="email" class="form-control" placeholder="Enter email">
    <small class="text-muted">We'll never share your email with anyone else.</small>
  </fieldset>
  <fieldset class="form-group">
    <label>Name</label>
    <input type="text" class="form-control" placeholder="Name">
  </fieldset>
  <fieldset class="form-group">
    <label>Message</label>
    <textarea class="form-control" rows="3"></textarea>
  </fieldset>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

I've coded up a basic contact form that you'll commonly see on a blog. It has e-mail, name, and message fields along with a **submit** button. Save your file and then preview your project in a browser. The **Contact** page should now look like this:

The screenshot shows a contact form on a website. At the top, there's a navigation bar with 'Learning Bootstrap 4' and links for 'Home', 'About', and 'Contact'. To the right of the navigation is a search bar with a 'Search' button. Below the navigation, the page title is 'Contact'. The form itself has three fields: 'Email' (with placeholder 'Enter email' and a note 'We'll never share your email with anyone else.'), 'Name' (with placeholder 'Name'), and 'Message' (a large text area). At the bottom of the form is a blue 'Submit' button.

That concludes the updates to the **Contact** page for now. Later on in the book, we'll add some additional components to this page. Let's jump back into learning about forms in Bootstrap 4 by reviewing some additional form controls.

Additional form fields

Now that we've learned how to build a basic form and added one to our project, let's circle back and talk about some more advanced form fields and variations you can apply with Bootstrap 4. I'm going to start by showing you how to lay out forms in a few different ways.

Creating an inline form

Let's start by learning how to create an inline form. This is a layout you might want to use in the header of a project or perhaps for a login page. In this case, we're going to align the fields and buttons of the form vertically across the page. For this example, let's create a simple login form with the following code:

```
<form class="form-inline">
  <div class="form-group">
    <label>Name</label>
    <input type="text" class="form-control" placeholder="Mike Smith">
  </div>
  <div class="form-group">
    <label>Email</label>
    <input type="email" class="form-control" placeholder="mike@gmail.com">
  </div>
  <button type="submit" class="btn btn-primary">Login</button>
</form>
```

There are a few things going on in this form, so let me explain them for you:

- For inline forms, we need to add a CSS class named `.form-inline` to the `<form>` tag.
- You'll also notice the `<fieldset>` tags have been replaced with `<div>` tags. This is so they can be set to display as `inline-block`, which won't work with a `fieldset`.

Aside from those two differences, the form is coded up the same way as a regular one. Once you're done, your form should look like this in the browser:

```
<form>
  <input type="text" value="Mike Smith" />
  <input type="text" value="mike@gmail.com" />
  <input type="button" value="Login" />
</form>
```

If you're like me, you might find the labels next to the text inputs kind of ugly. The good news is there is an easy way to hide them.

Hiding the labels in an inline form

The reason those labels are there is for accessibility and screen readers. We don't want to remove them altogether from the code, but we can hide them by adding a CSS class named `.sr-only`. This class stands for **screen reader only** and will therefore only show the labels if they are viewed on an accessible screen reader. Here is an example of how to add the CSS class:

```
<label class="sr-only">Name</label>
```

After you apply that CSS class to all the labels in the form, it should now appear like this in the browser:

```
<form>
  <input type="text" value="Mike Smith" />
  <input type="text" value="mike@gmail.com" />
  <input type="button" value="Login" />
</form>
```

That concludes how to make a basic inline form. However, what if you want to include other fields in an inline manner? Let's see how we can add checkboxes and radios.

Adding inline checkboxes and radio buttons

If you'd like to include checkboxes and radio buttons to an inline form you need to make some changes to your code. Let's start by going over the checkbox code. Insert the following code after the last text input in the inline form:

```
<label class="checkbox-inline">
  <input type="checkbox" value="option1"> Remember me?
</label>
```

There are a couple of things here that you need to be aware of:

- First, there is no longer a `<div>` wrapped around the checkbox
- You need to add a class named `.checkbox-inline` to the checkbox's `<label>` tag

Once you do this, save your form and it should look like this in the browser:

A screenshot of a login form. It contains two input fields: one for 'Email' with the placeholder 'mike@gmail.com' and another for 'Password' with the placeholder 'password'. Below these is a 'Remember me?' checkbox followed by the word 'Remember me?'. To the right of the checkbox is a blue 'Login' button.

Now that we've added the checkbox, let's check out an example using radio buttons. Add the following code to your form after the checkbox code:

```
<label class="radio-inline">  
  <input type="radio" name="inlineRadioOptions" id="inlineRadio1"  
  value="option1"> Yes  
</label>  
<label class="radio-inline">  
  <input type="radio" name="inlineRadioOptions" id="inlineRadio2"  
  value="option2"> No  
</label>
```

As you can see, the pattern here is exactly the same. The `<div>` around each radio button has been removed. Instead, there is a CSS class named `.radio-inline` that needs to be added to each radio `<label>` tag. Once you've completed this step, your form should look like this:

A screenshot of a login form. It contains two input fields: one for 'Email' with the placeholder 'mike@gmail.com' and another for 'Password' with the placeholder 'password'. Below these is a 'Remember me?' checkbox followed by the words 'Remember me?'. To the right of the checkbox are two radio buttons labeled 'Yes' and 'No'. To the right of the radio buttons is a blue 'Login' button.

That completes everything you need to know about inline forms. Let's now move on to some more utility-type actions that you can apply to your form fields.

Changing the size of inputs

Bootstrap comes with a few handy utility CSS classes that you can use with form fields to have them appear at different sizes. Along with the default size, you can choose to display your fields in a larger or smaller size. Let's take a look at the code to render all three size

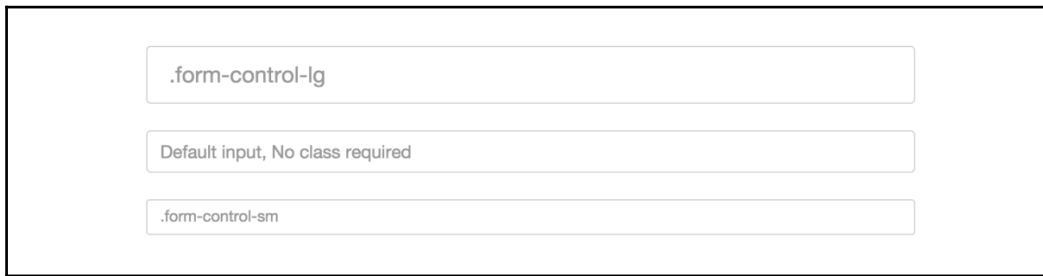
variations:

```
<input class="form-control form-control-lg" type="text" placeholder="form-control-lg">
<input class="form-control" type="text" placeholder="Default input, No class required">
<input class="form-control form-control-sm" type="text" placeholder="form-control-sm">
```

To use the different size inputs, you simply have to add an additional class to the tag:

- For a larger input, use the class `.form-control-lg`
- For a smaller input, use the class `.form-control-sm`
- The default input size requires no extra CSS class

Here's how each version looks in the browser:



As you can see, the larger input is taller and has some additional padding. The smaller input is shorter with reduced padding. These classes only cover the vertical size of an input. Now let's learn how to control the width of inputs.

Controlling the width of form fields

Since Bootstrap is a mobile-first framework, form fields are designed to stretch to fit the width of their column. Therefore, if you are using `.col-md-12` for your column class, the field is going to stretch to the width of the layout. This may not always be what you want, you may only want the input to stretch to half of the width of the layout.

If this is the case, you need to wrap your field in a `<div>` with a column class on it to control the width. Let's check out some example code to get the point across:

```
<div class="col-md-12">
  <input type="text" class="form-control" placeholder="full width">
```

```
</div>
<div class="col-md-6">
  <input type="text" class="form-control" placeholder="half width">
</div>
```

In the preceding code, I've removed some of the labels and other form code to make it easier to see what is going on. Here's a breakdown of what you need to know:

- You need to wrap your input in a `<div>` with a column class on it
- The first input will stretch to the width of the layout because of the `.col-md-12` class
- The second input will only stretch to fill 50% of the layout because of the `.col-md-6` class

Let's take a look at how this will look in the actual browser:



As you can see, the second input only stretches to half of the width. This is how you can control the width of inputs if you don't want them to fill the entire layout of your page. The last thing I'd like to cover when it comes to forms is validation of input fields.

Adding validation to inputs

Bootstrap 4 comes with some powerful yet easy to use validation styles for input fields. Validation styles are used to show things such as errors, warnings, and success states for form fields when you submit the actual form. Let's take a look at the code to render all three validation states:

```
<div class="form-group has-success">
  <label class="form-control-label">Input with success</label>
  <input type="text" class="form-control form-control-success">
</div>
<div class="form-group has-warning">
  <label class="form-control-label">Input with warning</label>
  <input type="text" class="form-control form-control-warning">
</div>
<div class="form-group has-danger">
```

```
<label class="form-control-label">Input with danger</label>
<input type="text" class="form-control form-control-danger">
</div>
```

The markup for each validation variation is very similar to a regular input with the addition of a few CSS classes to apply the proper state look and feel. Let's go over each change you need to be aware of:

- The first input is the success state. The wrapping `<div>` needs to have a class called `.has-success` added to it.
- Each `<label>` tag needs to have a class named `.form-control-label` added to it. This is required to color the label to match the state color.
- The success input requires a class named `.form-control-success`.
- The second input is the warning state. The wrapping `<div>` needs a class named `.has-warning` added to it.
- The warning input also needs a class named `.form-control-warning` added.
- Finally, the last input is the danger or error state. The wrapping `<div>` needs to have a class named `.has-danger` added.
- The danger input also needs a class named `.form-control-danger` added.

Let's take a look at how all these validation inputs should look in the browser:



As you can see, the inputs and labels are colored to match their state. You'll also notice each input has an icon to the right edge of it. These icons are automatically added when you include the required CSS files. There is no need to actually use any images here, which is great. That concludes everything that you need to know about forms in Bootstrap 4. In the next section, I'll teach you about the **Jumbotron** component.

Using the Jumbotron component

If you're new to Bootstrap, you may be asking yourself what the heck is a Jumbotron component. Jumbotron is used to feature a block of content, usually at the top of your page. This is your standard main feature block that you'll see on a number of websites. If you require something more sophisticated than a simple page title, Jumbotron is the component you'll want to use. Let's take a quick look at the code required to create this component:

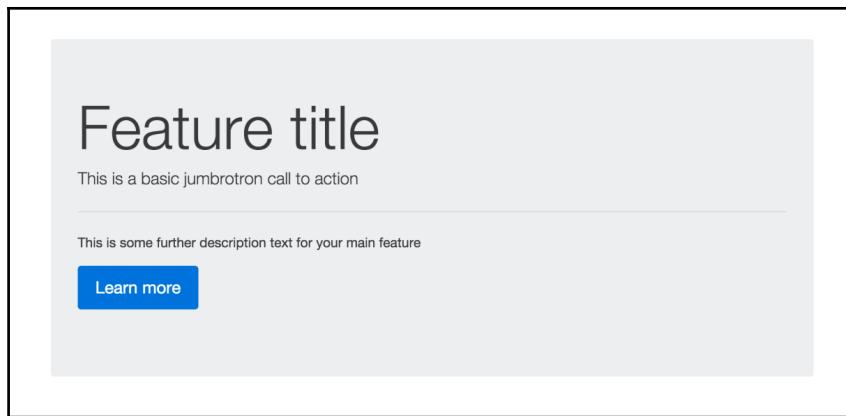
```
<div class="jumbotron">
  <h1 class="display-3">Feature title</h1>
  <p class="lead">This is a basic jumbotron call to action</p>
  <hr class="m-y-2">
  <p>This is some further description text for your main feature</p>
  <p class="lead">
    <a class="btn btn-primary btn-lg" href="#" role="button">Learn more</a>
  </p>
</div>
```

There are some new CSS classes here that we need to review, as well as some existing ones we have already learned about. Let's break down what's happening in the code:

- The Jumbotron component is based off a `<div>` with a CSS class named `.jumbotron`. Within this `<div>`, you can pretty much use whatever text formatting tags you like. However, there are a few basics you should include to make it look good.
- The first tag you'll see is the `<h1>` with a class of `.display-3` on it. Since the Jumbotron is more of a “display” component, you'll want to beef up the size of your `<h1>` by using the optional class we learned about earlier in the book.
- Next, you'll see a simple `<p>` tag for the feature's tagline. On that tag, there is a class named `.lead`. This class increases the base font size by 25% and sets the `font-weight` to 300 which is a lighter weight. Again, this gives the Jumbotron component more of a “feature” like look and feel.
- After the tagline text, you'll see an `<hr>` tag with a class of `.m-y-2` on it. If you remember, this is a utility spacing class. The `-y` in this case will add a margin above and below the `<hr>` tag.

- Next we have another `<p>` tag with some additional descriptive text in it.
- Finally, we have a `<button>` wrapped in a `<p>` tag so that there is a conclusion to the call to action in the Jumbotron block. Note that the user of the `.btn-lg` class will produce a larger-sized button.

After you've coded up your Jumbotron component, it should look like this in the browser:



By default, the Jumbotron component will stretch to fit the width of the column it is contained within. In most cases, you'll likely want it to span the entire width of your page. However, in some cases, you might want a Jumbotron to stretch from one edge of the browser to the other without any horizontal padding on it. If this is the case, you need to add the `.jumbotron-fluid` class to the main `<div>` and make sure it is outside of a Bootstrap `.container`. Let's take a look at the following code to see what I mean:

```
<div class="jumbotron jumbotron-fluid">
  <div class="container">
    <h1 class="display-3">Feature title</h1>
    <p class="lead">This is a basic jumbotron call to action</p>
  </div>
</div>
```

As you can see, the `.container` `<div>` is now inside of the Jumbotron `<div>`. This is how you remove the horizontal padding on the section. Once completed, it should look like this in the browser:



That concludes the use of the Jumbotron component in Bootstrap 4. Next let's move on to learning how to use the Label component.

Adding the Label component

The Label component is used to add context to different types of content. A good example would be notifications on an application. You might use a label to indicate how many unread emails there are in an email app. Another would be to insert a warning tag next to an item in a table or list. Like buttons, labels are available in a number of color variations to meet your needs in your project. Let's take a look at the code to render the basic label options:

```
<span class="label label-default">Default</span>
<span class="label label-primary">Primary</span>
<span class="label label-success">Success</span>
<span class="label label-info">Info</span>
<span class="label label-warning">Warning</span>
<span class="label label-danger">Danger</span>
```

You'll likely notice some similarities here with the Button component CSS classes. When using a label, you should use the `` tag as your base for the component. Here are some more important facts when using this component:

- Every variation of the Label component requires the use of the `.label` class on the `` tag
- The **Default** label uses the `.label-default` class and is grey
- The **Primary** label uses the `.label-primary` class and is blue
- The **Success** label uses the `.label-success` class and is green
- The **Info** label uses the `.label-info` class and is light blue
- The **Warning** label uses the `.label-warning` class and is yellow
- Finally, the **Danger** label uses the `.label-danger` class and is red

Once you've coded that up, it should look like this in your browser:



Default Primary Success Info Warning Danger

By default, labels will be rectangular with slightly rounder corners. If you'd like to display them in pill format, you can do so by adding the `.label-pill` class to the `` tag. Here's an example to see what I mean:

```
<span class="label label-pill label-default">Default</span>
<span class="label label-pill label-primary">Primary</span>
<span class="label label-pill label-success">Success</span>
<span class="label label-pill label-info">Info</span>
<span class="label label-pill label-warning">Warning</span>
<span class="label label-pill label-danger">Danger</span>
```

If you add that class to your labels, they should look like this in the browser:



Default Primary Success Info Warning Danger

That concludes the Label component in Bootstrap 4. Next, I'll teach you how to use the Alerts component.

Using the Alerts component

The Alerts component in Bootstrap provides contextual messages for typical uses, such as validation and general information, that need to stand out more. Like our previous components, it comes in a few different variations depending on your needs. Let's start by looking at the basic code required to render the different alert options:

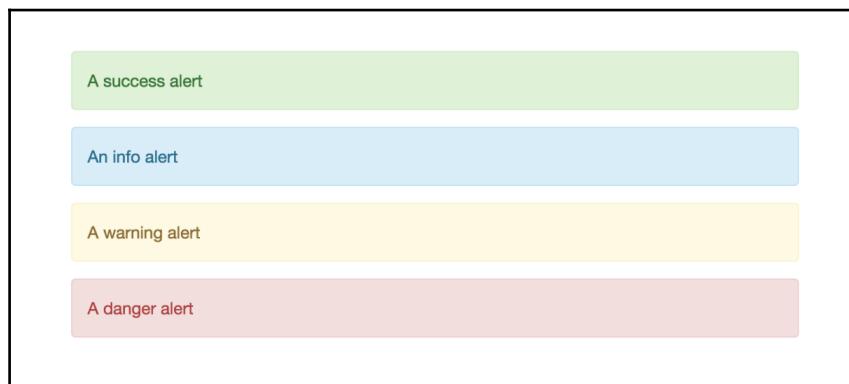
```
<div class="alert alert-success" role="alert">
  A success alert
</div>
<div class="alert alert-info" role="alert">
  An info alert
</div>
<div class="alert alert-warning" role="alert">
  A warning alert
</div>
```

```
<div class="alert alert-danger" role="alert">  
  A danger alert  
</div>
```

The classes used to create an alert can be added to any block element, but for demo purposes we'll implement them using `<div>` tags. Here are the key points you need to know:

- Any instance of the Alert component will require the use of the `.alert` CSS class on the `<div>` tag
- You also need a second CSS class to indicate which version of the alert you want to use
- The **Success** alert uses the class `.alert-success` and is green
- The **Info** alert uses the class `.alert-info` and is blue
- The **Warning** alert uses the class `.alert-warning` and is yellow
- The **Danger** alert uses the class `.alert-danger` and is red

Once you've set up the code for those alerts, they should look like this in the browser:



That was a basic example of using Alerts. There are some additional things you can do to extend this component such as adding a dismiss button.

Adding a dismiss button to alerts

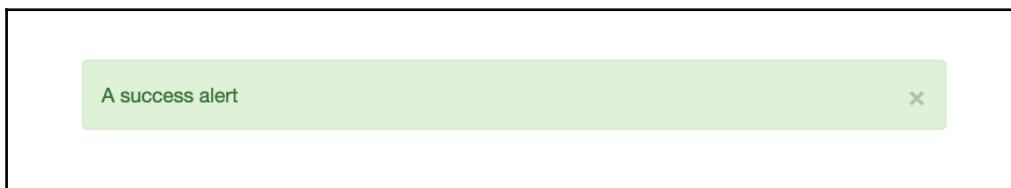
If you want to make your alert bar dismissible, you can add a button to do this. To include the link, update the code for your bar, as follows:

```
<div class="alert alert-success" role="alert">
    <button type="button" class="close" data-dismiss="alert" aria-
label="Close">
        <span aria-hidden="true">&times;</span>
    </button>
    A success alert
</div>
```

The previous Alert bar code doesn't change, but you do need to add a button before the alert message:

- The `<button>` requires a class named `.close` to appear
- You'll also need the `data-dismiss` attribute to be included with a value of `alert`
- The `×` code will be rendered as an **X** in the browser

Once you've added the new code, your alert bar should look like this:



Now your alert bar has a dismissible **X** button that can be triggered to close when you implement the functionality of the component in your app or website. That completes the Alert component in Bootstrap 4. In the next section, I'll teach you about the best new component in version 4, which is Cards.

Using Cards for layout

In my opinion, the best new feature in Bootstrap 4 is the new Card component. If you're unfamiliar with Cards, they were made popular with the release of Google Material Design. They are a mobile first content container that works well for phones, tablets, and the desktop.

We'll be using the Card component heavily in our blog project so let's jump right in and start learning how to use them. Check out the following code to learn how to render a basic card:

```
<div class="card">
```

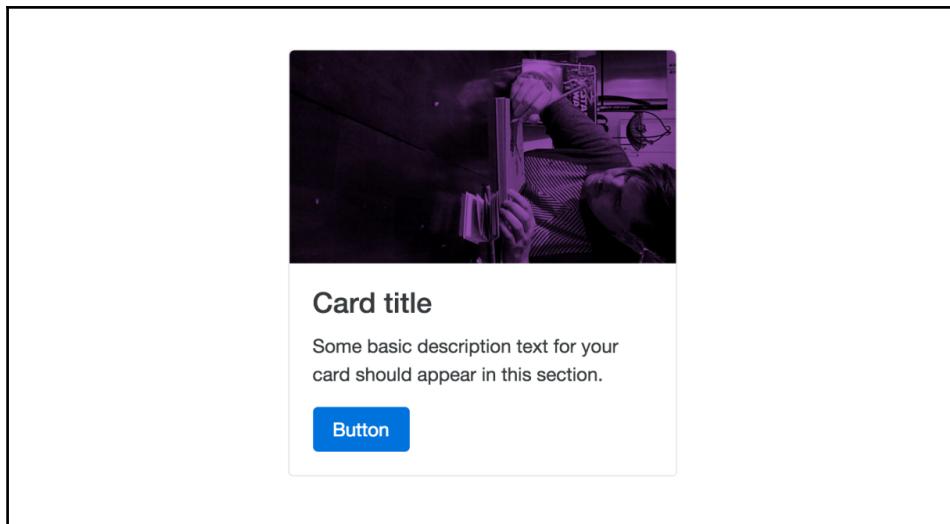
```

<div class="card-block">
  <h4 class="card-title">Card title</h4>
  <p class="card-text">Some basic description text for your card should
appear in this section.</p>
  <a href="#" class="btn btn-primary">Button</a>
</div>
</div>
```

There are a number of new CSS classes you need to be aware of here, so let's go through them one by one:

- Any instance of the Card component must use a `<div>` tag with a CSS class named `.card` on it.
- If you wish to include an image inside your card, it comes next. The image requires a class named `.card-img-top` to display the image at the top of the card. Although not required, I would also recommend adding the class `.img-fluid` to your image. This will make the image responsive so that it will automatically resize to match the width of your card.
- After the image, you need to start a new `<div>` with a CSS class named `.card-block`. This part of the Card will contain the actual textual content.
- The first thing your card should have is a title. Use an `<h4>` tag with a CSS class of `.card-title` for this section.
- Next, you can insert a paragraph of text with a `<p>` tag and a class of `.card-text`. If you choose to have multiple paragraphs, make sure each one uses that same class name.
- Finally, I've inserted a primary `<button>` so the user has something to click on to view the full piece of content.

After you've finished coding this up, it should appear like this in your browser. Note for demo purposes, I've included an image of my own so you can see how it works. You'll need to provide your own images for your projects:



As you can see, this will render a neat-looking little content component that you can use in many different ways. Let's move on by learning some other ways that you can customize the Card component.

Moving the Card title

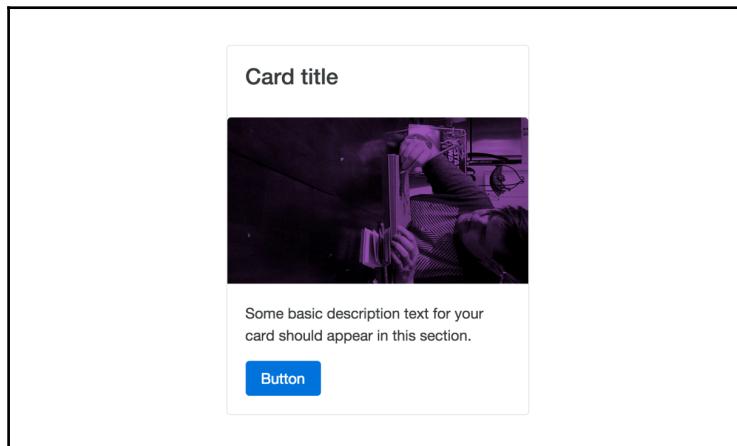
Perhaps you want to move the title of your card above the image? This is actually really easy to do, you simply need to move the `<title>` tag before the image in the flow of the component, like this:

```
<div class="card">
  <div class="card-block">
    <h4 class="card-title">Card title</h4>
  </div>
  
  <div class="card-block">
    <p class="card-text">Some basic description text for your card should
      appear in this section.</p>
    <a href="#" class="btn btn-primary">Button</a>
  </div>
</div>
```

There are a couple of things here that you need to know about:

- There are now two instances of `<div class="card-block">` in this card. It is perfectly fine to reuse this section within a single card. You'll notice that the header tag is wrapped inside of this `<div>`. This is required to apply the proper padding and margin around the title in the card.
- The second thing you need to note is that the header tag has been moved above the image in the Card layout.

After making this change, your card should look like this:



Hopefully this shows you how easy it is to work with different content in cards. Let's continue by showing some other things that you can do.

Changing text alignment in cards

By default, text and elements will always align left in a card. However, it is possible to change this quite easily. Let's create a second card and then we'll center one and right align the other. I'm going to remove the image so the code is easier to understand:

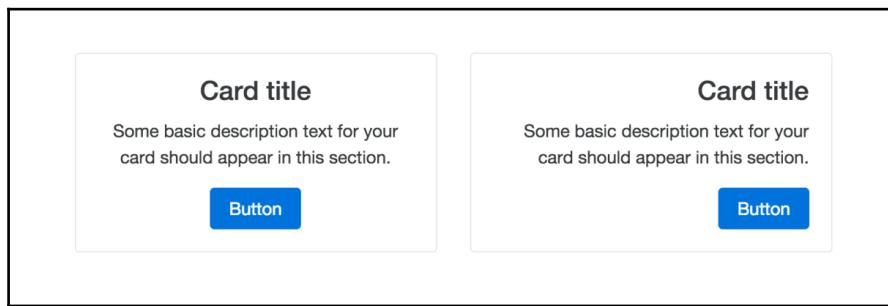
```
<div class="card">
  <div class="card-block text-xs-center">
    <h4 class="card-title">Card title</h4>
    <p class="card-text">Some basic description text for your card should
      appear in this section.</p>
    <a href="#" class="btn btn-primary">Button</a>
  </div>
```

```
</div>
<div class="card">
  <div class="card-block text-xs-right">
    <h4 class="card-title">Card title</h4>
    <p class="card-text">Some basic description text for your card should appear in this section.</p>
    <a href="#" class="btn btn-primary">Button</a>
  </div>
</div>
```

Not much has changed here, but let's go over what is different:

- First, as I mentioned, I removed the image to make the code simpler
- On the first card, I've added a class of `.text-xs-center`, which will center the text in the card
- On the second card, I added a class named `.text-xs-right`, which will right align everything

That's all you need to do. If you view this in the browser it should look like this:



So with one additional CSS class we can easily control the alignment of the text and elements in a card. Cards are a pretty powerful component, so let's continue to learn how you can customize them.

Adding a header to a Card

If you want to add a header to your Card, this is also pretty easy to do. Check out this code sample to see it in action:

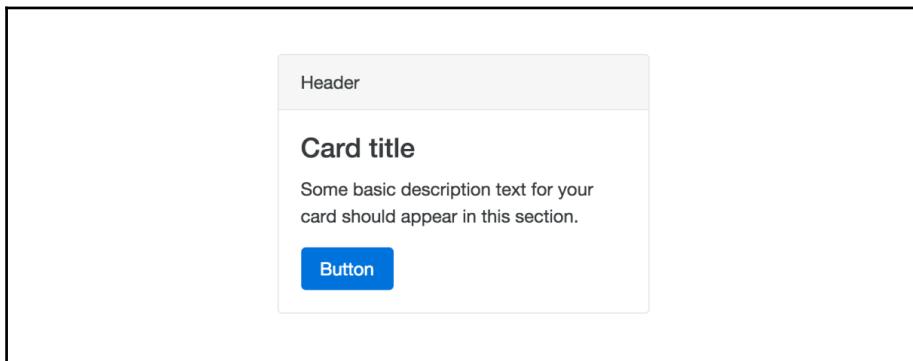
```
<div class="card">
  <div class="card-header">
    Header
  </div>
```

```
<div class="card-block">
  <h4 class="card-title">Card title</h4>
  <p class="card-text">Some basic description text for your card should
appear in this section.</p>
  <a href="#" class="btn btn-primary">Button</a>
</div>
</div>
```

With the addition of a new section of code, we can add a header:

- Before the `.card-block` section, insert a new `<div>` with a class named `.card-header`
- Within this new `<div>`, you can add the header title

Save your file and check it out in the browser, and it should look like this:



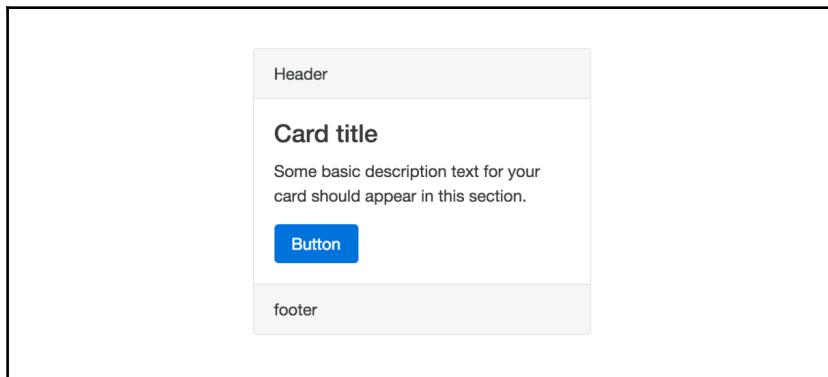
That's a super easy way to add a header section to your card. You can add a footer in the same manner. Let's add some additional code for the footer:

```
<div class="card">
  <div class="card-header">
    Header
  </div>
  <div class="card-block">
    <h4 class="card-title">Card title</h4>
    <p class="card-text">Some basic description text for your card should
appear in this section.</p>
    <a href="#" class="btn btn-primary">Button</a>
  </div>
  <div class="card-footer">
    footer
  </div>
</div>
```

The setup for the footer is very similar to the header; let's break it down:

- This time, below the `.card-block` section, insert a new `<div>` with a class named `.card-footer`
- Inside this new `<div>`, insert your footer text

Save the file again and view it in the browser, and it should look like this:



Easy as that, we've now also included a footer with our Card. Next, let's learn a way to apply a different look and feel to our Card.

Inverting the color scheme of a Card

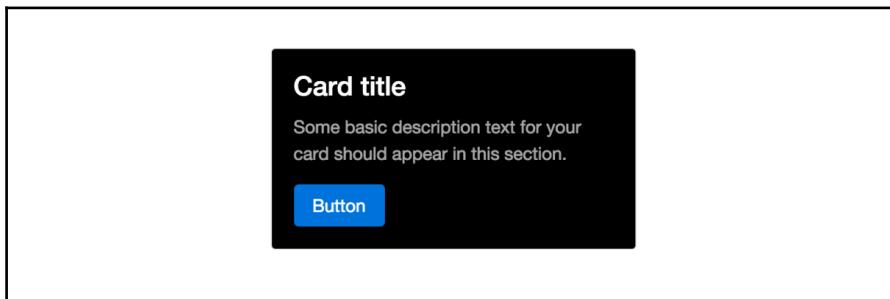
In some cases, you may want a different look and feel for your Card to make it stand out more. There are some CSS classes included with Bootstrap that will allow you to inverse the color scheme. Let's take a look at the code to apply this style:

```
<div class="card card-inverse" style="background:#000;">
  <div class="card-block">
    <h4 class="card-title">Card title</h4>
    <p class="card-text">Some basic description text for your card should
      appear in this section.</p>
    <a href="#" class="btn btn-primary">Button</a>
  </div>
</div>
```

Again, this variation is pretty easy to apply with a couple of small changes:

- On the `<div>` with the `.card` class, add a second class named `.card-inverse`.
- This will only inverse the text in the card. You need to set the `background-color` yourself. For speed, I just did an inline CSS style in the demo code. I'd recommend actually creating a CSS class in your stylesheet for your own project, which is a nicer way to do things.

That's all you need to do. Once you're done, your card should look like this:



In this case, you do need to specify the custom background color. However, Bootstrap does have some background color variations that you can use if you want to add an additional CSS class. The naming convention for these options is just like buttons and labels. Let's take a look at what the code will look like:

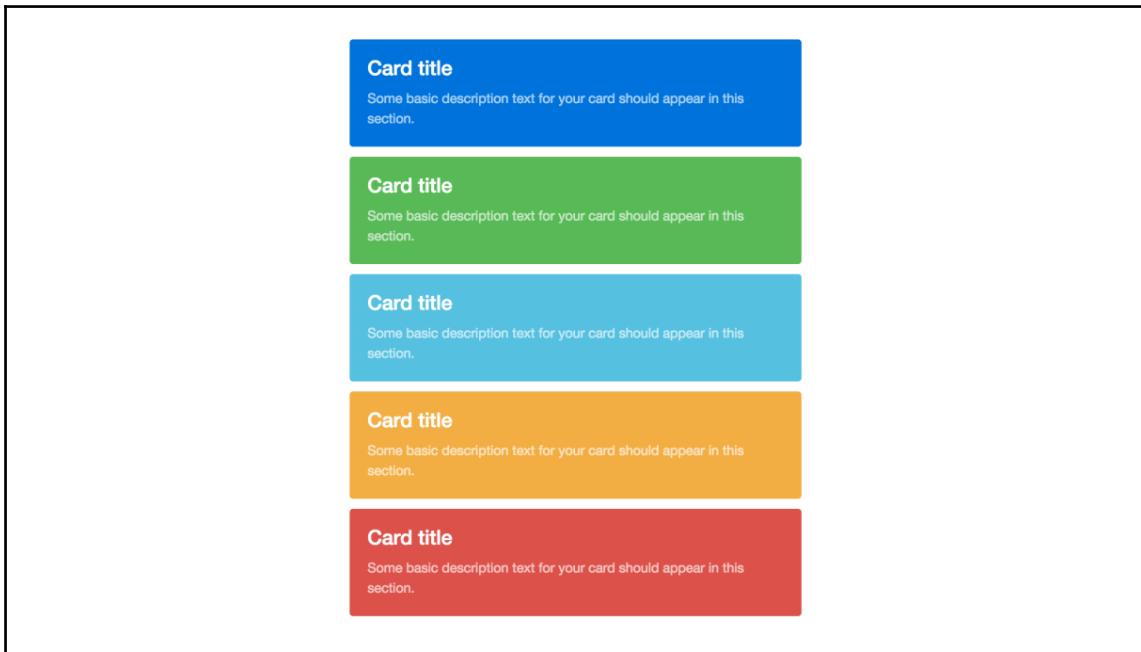
```
<div class="card card-inverse card-primary">
  <div class="card-block">
    <h4 class="card-title">Card title</h4>
    <p class="card-text">Some basic description text for your card should
      appear in this section.</p>
  </div>
</div>
<div class="card card-inverse card-success">
  <div class="card-block">
    <h4 class="card-title">Card title</h4>
    <p class="card-text">Some basic description text for your card should
      appear in this section.</p>
  </div>
</div>
<div class="card card-inverse card-info">
  <div class="card-block">
    <h4 class="card-title">Card title</h4>
    <p class="card-text">Some basic description text for your card should
      appear in this section.</p>
  </div>
</div>
```

```
</div>
</div>
<div class="card card-inverse card-warning">
  <div class="card-block">
    <h4 class="card-title">Card title</h4>
    <p class="card-text">Some basic description text for your card should
appear in this section.</p>
  </div>
</div>
<div class="card card-inverse card-danger">
  <div class="card-block">
    <h4 class="card-title">Card title</h4>
    <p class="card-text">Some basic description text for your card should
appear in this section.</p>
  </div>
</div>
```

This is a bunch of code, but there are only a couple of things that change from our previous card example:

- All I've done is add an additional CSS class to the `<div>` with our base `.card` class on it. Let's review each one in the following points.
- The **Primary** card uses the `.card-primary` class and is blue.
- The **Success** card uses the `.card-success` class and is green.
- The **Info** card uses the `.card-info` class and is light blue.
- The **Warning** card uses the `.card-warning` class and is yellow.
- The **Danger** card uses the `.card-danger` class and is red.

Once you've set up the above code, your cards should look like this in the browser:



That concludes the basic and advanced styling you can do with the Card component. Why don't we take a break from learning for a bit and actually build some Cards in our blog project.

Adding a location card to the Contact page

Let's jump back into our project by adding a simple Card component to the **Contact** page. Reopen `contact.ejs` in your text editor and head down to the main body that we recently updated with a contact form. Find the following column code for that section:

```
<div class="col-md-12">
```

We're going to split this full width column into two separate columns. Change the class on the previous snippet of code to `.col-md-8` and add a new `<div>` with a class of `.col-md-4` on it. When you're done, the body of the page code should now look like this:

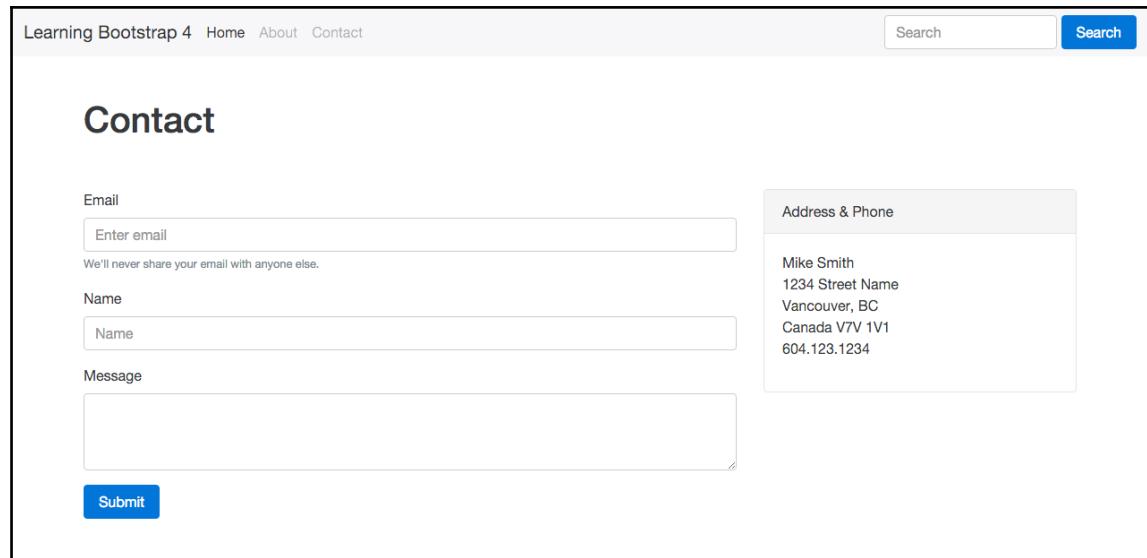
```
<div class="col-md-8">
  <form>
    <fieldset class="form-group">
```

```
<label>Email</label>
<input type="email" class="form-control" placeholder="Enter
email">
    <small class="text-muted">We'll never share your email with
anyone else.</small>
</fieldset>
<fieldset class="form-group">
    <label>Name</label>
    <input type="text" class="form-control" placeholder="Name">
</fieldset>
<fieldset class="form-group">
    <label>Message</label>
    <textarea class="form-control" rows="3"></textarea>
</fieldset>
    <button type="submit" class="btn btn-primary">Submit</button>
</form>
</div>
<div class="col-md-4">
</div>
```

Now that the column is set up, let's insert a Card component into our new column. Enter the following code into the second column in the layout:

```
<div class="card">
    <div class="card-header">
        Address & Phone
    </div>
    <div class="card-block">
        <ul class="list-unstyled">
            <li>Mike Smith</li>
            <li>1234 Street Name</li>
            <li>Vancouver, BC</li>
            <li>Canada V7V 1V1</li>
            <li>604.123.1234</li>
        </ul>
    </div>
</div>
```

Once you've inserted the Card component code, save your file and check it out in a browser. It should look like this:



Now the **Contact** page is starting to take more shape. Let's add the Card component to a few other pages before we move on to our next Content component.

Updating the Blog index page

Now that we've covered the card component, it's time to set up the main layout for our Blog index page. The design is going to rely heavily on the Card component, so let's get to it. First of all, open up `index.ejs` in your text editor and find the body of the page section. The left column will look like this:

```
<div class="col-md-8">
```

Within this `<div>` currently is some filler text. Delete the filler text and insert the following Card component, which will be our first Blog post:

```
<div class="card">
  
  <div class="card-block">
    <h4 class="card-title">Post title</h4>
    <p><small>Posted by <a href="#">Admin</a> on January 1, 2016 in <a href="#">Category</a></small></p>
    <p class="card-text">Some quick example text to build on the card title and make up the bulk of the card's content.</p>
    <a href="#" class="btn btn-primary">Read More</a>
```

```
</div>
</div>
```

Now that we've added our first card to the Blog roll, let's break down what's happening:

- I've started by including a photo I took in Nova Scotia a few summers ago. I've given it a class of `.img-fluid` so it stretches the width of the card.
- From there, I've set up my card exactly like I taught you previously, but in this case, I've added some real content for a blog.

Let's go ahead and add the rest of the Card component code for the blog roll. Insert the following code after the first Card in the left column:

```
<div class="card">
  <div class="card-block">
    <h4 class="card-title">Post title</h4>
    <p><small>Posted by <a href="#">Admin</a> on January 1, 2016 in <a href="#">Category</a></small></p>
    <p>Pellentesque habitant morbi tristique...</p>
    <a href="#" class="btn btn-primary">Read More</a>
  </div>
</div>
<div class="card">
  
  <div class="card-block">
    <h4 class="card-title">Post title <span class="label label-success">Updated</span></h4>
    <p><small>Posted by <a href="#">Admin</a> on January 1, 2016 in <a href="#">Category</a></small></p>
    <p class="card-text">Some quick example text to build on the card title and make up the bulk of the card's content.</p>
    <a href="#" class="btn btn-primary">Read More</a>
  </div>
</div>
<div class="card">
  <div class="card-block">
    <h4 class="card-title">Post title</h4>
    <p><small>Posted by <a href="#">Admin</a> on January 1, 2016 in <a href="#">Category</a></small></p>
    <p>Pellentesque habitant morbi tristique senectus...</p>
    <a href="#" class="btn btn-primary">Read More</a>
  </div>
</div>
```

That's a long chunk of code. The filler text is just in there to give you an idea. Feel free to remove that or replace it with actual text. Now that we've filled out the left column with a good amount of content, your page should look like this:

Learning Bootstrap 4 Home About Contact

Search

Search

Blog

Sidebar

Pelletacula habilit morbi tristique senectus et netus et malesuada fames ac turpis egas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, erat. Donec eu libet sit amet quam egas semper. Aenean ultricies vitae est. Maecenas non libet sit amet quam egas semper. Ut sit amet et sapien ullamcorper pharetra. Vestibulum est wisi, condimentum sed, commodo vitae, orname sit amet, wisi. Aenean fermentum, elit eget invidit condimentum, eros ipsum nrum ord, sagittis tempus lacus enim ac dui. Donec non libet sit amet et sapien ullamcorper pharetra. Vestibulum est wisi, condimentum sed, commodo vitae, orname sit amet, wisi. Aenean fermentum, elit eget invidit condimentum, eros ipsum nrum ord, sagittis tempus lacus enim ac dui. Donec non libet sit amet et sapien ullamcorper pharetra. Ut felis. Praesent dapibus, neque id cursus facilis, tunc neque egas augue, eu vulputate magna eros eu iat. Aliquam erat volutpat. Nam dui mi, inicidunt quis, accumsan portitor, facilis luctus, metus

Post title

Posted by Admin on January 1, 2016 in Category

Some quick example text to build on the card title and make up the bulk of the card's content.

[Read More](#)

Post title

Posted by Admin on January 1, 2016 in Category

Pelletacula habilit morbi tristique senectus et netus et malesuada fames ac turpis egas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, erat. Donec eu libet sit amet quam egas semper. Aenean ultricies vitae est. Maecenas non libet sit amet quam egas semper. Ut sit amet et sapien ullamcorper pharetra. Vestibulum est wisi, condimentum sed, commodo vitae, orname sit amet, wisi. Aenean fermentum, elit eget invidit condimentum, eros ipsum nrum ord, sagittis tempus lacus enim ac dui. Donec non libet sit amet et sapien ullamcorper pharetra. Ut felis. Praesent dapibus, neque id cursus facilis, tunc neque egas augue, eu vulputate magna eros eu iat. Aliquam erat volutpat. Nam dui mi, inicidunt quis, accumsan portitor, facilis luctus, metus

[Read More](#)

Post title Updated

Posted by Admin on January 1, 2016 in Category

Some quick example text to build on the card title and make up the bulk of the card's content.

[Read More](#)

Post title

Posted by Admin on January 1, 2016 in Category

Pelletacula habilit morbi tristique senectus et netus et malesuada fames ac turpis egas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, erat. Donec eu libet sit amet quam egas semper. Aenean ultricies vitae est. Maecenas non libet sit amet quam egas semper. Ut sit amet et sapien ullamcorper pharetra. Vestibulum est wisi, condimentum sed, commodo vitae, orname sit amet, wisi. Aenean fermentum, elit eget invidit condimentum, eros ipsum nrum ord, sagittis tempus lacus enim ac dui. Donec non libet sit amet et sapien ullamcorper pharetra. Ut felis. Praesent dapibus, neque id cursus facilis, tunc neque egas augue, eu vulputate magna eros eu iat. Aliquam erat volutpat. Nam dui mi, inicidunt quis, accumsan portitor, facilis luctus, metus

[Read More](#)

Pelletacula habilit morbi tristique senectus et netus et malesuada fames ac turpis egas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, erat. Donec eu libet sit amet quam egas semper. Aenean ultricies vitae est. Maecenas non libet sit amet quam egas semper. Ut sit amet et sapien ullamcorper pharetra. Vestibulum est wisi, condimentum sed, commodo vitae, orname sit amet, wisi. Aenean fermentum, elit eget invidit condimentum, eros ipsum nrum ord, sagittis tempus lacus enim ac dui. Donec non libet sit amet et sapien ullamcorper pharetra. Ut felis. Praesent dapibus, neque id cursus facilis, tunc neque egas augue, eu vulputate magna eros eu iat. Aliquam erat volutpat. Nam dui mi, inicidunt quis, accumsan portitor, facilis luctus, metus

Learnin Bootstrap 4 © 2016

Now that the main blog roll content is complete, let's also add the right column content.

Adding the sidebar

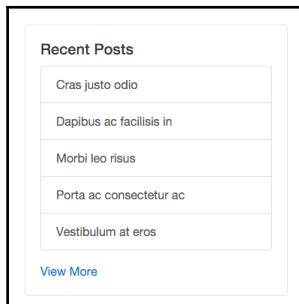
Let's add some more content to the index page by adding the sidebar. We'll also use the Card component here, but in this case, some different variations of it. Go back to `index.ejs` and remove the filler text from the second column. Instead, insert the following Card code:

```
<div class="card card-block">
  <h5 class="card-title">Recent Posts</h5>
  <div class="list-group">
    <button type="button" class="list-group-item">Cras justo odio</button>
    <button type="button" class="list-group-item">Dapibus ac facilisis
      in</button>
    <button type="button" class="list-group-item">Morbi leo risus</button>
    <button type="button" class="list-group-item">Porta ac consectetur
      ac</button>
    <button type="button" class="list-group-item">Vestibulum at
      eros</button>
  </div>
  <div class="m-t-1"><a href="#">View More</a></div>
</div>
```

You'll notice in this Card I'm using a different variation, which is the List Group option. To do this, follow these steps:

- Create a new `<div>` with a class of `.list-group` inside your card.
- Inside, insert a `<button>` with a class of `.list-group-item` on it for every item of your list. I've added five different options.

Once you're done, save your file and it should look like this in the browser:



As you can see, that will draw a nice-looking sidebar list component. Let's fill out the rest of the sidebar by inserting the following code after the first Card component:

```
<div class="card card-block">
  <h5 class="card-title">Archives</h5>
  <div class="list-group">
    <button type="button" class="list-group-item">Cras justo odio</button>
    <button type="button" class="list-group-item">Dapibus ac facilisis
      in</button>
    <button type="button" class="list-group-item">Morbi leo risus</button>
    <button type="button" class="list-group-item">Porta ac consectetur
      ac</button>
    <button type="button" class="list-group-item">Vestibulum at
      eros</button>
  </div>
  <div class="m-t-1"><a href="#">View More</a></div>
</div>
<div class="card card-block">
  <h5 class="card-title">Categories</h5>
  <div class="list-group">
    <button type="button" class="list-group-item">Cras justo odio</button>
    <button type="button" class="list-group-item">Dapibus ac facilisis
      in</button>
    <button type="button" class="list-group-item">Morbi leo risus</button>
    <button type="button" class="list-group-item">Porta ac consectetur
      ac</button>
    <button type="button" class="list-group-item">Vestibulum at
      eros</button>
  </div>
  <div class="m-t-1"><a href="#">View More</a></div>
</div>
```

This will produce two more List Group Card components for the sidebar of your blog project. Once it's all done, the entire page should now look like this:

Learning Bootstrap 4 Home About Contact Search

Blog

Post title

Posted by Admin on January 1, 2016 in Category

Some quick example text to build on the card title and make up the bulk of the card's content.

[Read More](#)

Post title

Posted by Admin on January 1, 2016 in Category

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci, sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar facilisis. Ut felis. Praesent dapibus, neque id cursus faucibus, tortor neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus, metus.

[Read More](#)

Post title

Posted by Admin on January 1, 2016 in Category

Some quick example text to build on the card title and make up the bulk of the card's content.

[Read More](#)

Post title

Posted by Admin on January 1, 2016 in Category

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci, sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar facilisis. Ut felis. Praesent dapibus, neque id cursus faucibus, tortor neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus, metus.

[Read More](#)

Post title

Posted by Admin on January 1, 2016 in Category

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci, sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar facilisis. Ut felis. Praesent dapibus, neque id cursus faucibus, tortor neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus, metus.

[Read More](#)

Learning Bootstrap 4 2016

That concludes the user of the Card component on the index page. The last page we need to set up with the Card component is our Blog post page.

Setting up the Blog post page

The index page is a list of all the Blog posts in our project. The last page we need to setup is the Blog post page, which is just a single post in our project. Open up the `blog-post.ejs` template you created earlier in the book and let's start updating some code. Head down to the page body section and find this line of code:

```
<div class="col-md-8">
```

Currently, you'll see some filler text in that `<div>`; let's replace it with the following code:

```
<div class="card">
  <div class="card-block">
    <p><small>Posted by <a href="#">Admin</a> on January 1, 2016 in <a href="#">Category</a></small></p>
    <p>Pellentesque habitant morbi tristique senectus et...</p>
    <p><code>&lt;p&gt;this is what a code sample looks like&lt;/p&gt;</code></p>
    <p>Pellentesque habitant morbi tristique senectus et netus...</p>
    <!-- pre sample start //-->
    <h4>pre sample code</h4>
    <pre>This is what code will look like</pre>
    <!-- pre sample end //-->
    <!-- image //-->
    <h4>responsive image</h4>
    <p></p>
    <!-- table //-->
    <h4>table</h4>
    <table class="table">
      <thead>
        <tr>
          <th>#</th>
          <th>First Name</th>
          <th>Last Name</th>
          <th>Username</th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <th scope="row">1</th>
          <td>john</td>
          <td>smith</td>
          <td>@jsmith</td>
        </tr>
        <tr>
          <th scope="row">2</th>
```

```
<td>steve</td>
<td>stevens</td>
<td>@steve</td>
</tr>
<tr>
  <th scope="row">3</th>
  <td>mike</td>
  <td>michaels</td>
  <td>@mike</td>
</tr>
</tbody>
</table>
</div>
</div>
```

There's a good chunk of things going on in this code. I've thrown in a few other components we've already learned about so you can see them in action. The Card component has the following things included inside it:

- Text, `<code>` and `<pre>` tags
- Tables
- Images

Let's also update this template to use the same sidebar code as the index page. Copy the right column code from the index template and paste it into the same location in the blog post template.

When you are done, the page should now look like this:

Learning Bootstrap 4 Home About Contact

Post Title

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci, sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar facilisis. Ut felis. Praesent dapibus, neque id cursus facilisis, tortor neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus, metus.

Posted by Admin on January 1, 2016 in Category

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. [click for a popover](#) Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci, sagittis tempus lacus enim ac dui. [Hover for tooltip](#) Donec non enim in turpis pulvinar facilisis. Ut felis. Praesent dapibus, neque id cursus facilisis, tortor neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus, metus.

<pre>this is what a code sample looks like</pre>

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci, sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar facilisis. Ut felis. Praesent dapibus, neque id cursus facilisis, tortor neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus, metus.

pre sample code
This is what code will look like

responsive image



table

#	First Name	Last Name	Username
1	john	smith	@jsmith
2	steve	stevens	@steve
3	mike	michaels	@mike

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci, sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar facilisis. Ut felis. Praesent dapibus, neque id cursus facilisis, tortor neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus, metus.

Learning Bootstrap 4 2016

Recent Posts

Cras justo odio
Dapibus ac facilisis in
Morbi leo risus
Porta ac consectetur ac
Vestibulum at eros

[View More](#)

Archives

Cras justo odio
Dapibus ac facilisis in
Morbi leo risus
Porta ac consectetur ac
Vestibulum at eros

[View More](#)

Categories

Cras justo odio
Dapibus ac facilisis in
Morbi leo risus
Porta ac consectetur ac
Vestibulum at eros

[View More](#)

As you can see, we're using a single Card component to hold all of the content for the body of the page. We're also using the same Card components for the sidebar that we copied over from the index page. Now that we've added the Cards to all of our page templates, let's get back to learning about some other Content components in Bootstrap 4.

How to use the Navs component

The Navs component in Bootstrap can be displayed in a couple of different ways. The default view for the component is just a simple unstyled list of links. This list can also be transformed into tabs or pills for ways of organizing your content and navigation. Let's start by learning how to create a default Nav component:

```
<ul class="nav">
  <li class="nav-item">
    <a class="nav-link" href="#">Link 1</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Link 2</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Link 3</a>
  </li>
</ul>
```

The most basic version of the Nav component is built using the preceding code:

- The component is based on an unordered list with a class of .nav
- Each `` tag in the list requires a class of .nav-item
- Nested inside the `` tag must be an `<a>` tag with a class of .nav-link

Once you've completed adding that code it should look like this in the browser:



As I mentioned, this is just a basic unstyled list of links. One easy change you can make is to display the list of links inline horizontally. To achieve this, you just need to add a class named `.nav-inline` to the `` tag, like this:

```
<ul class="nav nav-inline">
```

This will display all the links in a horizontal line. Why don't we move on to something a little more exciting, such as converting this list into tabs.

Creating tabs with the Nav component

Converting the basic list to tabs is easy to do by adding a couple of things to our code. Take a look at this sample:

```
<ul class="nav nav-tabs">
  <li class="nav-item">
    <a class="nav-link active" href="#">Link 1</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Link 2</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Link 3</a>
  </li>
</ul>
```

I've made two changes to the code, let's review them now:

- On the `` tag, I removed the `.nav-inline` class and added `.nav-tabs`. This will render the list as tabs.
- I then added a class of `.active` to the first link so that it is the selected tab when the page loads.

After you've coded that up, it should look like this in the browser:



Just like that you can render the list as a set of tabs. The next variations you'll want to try are pills.

Creating a pill navigation

Changing the style of the Nav component to Pills is actually really easy. Take a look at the following sample code:

```
<ul class="nav nav-pills">
  <li class="nav-item">
    <a class="nav-link active" href="#">Link 1</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Link 2</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Link 3</a>
  </li>
</ul>
```

Let's breakdown what is new here. I've only made one change to the code. I've removed the `.nav-tabs` class from the `` tag and replaced it with a `.nav-pills` class. This is the only change you need to make.

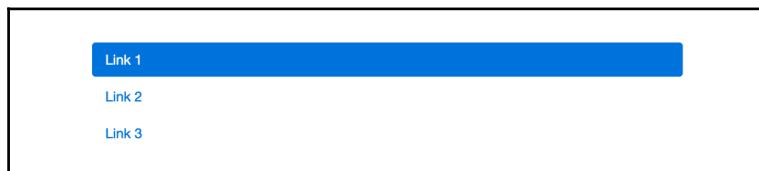
Save your file with the changes and it should look like this in the browser:



The preceding example is the default display for Nav pills. There is another variation you can try though, which are stacked pills. This pattern is commonly used in sidebar navigations. To create this version, update the following line of code:

```
<ul class="nav nav-pills nav-stacked">
```

Here I've simply added a class of `.nav-stacked` to the `` tag to stack the pills. Here's how it will look in the browser:



That concludes the Nav component in Bootstrap 4. As you learned, it's pretty easy to create a few different styles of navigation with a simple list of unordered links. In the next section, we'll review the more complicated navigation component, which is the Navbar.

Using the Bootstrap Navbar component

The Navbar component is a staple of Bootstrap that gets used all the time. In the past, this component has required a decent amount of markup to get it working. I'm glad to report that in Bootstrap 4 they have simplified this component and made it easier to use. Let's start by going over a basic example of the Navbar:

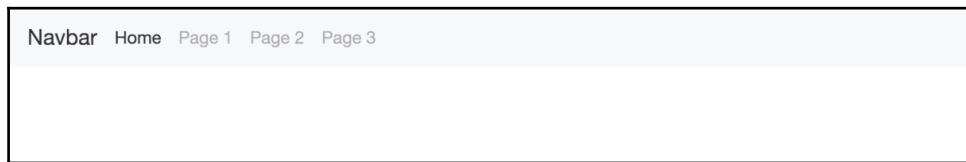
```
<nav class="navbar navbar-light bg-faded">
  <a class="navbar-brand" href="#">Navbar</a>
  <ul class="nav navbar-nav">
    <li class="nav-item active">
      <a class="nav-link" href="#">Home</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">Page 1</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">Page 2</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">Page 3</a>
    </li>
  </ul>
</nav>
```

You may notice some similarities here with the Nav component. The Navbar uses some of the same code, but you can extend it further and combine additional components into it. Let's start by breaking down this basic example:

- A Navbar component can be used outside or inside of a `<div>` with a `.container` class on it. If you want the Navbar to be flush with the edges of the browser, you should not include it inside a `.container` `<div>`. However, if you do want the default padding and margins applied, put it inside the `<div>`. For this example, I'm going to build it outside of a container.
- The Navbar component starts with an HTML5 `<nav>` tag that has the following CSS classes added to it.
 - `.navbar` is the default class that always needs to appear on the component.
 - `.navbar-light` is the color of component you want to use. There are some other variations you can pick from.

- `.bg-faded` is a utility class that you can use to make the background lighter. This is an optional class.
- The first element inside of a Navbar is the Brand. The Brand should be the title for your project. To render the element, create an `<a>` tag and give it a class of `.navbar-brand`. The anchor text for this link should be the name of your project or website. Keep in mind, using the Brand is optional.
- The core part of the Navbar is the list of navigation links. This is created with an unordered list, similar to the Nav component. In this case, your `` tag should have classes of `.nav` and `.navbar-nav` included.
- The nested `` and `<a>` tags should use the same `.nav-item` and `.nav-link` classes from the Nav component.

This will create a basic Navbar component for you. This is how it should look in the browser:



Now that you've learned how to build a basic Navbar, let's learn how to extend the component further.

Changing the color of the Navbar

In Bootstrap 3, you could invert the color scheme of the Navbar. However, in Bootstrap 4 you have multiple options for coloring the Navbar component. All that is needed to edit is some of the classes on the `<nav>` tag that wrap the component. Let's take a look at the code for some of the different color options:

```
<nav class="navbar navbar-inverse">
  ...
</nav>
<nav class="navbar navbar-primary">
  ...
</nav>
<nav class="navbar navbar-success">
  ...
</nav>
<nav class="navbar navbar-warning">
  ...
```

```
</nav>
<nav class="navbar navbar-info">
  ...
</nav>
<nav class="navbar navbar-danger">
  ...
</nav>
```

As you can see, we're reusing the keywords for color variations that we've used in other components. Let's break down each variation of the Navbar component:

- .navbar-inverse will color the component black and grey
- .navbar-primary will color the component blue
- .navbar-success will color the component green
- .navbar-warning will color the component yellow
- .navbar-info will color the component light blue
- .navbar-danger will color the component red

Once you're done coding that up, the navbars should look like this in the browser:



As you can see, we now have the Navbar in a whole range of colors you can choose from. Let's learn what else we can add to this component.

Making the Navbar responsive

Being that Bootstrap is a mobile-first framework, it would only make sense that you need the ability to make the Navbar component responsive. Let's check out the basic code for this:

```
<nav class="navbar navbar-light bg-faded">
    <button class="navbar-toggler hidden-sm-up" type="button" data-
        toggle="collapse" data-target="#responsive-nav">
        ≡
    </button>
    <div class="collapse navbar-toggleable-xs" id="responsive-nav">
        <a class="navbar-brand" href="#">Navbar</a>
        <ul class="nav navbar-nav">
            <li class="nav-item active">
                <a class="nav-link" href="#">Home</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#">Page 1</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#">Page 2</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#">Page 3</a>
            </li>
        </ul>
    </div>
</nav>
```

There's a few different things in the code here that you need to be aware of:

- After the opening `<nav>` class, you need to insert a `<button>` with the CSS classes `.navbar-toggle` and `.hidden-sm-up`. The first class says this button will toggle the navigation. The second class says only show the responsive navigation for sizes above small. You also need to include the data attribute `data-toggle="collapse"` to all the Nav to collapse. Finally, you need to add a `data-target`, which will point to the area you want to be collapsible. I've given that an ID of `#responsive-nav`.
- Next, head down to your list of links and wrap a `<div>` around them. This section needs CSS classes named `.collapse` and `.navbar-toggleable-xs`. You also need to give it an ID of `responsive-nav` to tie it to the button from the previous step.

That's it; once you code this up shrink your browser window to a small size and your bar should switch to look like this. Oh, and don't forget that the code `=` in the button will render a hamburger menu icon in the responsive Navbar:



That concludes the Navbar component in Bootstrap 4. I know this has been a long chapter, but we only have a few more components to go over.

Adding Breadcrumbs to a page

The Breadcrumbs component is a pretty easy one to use in Bootstrap. Let's check out the code for how to render one:

```
<ol class="breadcrumb">
  <li><a href="#">Home</a></li>
  <li><a href="#">Page 1</a></li>
  <li class="active">Page 2</li>
</ol>
```

As you can see, the code for this component is pretty basic, let's review it:

- The Breadcrumb component uses an ordered list or `` tag as its base.
- Within the ordered list, you simply just need to create a list of links. The last item in the list should have a class of `.active` on it.

Adding Breadcrumbs to the Blog post page

For this example, let's actually add some Breadcrumbs to our Blog post page template. Open up `blog-post.ejs` and add the following code after the container `<div>` at the top:

```
<div class="row m-t-1">
  <ol class="breadcrumb">
    <li><a href="#">Home</a></li>
    <li><a href="#">Blog</a></li>
    <li class="active">Post Title</li>
  </ol>
</div>
```

This code should come before the page title and once you make the update, your page should now look like this at the top:

The screenshot shows a web page with a header containing "Learning Bootstrap 4" and links for "Home", "About", and "Contact". To the right is a search bar with a blue "Search" button. Below the header, a breadcrumb navigation bar shows "Home / Blog / Post Title". The main content area has a heading "Post Title" and a large block of placeholder text from the "Blog Post" component.

There, now we've added a nice Breadcrumb to our blog post template. Let's move on to adding Pagination to our page templates.

Using the Pagination component

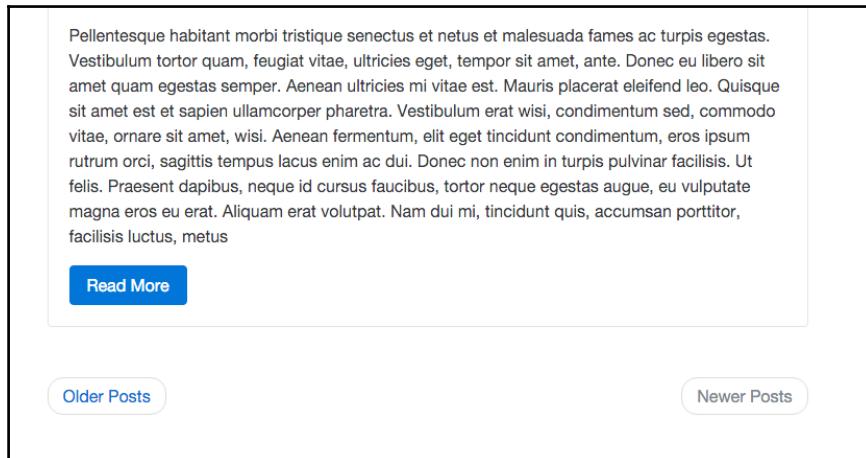
Let's continue adding some more components to our templates by learning how to use the Pagination component. For our blog project, we want to use the Pager version of the component. Open up `index.ejs` and insert the following code after the last Card component in our blog feed:

```
<nav>
  <ul class="pager m-t-3">
    <li class="pager-prev"><a href="#">Older Posts</a></li>
    <li class="pager-next disabled"><a href="#">Newer Posts</a></li>
  </ul>
</nav>
```

The Pager is wrapped in an HTML5 `<nav>` tag and uses an unordered list as its base:

- The `` tag should have a class of `.pager` added to it.
- The first list item in the group should have a class of `.pager-prev` on it.
- The second list item should have a class of `.pager-next` on it. In this case, I've also added the class `.disabled` which means there are no more posts to go to.

After you've added this code to your index template, it should look like this in the browser:



Let's also add this component to the Blog post page template.

Adding the Pager to the Blog post template

Open up `blog-post.ejs` and paste the same snippet of code from previously at the bottom of the left column, right after the end of the Card component. I won't bother posting another screenshot, as it should look the same as the previous example. Let's continue by learning how to use another component.

How to use the List Group component

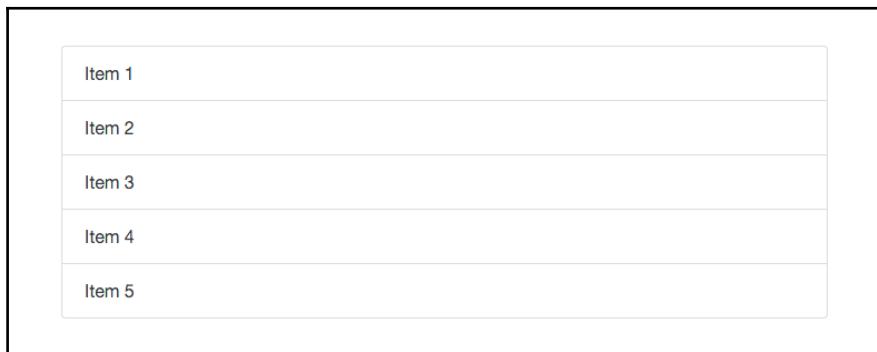
This is the last main content component we need to go over for this chapter. Let's get right into it by reviewing the code needed to render a List Group:

```
<ul class="list-group">
  <li class="list-group-item">Item 1</li>
  <li class="list-group-item">Item 2</li>
  <li class="list-group-item">Item 3</li>
  <li class="list-group-item">Item 4</li>
</ul>
```

Like the components before it, this one is based off of an unordered list:

- The `` tag needs a class of `.list-group` on it to start
- Each `` needs a class of `.list-group-item` on it

Once you're done, your List Group should look like this in the browser:



As you can see, with some minimal coding you can render a decent looking component. You may have missed it, but we actually already used this component when we were building our sidebar on the index and blog post page templates. Open up one of them in a text editor and you'll see the following code, which is a List Group:

```
<div class="card card-block">
  <h5 class="card-title">Recent Posts</h5>
  <div class="list-group">
    <button type="button" class="list-group-item">Cras justo odio</button>
    <button type="button" class="list-group-item">Dapibus ac facilisis
      in</button>
    <button type="button" class="list-group-item">Morbi leo risus</button>
    <button type="button" class="list-group-item">Porta ac consectetur
      ac</button>
    <button type="button" class="list-group-item">Vestibulum at
      eros</button>
  </div>
  <div class="m-t-1"><a href="#">View More</a></div>
</div>
```

That concludes the use of the List Group component. That also concludes the Content components chapter.

Summary

This has been a really long chapter but I hope you have learned a lot. We have covered Bootstrap components including buttons, button groups, button dropdown, forms, input groups, dropdowns, Jumbotron, Label, Alerts, Cards, Navs, Navbar, Breadcrumb, Pagination, and List Group. Our blog project is really starting to take shape now, too. In the next chapter, we'll dive into some JavaScript components in Bootstrap 4 that will include Modal, Tooltips, Popovers, Collapse, and Carousel.

7

Extending Bootstrap with JavaScript Plugins

In this chapter, we're going to dive deeper into Bootstrap components by learning how to extend the framework using JavaScript plugins. You may remember that back in the first chapter we included `bootstrap.min.js` in our template. This file contains a number of JavaScript components that come with Bootstrap. In this chapter, we'll go over how to use some of these components, including: Modals, Tooltips, Popovers, Collapse, and Carousel. Let's get right to it by learning how to create a Modal in Bootstrap 4.

Coding a Modal dialog

Modals go by a number of different names; you may also know them as dialogs, popups, overlays, or alerts. In the case of Bootstrap, this component is referred to as a Modal and that is how I'll be referring to it throughout the book. A Modal is made up of two required pieces of code. The first is a button, and here's the basic code required to render it:

```
<button type="button" class="btn btn-primary" data-toggle="modal" data-target="#firstModal">  
  Open Modal  
</button>
```

As you can see, this is a basic Button component with a few attributes added to it:

- The first is the `data-toggle` data attribute, which needs to be set to `modal`. This tells the browser that this `<button>` is attached to a Modal component.
- The second is the `data-target` attribute, which should be an ID. It doesn't really matter what you name this, I've called it `#firstModal`. It's important to note this ID name as it will be tied in later. Also make sure that the ID name is unique.

Once you've coded this up, it should look like a regular button in the browser:



Coding the Modal dialog

The second part of the Modal component is the dialog. This is the part that will pop up in the browser once you click the button. Let's take a look at some basic code for creating the dialog:

```
<div class="modal fade" id="firstModal" tabindex="-1" role="dialog" aria-hidden="true">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <button type="button" class="close" data-dismiss="modal" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
        <h4 class="modal-title">Dialog Title</h4>
      </div>
      <div class="modal-body">
        Some copy for your modal.
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>
        <button type="button" class="btn btn-primary">Save</button>
      </div>
    </div>
  </div>
</div>
```

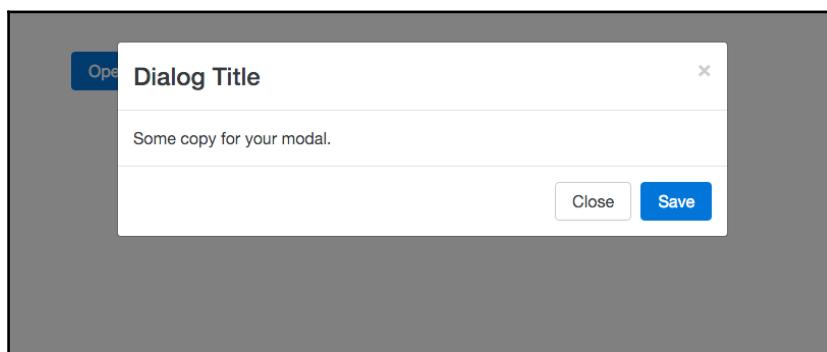
This is a bigger piece of code and there are a few things going on here that I need to explain to you:

- The entire dialog is wrapped in a `<div>` with a required class of `.modal`. There's also an optional `.fade` class there, which will fade the dialog in. Note the ID on this `<div>` because it's important. The ID value needs to match the `data-target` attribute you set on the button. This is how we tell the browser to link that button with this dialog. Finally, there are a couple of other attributes that are required by

Bootstrap including `tabindex`, `role`, and `aria-hidden`. Make sure you include those with their corresponding values.

- Inside the first `<div>` we have a second one with a class of `.modal-dialog` on it; make sure you include that.
- Next, the interior of the Modal is split into three parts: header, body, and footer.
- Inside our `.modal-dialog`, add another `<div>` with a class of `.modal-header` on it. Within this section you'll notice another button. This button is the **Close** or **X** icon for the Modal; although not required, it's a good idea to include this.
- After the button you need to include a header tag, in this case a `<h4>`, with a CSS class of `.modal-title` on it. Here you should enter the title for your Modal.
- The next section is another `<div>` for our body and it has a class of `.modal-body` on it. Within this section you should enter the body copy for your Modal.
- Finally, we have the footer section, which is another `<div>` with a class of `.modal-footer` on it. Inside this section you'll find two buttons that you need to include. The first is the white button labeled **Close** which when clicked will close the Modal. Note that the `<button>` tag has a data attribute called `data-dismiss` on it and its value is `modal`. This will close the Modal. The second button is a primary button that would be used as a Save button if you were hooking in the actual functionality.

After coding all that up, go to the browser and click on your button. You should then see a Modal that looks like this:



As you can see, our Modal has popped up over the button. You can read the Modal title and body and see the footer buttons as well as the **Close** or **X** button in the top-right corner. You may have noticed that you didn't actually have to write any JavaScript to make this Modal work. That is the power of the Bootstrap framework; all of the JavaScript is already written for you and you can simply call the Modal functionality by using the HTML data attributes, which makes things much easier. That concludes the lesson on Modals; next let's move on to learning how to use Tooltips.

Coding Tooltips

A Tooltip is a marker that will appear over a link when you hover over it in the browser. They are pretty easy to add with data attributes in Bootstrap, but we do need to make some updates to get them working. In Bootstrap 4 they have started using a third-party JavaScript library for Tooltips called Tether. Before we go any further, head over to the Tether website below and download the library:

<http://github.hubspot.com/tether/>

Once you've downloaded the library, unzip it and open the main directory where you'll see a number of files. Navigate to the `/dist/js` directory and find the file named `tether.min.js`:

Now copy `tether.min.js` into the `/js` directory of our blog project. This is the only file you need from Tether's directory, so you can keep the rest of the files or delete them. Once the file is in our project directory we need to update our template.

Updating the project layout

Now that we have the Tether file in our project directory we need to update our `_layout.ejs` template to include it when the page is compiled. From the root of our project directory, open up `_layout.ejs` and insert the following line of code near the bottom after `jQuery`. It's critical that the Tether file is loaded after `jQuery`, but before `bootstrap.min.js`:

```
<script src="js/tether.min.js"></script>
```

Save the file and make sure you recompile your project so that this is imported into all of your HTML files. Once that's done, you will now be able to use Tooltips on any page that is included in our project.

How to use Tooltips

Now that we've included the Tether library, we can learn how to actually use Tooltips in Bootstrap. Let's try them out on one of our project files. Open up `index.ejs` in your text editor and find a section of code that is just text, like this:

```
<p>Pellentesque habitant morbi tristique senectus et netus et malesuada  
fames ac turpis egestas...</p>
```

Once you've found that section of code, let's wrap an `<a>` tag around the first three words with the following attributes on it:

```
<p><a href="#" data-toggle="tooltip" >Pellentesque habitant morbi</a>  
tristique senectus et netus et malesuada fames ac turpis egestas.</p>
```

This is the basic markup needed to render a Tooltip. Let's breakdown what is happening here:

- The `data-toggle` attribute is required to tell the browser that this is a Tooltip. The value should be set to `tooltip`.
- The `title` attribute is also required and the value will be the text that appears in your Tooltip. In this case, I have set it to `This is a tooltip!`.

Before we can test this out in the browser, we need to add something else to our `_layout.ejs` template. Open that file in your text editor and insert the following code after the Tether library:

```
<script src="js/bootstrap.min.js"></script>  
<script>  
  $("a").tooltip();  
</script>
```

In Bootstrap 4, Tooltips need to be initialized before you can use them. Therefore, I'm using a little jQuery here to say that all `a` tags should be initialized to use the `Tooltip` method, which will activate all link tags for use with a Tooltip. This is a little trick you can use so you don't have to use an ID to indicate every Tooltip you want to initialize. Once you've completed this step, save all your files, recompile them, and then view your project in the browser; it should look like this when you rollover the link anchor text:

Post title

Posted **This is a tooltip!** on 1, 2016 in Category

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci, sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar facilisis. Ut felis. Praesent dapibus, neque id cursus faucibus, tortor neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus, metus

[Read More](#)

How to position Tooltips

By default, in Bootstrap the position for Tooltips is above the anchor text. However, using the `data-placement` attribute will allow you to place the tip above, below, left, or right of the anchor text. Let's take a look at the code required to render the different versions:

```
<p><a href="#" data-toggle="tooltip" data-placement="top">Pellentesque  
habitant morbi</a> tristique senectus et netus et malesuada fames ac turpis  
egestas.</p>  
<p><a href="#" data-toggle="tooltip" data-placement="bottom">Pellentesque  
habitant morbi</a> tristique senectus et netus et malesuada fames ac turpis  
egestas.</p>  
<p><a href="#" data-toggle="tooltip" data-placement="right">Pellentesque  
habitant morbi</a> tristique senectus et netus et malesuada fames ac turpis  
egestas.</p>  
<p><a href="#" data-toggle="tooltip" data-placement="left">Pellentesque  
habitant morbi</a> tristique senectus et netus et malesuada fames ac turpis  
egestas.</p>
```

As you can see, I've added the `data-placement` attribute to each link tag. The following values will control the position of the Tooltip when you hover over it:

- Top: `data-placement="top"`
- Bottom: `data-placement="bottom"`
- Right: `data-placement="right"`
- Left: `data-placement="left"`

Adding Tooltips to buttons

It's also quite easy to add a Tooltip to a button by using the same data attributes as links. Let's take a look at how to code a simple button with a Tooltip above it:

```
<button type="button" class="btn btn-primary" data-toggle="tooltip" data-placement="top" data-original-title="This is a button tooltip!></button>
```

Here you'll see a basic button component, but with the Tooltip data attributes:

- I've added the `data-toggle` attribute with a value of `tooltip`
- You can optionally include the `data-placement` attribute; if you leave it out it will default to top
- You need to include the `data-original-title` attribute and the value will be the Tooltip message

Updating the layout for buttons

To get Tooltips on buttons working, you need to initialize them the same way you did the links in the previous section. Open up `_layout.ejs` again in your text editor and include the following line of code. The entire section of JavaScript should now look like this:

```
<script>
  $("a").tooltip();
  $("button").tooltip();
</script>
```

Like we did with the link tags, we'll initialize all button tags to use the Tooltip component if called in the HTML template. Let's take a look at how our Tooltip on a button should look in the browser when it's done correctly:

Posted by Admin on January 1, 2016 in Category

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Pellentesque habitant morbi **tooltip** tristique senectus et netus et malesuada fames ac turpis egestas.

This is a button tooltip!

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci, sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar facilisis. Ut felis. Praesent dapibus, neque id cursus faucibus, tortor neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus, metus

[Read More](#)

Avoiding collisions with our components

Until now we've only used the Tooltip JavaScript component so our code is solid. However, in the next section, we will introduce a different component called Popovers. We need to do some clean up of our JavaScript code so that the two don't collide with each other and give us unwanted results.

Since this is the case, we should go back to `_layout.ejs` and edit the code by providing a specific ID for each Tooltip that you want to use in your project. Our script should now look like this:

```
<script>
  $("#tooltip-link").tooltip();
  $("#tooltip-button").tooltip();
</script>
```



Note that I removed the `a` and `button` selectors and replaced them with IDs named `#tooltip-link` and `#tooltip-button`. Now we also need to update our link and button code on the index template to include these IDs.

```
<p><a id="tooltip-link" data-toggle="tooltip" >Pellentesque habitant
morbi</a> tristique senectus et netus et malesuada fames ac turpis
egestas.</p>
```

```
<button type="button" id="tooltip-button" class="btn btn-primary" data-toggle="tooltip" data-placement="top" data-original-title="This is a button tooltip!</button>
```

As you can see, I've included the ID for each element in the preceding code. Now we are safe to start introducing new components without any worry of collisions occurring in the JavaScript. Let's move on to the component in question; Popovers.

Using Popover components

Popover components are similar to Tooltips but allow for more content to be included. Popovers are also revealed on a click action, not a hover action like Tooltips. Let's take a look at the basic code to render a Popover. First, let's make sure we add this Popover to our project, so open up `index.ejs` again and find another filler line of code to add this new component. When you do, enter the following code into the template:

```
<p><a id="popover-link" data-toggle="popover" data-content="This is the content of my popover which can be longer than a tooltip">This is a popover</a>. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante.</p>
```

As you can see, there are a few new things we need to go over here:

- First of all, you'll notice I've given the link tag this ID; `popover-link`.
- In this case, `data-toggle` is set to `popover`.
- The `title` attribute is required and will be the title for your Popover.
- Finally, we have a new attribute named `data-content`. The value for this should be the copy you want to appear on the Popover.

Updating the JavaScript

Like we did with Tooltips, we also need to update the JavaScript for this new component. Open up `_layout.ejs` again and insert the following line of code after the Tooltip JavaScript:

```
$( "#popover-link" ).popover();
```

This code will initialize a Popover component on the element with the #popover-link ID on it. Once you've completed that, save both files and go to your browser. Find the link you created for the Popover and click it. This is what you should see in the browser:



As you can see, the Popover component has more to it than the Tooltip. It includes a title and content. You should use this component if you need to give more context than can be achieved through the use of a regular Tooltip.

Positioning Popover components

Again, like Tooltips, it is possible to control the position of a Popover component. This is done in the same way by using the data-placement attribute on the link tag. Here's the code for each variation:

```
<p><a id="popover-link" data-placement="top" data-toggle="popover" data-content="This is the content of my popover which can be longer than a tooltip">This is a popover</a>. Pellentesque habitant morbi...</p>
```

```
<p><a id="popover-link" data-placement="bottom" data-toggle="popover" data-content="This is the content of my popover which can be longer than a tooltip">This is a popover</a>. Pellentesque habitant morbi...</p>
```

```
<p><a id="popover-link" data-placement="right" data-toggle="popover" data-content="This is the content of my popover which can be longer than a tooltip">This is a popover</a>. Pellentesque habitant morbi...</p>
```

```
<p><a id="popover-link" data-placement="left" data-toggle="popover" data-content="This is the content of my popover which can be longer than a tooltip">This is a popover</a>. Pellentesque habitant morbi...</p>
```

Since this works in exactly the same way as for Tooltips, I won't bother breaking it down any further. Simply include the data-placement attribute and give it one of the four positioning values to control where the Popover appears when clicked.

Adding a Popover to a button

A Popover component can also be easily added to a button. Open up the index template again and insert the following button code:

```
<p><button type="button" id="popover-button" class="btn btn-primary" data-toggle="popover" data-content="This is a button popover example">Popover Button</button></p>
```

As you can see, this markup is very similar to the Tooltip button. Let's break it down again:

- The button tag needs an ID of `popover-button` to be added
- As with the link, set the `data-toggle` attribute to `popover`
- Include a value for `title` and the `data-content` attribute

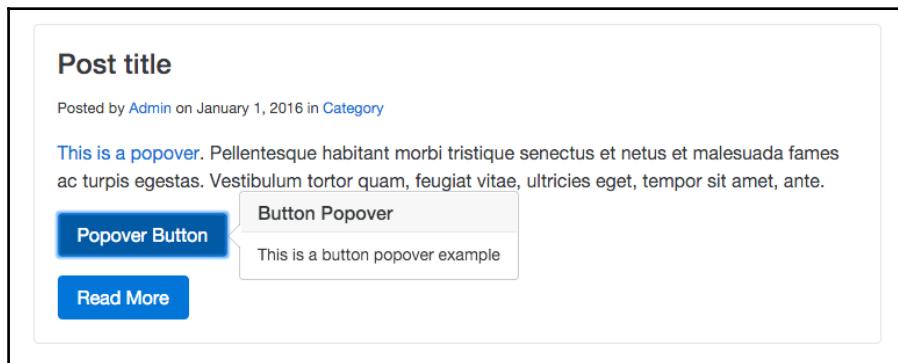
As with the previous examples, don't forget to update the JavaScript!

Adding our Popover button in JavaScript

The last thing we need to do is update the JavaScript to initialize our new Popover button. Open up `_layout.ejs` and insert the following line of code after the Popover link JavaScript:

```
$ ("#popover-button").popover();
```

Once that is complete, save both files and open up the index page in your browser. Locate the button you inserted and click it. Your Popover should look like this:



As you can see, you now have a button with a Popover component attached to it. This can be useful for calling out something important with a button, and then once it has been clicked it reveals a message to your users. I have a couple more JavaScript components I would like to review with you; the next one is the Collapse component.

Using the Collapse component

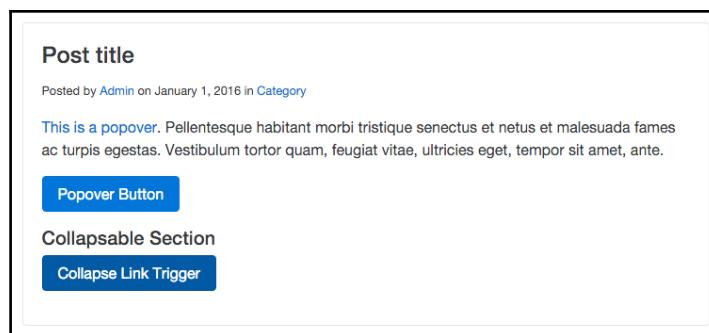
I find that the Collapse component's name is a bit confusing. What it really means is a collapsible section that can be shown or hidden on a click action. Let's start by creating a simple collapsible section of text on the `index.ejs` template. Open that template and insert the following code wherever you like:

```
<p><a class="btn btn-primary" data-toggle="collapse" href="#collapse-link" aria-expanded="false">Collapse Link Trigger</a></p>
```

The Collapse component is broken into two parts. The first is the trigger to show or hide the collapsible content. The second is the actual content you want to show or hide. Let's review it in more detail to show how to code this up:

- The first part is the trigger for the collapsible content, and I have chosen to use a link that has some button classes on it
- The link requires the `data-toggle` attribute with a value of `collapse` on it
- The `href` for the link needs to be a unique ID name, in this case, `#collapse-link`
- Finally, we set the `aria-expanded` value to `false` because we want the collapsible content to be hidden on page load

On page load, your new component should just appear like a regular button:



Coding the collapsible content container

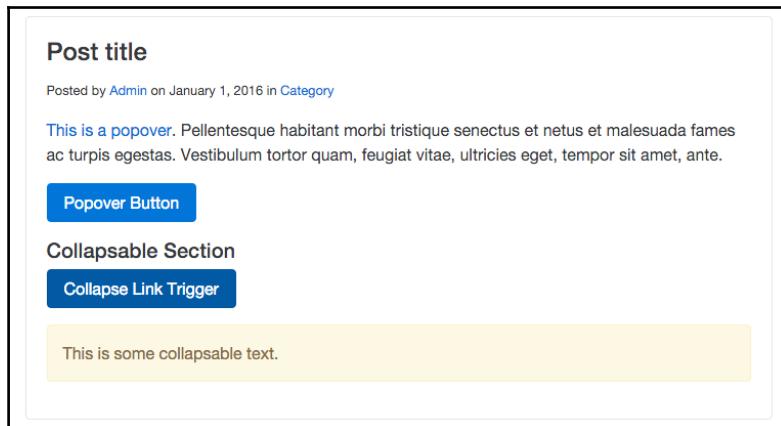
Now that the trigger for the Collapse is set up, we need to code the content container. After the link tag, insert the following code:

```
<div class="collapse" id="collapse-link">
  <p class="alert alert-warning">This is some collapsable text.</p>
</div>
```

Here's how to assemble this section of code:

- We start with a `<div>` that needs to have a CSS class of `collapse` on it. You also need to include an ID here. This should match the ID you set as the `href` in the trigger link; in this case, `#collapse-link`.
- Within the `<div>` you can include any content you want. This content will be the hidden, collapsible content that you will show or hide when the trigger is clicked. To make the example really obvious, I've wrapped a warning Alert around some text to make it stand out.

After you've coded this up and saved your file, head to the browser, find the button, and click it. You should see the following in your window once you click the trigger link:



This is only a simple example of how you can code up the Collapse component. Using additional code and setup, you can use this component to create an Accordion.

Coding an Accordion with the Collapse component

In the previous section, I taught you a pretty simple way to use the Collapse component. The same component can be used to create a more complex version, which is the Accordion. Let's take a look at the basic code to create an Accordion:

```
<div id="accordion">
  <div class="panel panel-default">
    <div class="panel-heading" role="tab" id="headerOne">
      <h4 class="panel-title">
        <a data-toggle="collapse" data-parent="#accordion"
           href="#sectionOne" aria-expanded="true" aria-controls="sectionOne">
          Section #1
        </a>
      </h4>
    </div>
    <div id="sectionOne" class="panel-collapse collapse in" role="tabpanel"
         aria-labelledby="headerOne">
      This is the first section.
    </div>
  </div>
  <div class="panel panel-default">
    <div class="panel-heading" role="tab" id="headerTwo">
      <h4 class="panel-title">
        <a class="collapsed" data-toggle="collapse" data-
           parent="#accordion" href="#sectionTwo" aria-expanded="false" aria-
           controls="sectionTwo">
          Section #2
        </a>
      </h4>
    </div>
    <div id="sectionTwo" class="panel-collapse collapse" role="tabpanel"
         aria-labelledby="headerTwo">
      This is the second section.
    </div>
  </div>
  <div class="panel panel-default">
    <div class="panel-heading" role="tab" id="headerThree">
      <h4 class="panel-title">
        <a class="collapsed" data-toggle="collapse" data-
           parent="#accordion" href="#sectionThree" aria-expanded="false" aria-
           controls="sectionThree">
          Section #3
        </a>
      </h4>
```

```
</div>
<div id="sectionThree" class="panel-collapse collapse" role="tabpanel"
aria-labelledby="sectionThree">
  This is the third section.
</div>
</div>
```

Now that might look like a ton of code, but it's actually a repeating pattern that is pretty easy to put together once you understand it. Let me breakdown everything that is happening here:

- The entire component is wrapped in a `<div>` with an ID on it. In this case, I'm using `#accordion`.
- Each section of the Accordion is a `<div>` with a class of `.panel` on it. I've also included the `.panel-default` class to just do the most basic styling.
- Each panel is made up of a heading and a body or section. Let's cover the header first. Create another `<div>` with a class of `.panel-heading` on it. Also include the `role` attribute with a value of `tab` and you need to give your header a unique ID, in this case, `#headerOne`.
- Inside the header include a header tag, in this case, a `<h4>`, with a class of `.panel-title`.
- Finally, nested inside the header tag, code a link that has a few attributes that you need to include:
 - `.collapsed` is required for the Accordion component.
 - `data-toggle` is also required.
 - `data-parent` should be the same ID that you set on the first `<div>` for the accordion.
 - `href` will be a link to the body of the section that will be collapsable. In this case, it is called `sectionOne`.
 - `aria-expanded` should be set to `true` because we want this section to be open on page load. The other links should be set to `false`, unless you want them to be open on page load.
 - `aria-controls` should also match the ID name of the corresponding section.
 - Now that the header has been broken down, let's cover the body of the panel.

- After the header, insert another `<div>` with an ID of `#sectionOne` on it. It should also have a class of `.panel-collapse` and `.collapse` on it. Include the attribute `role` with a value of `tabpanel` on it. Finally, include `aria-labelledby` with the value of `sectionOne`.
- Inside this `<div>` include the content of the section that you want to display.

For the next sections, you need to repeat what you did for the first panel. Simply copy and paste and then you need to change a few things:

- Change `headerOne` to `headerTwo`
- Change `sectionOne` to `sectionTwo`
- Change up the header title and content of the body for the second section

Do the same for the third section, and then the Accordion component is done. Once you're done, this is what it should look like in the browser:



That completes the Collapse and Accordion components. We have one more to go, which is the Carousel component.

Coding a Bootstrap Carousel

Carousel is a popular component used on many different types of websites. We're going to build a Carousel in the Blog Post template of our project. Let's start by opening up `blog-post.ejs` from the project directory in your text editor. After the page title block of code, insert the following markup:

```
<div id="carousel-example-generic" class="carousel slide" data-  
ride="carousel">  
  <ol class="carousel-indicators">  
    <li data-target="#carousel-example-generic" data-slide-to="0"  
    class="active"></li>  
    <li data-target="#carousel-example-generic" data-slide-to="1"></li>  
    <li data-target="#carousel-example-generic" data-slide-to="2"></li>  
  </ol>
```

```
<div class="carousel-inner" role="listbox">
  <div class="carousel-item active">
    
  </div>
  <div class="carousel-item">
    
  </div>
  <div class="carousel-item">
    
  </div>
</div>
<a class="left carousel-control" href="#carousel-example-generic"
  role="button" data-slide="prev">
  <span class="icon-prev" aria-hidden="true"></span>
  <span class="sr-only">Previous</span>
</a>
<a class="right carousel-control" href="#carousel-example-generic"
  role="button" data-slide="next">
  <span class="icon-next" aria-hidden="true"></span>
  <span class="sr-only">Next</span>
</a>
</div>
```

This is a larger component like the Accordion so let's go through it section by section:

The Carousel component starts with a `<div>` and it needs a unique ID. In this case, `#carouselOne`. Also include the following classes: `.carousel` and `.slide`. Finally, you need to add the attribute `data-ride` with a value of `carousel`.

Adding the Carousel bullet navigation

The first thing we need to add to the Carousel is the bullet or indicator navigation. It's made up of an ordered list. Here's the code, then we'll break it down:

```
<ol class="carousel-indicators">
  <li data-target="#carouselOne" data-slide-to="0" class="active"></li>
  <li data-target="#carouselOne" data-slide-to="1"></li>
  <li data-target="#carouselOne" data-slide-to="2"></li>
</ol>
```

Here's how the Carousel navigation works:

- On the `` tag allocate a class of `.carousel-indicators`.
- Each `` in the list needs to have a few things:
 - The `data-target` needs to be the same ID that you gave to your

root Carousel `<div>`, in this case, `#carouselOne`.

- Include the `data-slide-to` attribute and the first value should be 0. Increase it by one for each list item after the first.

Including Carousel slides

The next step is to include the actual Carousel slides. I'm not going to include images in the code, that will be up to you to insert, but don't worry, I'll show you where to put them. Here's the code for the section that wraps the slides:

```
<div class="carousel-inner" role="listbox">  
  ..  
</div>
```

Give that `<div>` a class of `.carousel-inner` and add the `role` attribute with a value of `listbox`. Inside this `<div>` you're going to add another section for each image slide in the Carousel. Here's the code for one slide in the Carousel:

```
<div class="carousel-item active">  
    
</div>
```

Let's breakdown what's happening here in the code:

- In this case, insert a `<div>` tag with the classes `.carousel-item` and `.active`



Note you should only include the `.active` class on the first slide. This is where the Carousel will start on page load.

- Inside the `<div>`, insert an `img` tag with the following attributes:
 - Insert the `src` attribute and the value should be the path to the image file for the slide
 - Optionally, include an `alt` attribute with a value for the image

Adding Carousel arrow navigation

The last thing we need to add to the Carousel is the arrow navigation. Here's the code for rendering the arrows:

```
<a class="left carousel-control" href="#carouselOne" role="button" data-slide="prev">
  <span class="icon-prev" aria-hidden="true"></span>
  <span class="sr-only">Previous</span>
</a>
<a class="right carousel-control" href="#carouselOne" role="button" data-slide="next">
  <span class="icon-next" aria-hidden="true"></span>
  <span class="sr-only">Next</span>
</a>
```

Let me explain how the arrow navigation works:

- The left and right arrow navigation is based on `href` tags.
- The first will be the left arrow; code a link with the following classes on it: `.left` and `.carousel-control`.
- The `href` for the link should be set to the main ID for the Carousel, in this case, `#carouselOne`.
- Set the `role` attribute to `button`.
- Finally, set the `data-slide` attribute to `prev`.
- Within the link, add a `` with a class of `.icon-prev` on it. This will render the arrow icon. Include the `aria-hidden` attribute and set it to `true`.
- Lastly, you can include another optional `` for accessibility reasons. If you want to include it, give it a class of `.sr-only`. Within the `` include the text `Previous`.
- Now let's go over the differences for the right arrow:
 - Code another link tag and switch the `.left` class to `.right`.
 - Change the `data-slide` attribute value to `next`.
 - In the first `` tag change the class value to `.icon-next`.
 - If you included the accessibility `` tag change the text to `Next`.

That completes the setup of the Carousel component. Fire up the project server and view the Blog Post page in the browser, and it should look like this:



That concludes the chapter on JavaScript components in Bootstrap. In this chapter, I taught you how to code up the following components: Modals, Tooltips, Popovers, Collapse, Accordion, and the Carousel. In the next chapter, I'll teach you how to use **Sass** in Bootstrap.

Summary

In this chapter, we have covered all components in Bootstrap that rely on JavaScript. This included: Modals, Tooltips, Popovers, Collapse, and Carousel.

In the next chapter, we will see how in Bootstrap 4 the framework has moved from Less to Sass as its CSS preprocessor. We will cover the basics of using Sass in a Bootstrap theme. I'll also explain how you can customize or use existing variables, or write your own.

8

Throwing in Some Sass

Up until now we've covered a bunch of different Bootstrap components and how to use them. In this chapter, we're going to change gears and learn about Sass, which will allow you to customize the look and feel of your components. I'll start by introducing some Sass basics that you need to know, move on to writing some basic code, and then show you the power of using variables in your components to save yourself valuable time when creating your web app or project.

Learning the basics of Sass

Sass stands for **Syntactically Awesome Style Sheets**. If you've never used or heard of Sass before, it's a CSS preprocessor. A preprocessor extends regular CSS by allowing the use of things such as variables, operators, and mixins in CSS. Sass is written during the development stage of your project and it needs to be compiled into regular CSS before you deploy your project into production. I'll cover that in more detail in the next section but don't worry because Harp.js makes this really easy to do.

Up until version 4 of Bootstrap, the CSS preprocessor used was actually Less. For a good while both Sass and Less were popular in frontend design circles. However, over the last few years, while Sass has emerged as the best choice for developers, the Bootstrap team decided to make the change in version 4. If you are familiar with Less but have never used Sass, don't worry as they are pretty similar to use so it won't take much to get you up-to-speed.

Using Sass in the blog project

As I mentioned in the previous section, Sass is part of the development process and the browser cannot read it in its native format. Before you can deploy the project, you need to convert or compile the Sass files into regular CSS files. Normally this would require you to install a Ruby gem and you would have to manually compile your code before you can test it. Luckily for us, Harp.js actually has an Sass compiler built into it. So when you run the `harp compile` command to build your templates, it will also build your Sass files into regular CSS. I bet you're starting to like Harp even more after learning that.

Updating the blog project

Before we go any further, we need to make a few updates to our blog project to set it up for Sass. Head to your project directory and navigate to the CSS directory. In this directory, create a new file called `custom.scss`.

The file extension used for Sass files is `.scss`.



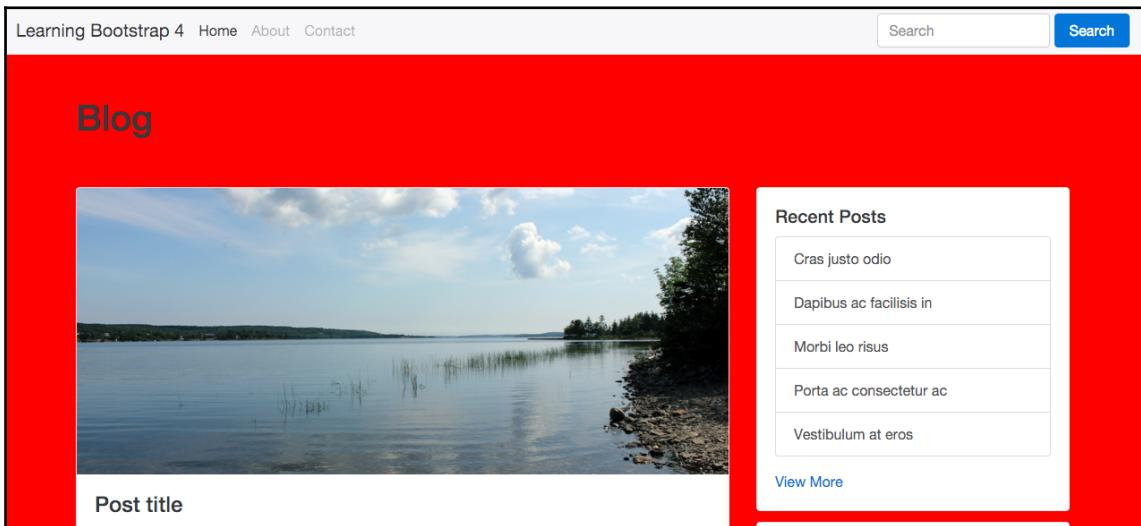
What we're doing here is creating a custom style sheet that we are going to use to overwrite some of the default Bootstrap look-and-feel CSS. To do this, we need to load this custom file after the Bootstrap framework CSS file in our layout file. Open up `_layout.ejs` in the root of the project directory and insert the following line of code after `bootstrap.min.css`. Both lines together should look like this:

```
<link rel="stylesheet" href="css/bootstrap.min.css">
<link rel="stylesheet" type="text/css" href="css/custom.css">
```

Note here that I'm using the `.css` file extension for `custom.css`. This is because, after the files are compiled, the template will be looking for the actual CSS file, not the Sass file. The critical part is just that the actual filenames match and that you use `.css` in the layout file. Before we go any further, let's test out our Sass file to make sure it is set up properly. Open up `custom.scss` in your text editor and add the following code:

```
body {
  background: red;
}
```

This is just a simple way to make sure that Sass is compiling to CSS and is being inserted into our layout. Compile your project and launch the server. If you've done everything correctly the background for your homepage should be red and look like this:



Hopefully this is what you're seeing and you can confirm you've set up your file correctly. Once you've successfully done this, delete the CSS we entered in the Sass file.



It's perfectly acceptable to write regular CSS in Sass files. Ideally, you want to combine regular CSS code with Sass syntax to take full advantage of the preprocessor.

Now that you've finished setting up your files, let's start to learn a little bit more about using Sass in your project.

Using variables

In Sass, variables are called by using the \$ sign character. If you're familiar with Less, the @ symbol is used for variables. So in that case, all you would need to do is use \$ instead of @. To write a variable, start with the \$ sign and then insert a descriptive keyword that can be anything you like. Here are a few examples of generic variable names:

```
$background-color  
$text-size  
$font-face
```

```
$margin
```

I've named these pretty generically and they actually match some CSS property names. This is a good idea and they are easy to reuse and make sense of if multiple developers are working on the same project. However, like I said, you can name your variables whatever you want. If you'd like to get more creative, you could name variables like this:

```
$matts-best-color  
$awesome-background-color  
$fantastic-font-face
```

These are extreme examples and it is advisable not to name your variables in this way. To you `$awesome-background-color` might mean red but to another person it could mean anything. It's always a good idea to name your variables in a descriptive manner that makes sense.

I've shown you how to write the variable name but the other side of the equation is the actual value for the variable. Let's add in some sample values for our first set of variable names:

```
$background-color: #fff;  
$text-size: 16px;  
$font-face: helvetica, sans-serif;  
$margin: 1em;
```

You write Sass variables the same way that you would write CSS properties. It's also worth noting that you should enter your variables at the very top of your style sheet so that they can be used in all of the CSS you write after them.

Using the variables in CSS

Now that we've written some variables, let's actually insert them into some CSS. After the variables in `custom.scss`, enter the following code:

```
body {  
  background: $background-color;  
  font-size: $text-size;  
  font-family: $font-face;  
  margin: $margin;  
}
```

So instead of using actual values for our CSS properties, we're using the variable names that we set up. This starts to get more powerful as we add more CSS. Let's reuse some of these variables:

```
body {  
  background: $background-color;  
  font-size: $text-size;  
  font-family: $font-face;  
  margin: $margin;  
}  
  
h1 {  
  font-size: 36px;  
  font-family: georgia, serif;  
}  
  
h2 {  
  font-size: $text-size;  
  font-family: $font-face;  
}
```

In this example, you can see a few things going on that I should explain:

- For the `<h1>` tag, I'm not using any variables. I'm using regular CSS property values.
- For the `<h2>` tag, I'm reusing the same variables to insert the `font-size` and `font-family` values.

As your style sheet grows longer, I'm sure you'll see the value in this strategy. For example, if I decide I want to change my `font-size` to `24px`, all I need to do is change the value for the `$text-size` variable to `24px`. I don't have to go through my entire style sheet and change all the values individually. These are just the basics of what you can do with variables. Let's look at a more advanced use case.

Using other variables as variable values

That might sound like a bit of a mouthful, but you can actually use a variable as the default value for another variable. A good example of where you might want to do this is when you are defining a color palette. You can switch the hex values to readable names and then use them for your other variables. This is much easier to scan and understand when you are debugging your code. Here's an example of what I mean:

```
$black: #000;  
$white: #fff;  
$red: #c00;  
  
$background-color: $white;  
$text-color: $black;
```

```
$link-color: $red;
```

Let me break down what is happening here for you:

- First I've created three color variables for `black`, `white`, and `red`
- Next I've created three CSS property variables for `background-color`, `text-color`, and `link-color`; the values for these CSS property variables are the color variables

Instead of using hex number values for the CSS property variables, I used a color keyword variable which is much easier to read and understand. That concludes the introduction to variables in Sass. Next we'll learn about importing different files into `custom.css` and using partials.

Importing partials in Sass

Just as you can do in `Harp.js`, you can use partials in Sass. If you've forgotten what a partial is, it's a little snippet of code that is saved into a different file and then imported into the main CSS theme or layout, in the case of Harp. This can be handy for making your CSS modular and easier to manage. For example, it would make a ton of sense to break every Bootstrap component into its own CSS file and then use the `@import` directive to bring them all into a single master theme, which is then included in your project. Let's go over an example of how you could do this for a single component. In your project, go to the `/css` directory and create a new sub-folder called `/components`. The full path should be:

```
/css/components
```

In the `/components` directory, create a new Sass file and name it `_buttons.scss`. Make sure you always insert an underscore at the start of the filename of a partial. The compiler will then ignore these files as the underscore means it is being inserted into another file. Enter the following at the top of the file as a marker:

```
/* buttons */
```

Save the buttons file and then open up `custom.scss` and add the following line of code to the file:

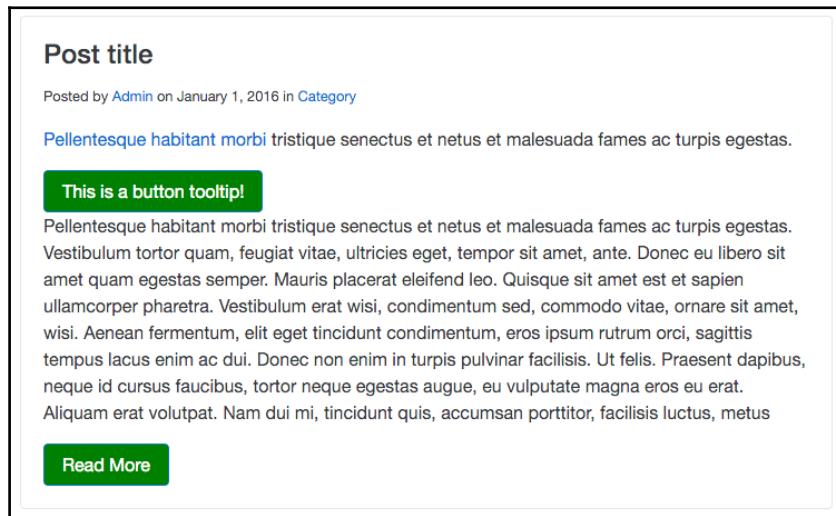
```
@import "components/_buttons.scss";
```

That line of code uses the `@import` rule, which will allow us to import the `_buttons.scss` file into our main theme file that we are calling `custom.scss`. As I've mentioned, the reason you need to do this is for maintainability. This makes the code much easier to read and to add/remove components, which is just another way of saying it makes it more modular.

Before we can test this out to make sure it works, we need to add some code to our `_buttons.scss` file. Let's add some simple CSS to change the primary button as an example:

```
.btn-primary {  
    background-color: green;  
}
```

After adding this code, save the file and do a `harp compile`. Then launch the server and check out the home page; the buttons will be green like this:



After testing that out, you may want to take that custom code out unless you want the buttons to remain green. That's just a simple example of how you can use partials to make your Bootstrap components more modular. I'll get into that topic in greater depth in a future chapter but for now we are going to focus on using Sass mixins.

Using mixins

Writing something in CSS, such as, for example, browser vendor prefixes, can be really tedious. Mixins allow you to group CSS declarations together so that you can reuse them through your project. This is a great because you can include the code for, say, a border-radius, using one line of code instead of multiple lines for each browser. To start, open up `custom.scss` and insert the following code at the top of the file:

```
@mixin border-radius($radius) {  
    -webkit-border-radius: $radius;  
    -moz-border-radius: $radius;  
    -ms-border-radius: $radius;  
    border-radius: $radius;  
}
```

Let's go over a few things that are happening here:

- A mixin is always started in Sass with the `@mixin` keyword
- Following that, you want to include the property name to target as well as set a variable, in this case `$radius`
- We then apply the `$radius` variable to each browser prefix instance

We've set up the mixin to handle the `border-radius` property but we still need to add the corner value to an element. Let's change the `border-radius` value for the default Bootstrap button. Open up `_buttons.scss` and insert the following code:

```
.btn {  
    @include border-radius(20px);  
}
```

Let me explain what is happening here:

- I'm targeting all Bootstrap buttons by inserting the `.btn` class
- Inserting the `@include` keyword will grab the `border-radius` mixin
- Lastly, I've provided a value of `20px`, which will make our buttons look really rounded on each end

Save your file, run the `harp compile` command, and then, when you view the project in the browser, it should look like this:

Post title

Posted by Admin on January 1, 2016 in Category

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

This is a button tooltip!

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci, sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar facilisis. Ut felis. Praesent dapibus, neque id cursus faucibus, tortor neque egestas augue, eu vulputate magna eros eu erat. Aliquam erat volutpat. Nam dui mi, tincidunt quis, accumsan porttitor, facilisis luctus, metus

[Read More](#)

That concludes a fairly simple example of how to use mixins in Bootstrap 4. You can use them for many other reasons but replacing CSS3 vendor prefixes is one of the most common and useful. Next we'll cover a slightly more complicated topic in Sass, which is the use of operators.

How to use operators

Sass allows you to perform basic math operations in CSS, which is useful for a number of reasons. First of all, you can use the following operators `+`, `-`, `*`, `/`, and `%`. To give you an understanding of how you can use operators in CSS, let's learn how to convert a pixel-based grid into percentages. We'll create two columns in pixels and then use some Sass to convert them to percentages. Open up `custom.scss` and insert the following code:

```
.left-column {  
    width: 700px / 1000px * 100%;  
}  
  
.right-column {  
    width: 300px / 1000px * 100%;  
}
```

Now, I've created two columns here. The `.left-column` class will have a width of 70% after we compile this Sass operator. The `.right-column` class will have a width of 30%. So if we add those together we'll get roughly a three-quarter layout with a larger column on the left and a smaller column on the right. Run a `harp compile` command to build this code and then open up `custom.css` in the `/www/css` folder. There you should find the following code:

```
.left-column {  
    width:70%;  
}  
  
.right-column {  
    width:30%;  
}
```

As you can see, our Sass operators have been converted into regular percentage values. That's just one way you can use operators in Sass; I'd encourage you to play around more with them. Next we're going to learn how to set up a library of Sass variables that you can use to create a Bootstrap theme.

Creating a collection of variables

One of the main things you'll want to do when using Sass in Bootstrap is to create a library of global variables that can be used throughout your theme. Think of things such as colors, backgrounds, typography, links, borders, margins, and padding. It's best to only define these common properties once and then you can reuse them through different components. Before we go too far, we need to create a new `.scss` file. Open up your text editor, create a new file, and call it `_variables.scss`. Save that file to the `/css/components` directory. For now, you can just leave it blank.

Importing the variables to your custom style sheet

Now that we've created the variables Sass file, we need to import it into our custom style sheet. Open up `custom.css` in your text editor and paste the following line of code at the top of the file:

```
@import "components/_variables.scss";
```

It's important to note that this file must be at the top of your custom style sheet file. The variables will cascade through all the code that follows them so they must load first. Let's start filling out our variables file with a color palette.

Adding a color palette

Save the custom style sheet and then go back to the variables file. Let's start by inserting a color palette into the variables file like this:

```
$red: #e74c3c;
$red2: #c0392b;
$blue: #3498db;
$blue2: #2980b9;
$green: #2ecc71;
$green2: #27ae60;
$yellow: #f1c40f;
$yellow2: #f39c12;
$purple: #9b59b6;
$purple2: #8e44ad;
$white: #fff;
$off-white: #f5f5f5;
$grey: #ccc;
$dark-grey: #333;
$black: #000;
```

As you can see, I've set up a palette of several colors that I'll use through my components and later my theme. Here are a few key points to keep in mind:

- It's good to have two variations for your key colors. This comes in handy for a component such as a button where \$red would be the static color and \$red2 would be the hover or active color for the button.
- I'm guessing you can already see how using variable names such as \$purple is much more readable than hex values in a long style sheet.

Adding some background colors

The next thing you should add to your collection of variables is background colors. As we move through this variables file, we're going to create a variable for all properties that get used over and over again in our style sheet.

Add the following background color variables to the file:

```
$primary-background: $white;  
$secondary-background: $off-white;  
$inverse-background: $black;
```

Let me explain, as best practice, how I have set this up:

- First of all, I'm using the color variables we just set up as the values for our new background color variables. This keeps things simple and it also allows you to change the color and have it cascade through all your other variables. This is a great time-saving tip.
- At the very least, it's a good idea to define a `primary`, `secondary`, and `inverse` background color variable. Note how I'm reusing the same language here that Bootstrap uses. This is a good practice to follow. Feel free to define additional background colors if you think you'll need them in your project.

Setting up the background color variables is pretty simple. Next let's set up our base typography variables.

Setting up variables for typography

The next section of variables we are going to set up is for the base typography styles. Insert the following code after the background colors:

```
$body-copy: helvetica, arial, verdana, sans-serif;  
$heading-copy: helvetica, arial, verdana, sans-serif;  
$base-font-size: 16px;  
$font-size: 1em;  
$base-line-height: 1.75;
```

Let me explain why I'm setting the following variables for the typography:

- For consistency, it's good to have a body and heading typeface. In this case, I'm using the same font stack for both but you could easily change the heading variable to something else. As you are coding your CSS, it's really easy to think of the `font-family` in either the body or heading version, compared with trying to remember the entire font stack for each, which also involves much more typing.
- For the `$base-font-size` variable, we are going to use a pixel value. This is one of the only places you'll see pixels and it's set to the base em size that everything else will work off. Remember that ems are a relative sizing unit, so if you ever want to make all your components a little bigger or smaller, you can just tweak this one pixel value.

- We also need a `$font-size` variable, which will be set to `1em`. This is a base unit and it can easily be changed in other selectors by using Sass operators. The reason we set it to `1em` is because it simply makes the math easy to do.
- Finally, I set the `$base-line-height` to `1.75` because I like a little extra line spacing in my copy. You could choose to leave this out if you are fine with the Bootstrap default, which is closer to `1.5`.

Now that we've set up our typography variables, let's move on to coding our text colors.

Coding the text color variables

As with the background colors, we need to set up some common color styles for text, as well as defining some colors for base HTML tags such as `<pre>` and `<code>`. Insert the following markup after the typography variables in the file:

```
$primary-text: $black;  
$light-text: $grey;  
$loud-text: $black;  
$inverse-text: $white;  
$code-text: $red;  
$pre-text: $blue;
```

Let me break down how each variable is set up:

- As in the background color variables, we are using a variable name for the value of our text color variables. I've included a variable called `$primary-text` and set it to black, following the same naming convention that was previously established.
- I've added `$light-text` and `$loud-text` variables so we can easily apply lighter or darker text throughout our components.
- I've also included an `$inverse-text` variable to be used with the corresponding background color.
- Finally, I've set up default colors for the `<pre>` and `<code>` tags, which we will use to overwrite the default colors so they match our theme and color palette.

That finishes off the color variables that I recommend setting up. Feel free to add more if you have other uses you want to cover. Next we'll continue with some text colors by adding links.

Coding variables for links

An extension of basic text colors will be colors for links in our project. Go ahead and add the following code after the text colors in the file:

```
$primary-link-color: $purple;  
$primary-link-color-hover: $purple2;  
$primary-link-color-active: $purple2;
```

In this case, I've decided to only define a primary link color to keep things simple. In your own projects, you will likely want to come up with a couple more variations.

- For the static link color, I'm using the `$purple` color variable.
- For the hover and active states of the primary link, I'm using `$purple2`. As I previously mentioned, this is an example of why it's a good idea to have two variations of each color in your palette.

Like I said, I've kept the link variables simple. It's nice to try and keep your set of variables as compact as possible. If you have too many then it starts to defeat the purpose of using them as it will be harder to remember them in your code. Next let's cover the variables we should set up for borders.

Setting up border variables

Another CSS property that gets used often is borders. That makes it a great candidate for Sass variables. Insert the following code after the link colors in the file:

```
$border-color: $grey;  
$border-size: 1px;  
$border-type: solid;  
$border-focus: $purple;
```

Let me explain why I've set up the border variables in this manner:

- When you are deciding on a value for `$border-color`, you should pick a color that you think will get used the most often in your components. Something like `$grey` is always a safe bet in most designs.
- As with the color value, you should set the `$border-size` to the most common border size you anticipate using. It's also a good idea to set this to `1px` because you can easily do the math to apply a Sass operator if you want a thinner or thicker border.

- Again for the `$border-type`, set it to the value you will use the most, which is probably going to be solid.
- Finally, set up a common `$border-focus` color. This is primarily used in form inputs once they are active. It's a good idea to pick a contrasting color for this variable so it really stands out when the input is in focus.

That concludes all the border variables I would recommend including. Next let's include some basic layout variables.

Adding variables for margin and padding

For consistent spacing throughout your designs, it's a good idea to use variables for `margin` and `padding` so that you can standardize on size. These properties are also used often so it's smart to make them variables that can be reused. Add the following code after the border markup:

```
$margin: 1em;  
$padding: 1em;
```

All I'm doing here is setting a base size (for both `padding` and `margin`) `1em`. Again, it's a good idea to set both of these to `1em` because it is easy to do the math if you want to use Sass operators to increase or decrease the values of specific components. Those are the last variables that I would recommend adding to your variables file. However, we should add at least one mixin to the file before we are finished.

Adding mixins to the variables file

Since mixins will also be used through a number of your components, you should define them in this variables file. Then they will be available to all the CSS code that follows them in the custom theme file. At the very least, I would recommend setting up a mixin for `border-radius`, which I will show you how to do next. You may also want to include additional mixins for other CSS3 features.

Coding a border-radius mixin

We talked a little bit about mixins earlier but let's review them again now that we are actually applying them to our project. Insert the following code after the layout variables in your file:

```
@mixin border-radius($radius) {  
    -webkit-border-radius: $radius;  
    -moz-border-radius: $radius;  
    -ms-border-radius: $radius;  
    border-radius: $radius;  
}
```

In Less, it is possible to set a global value for all your `border-radius` in a mixin. However, with Sass you have to set up the above formula but then on the actual selectors that follow you have to set the actual `border-radius` value. An example of that would look like this:

```
.my-component {  
    @include border-radius(5px);  
}
```

In this example, I've added the `border-radius` mixin to a CSS class called `.my-component`. The component will have a `border-radius` of `5px` applied to it. You will need to repeat this step on any CSS class or component where you want to apply the `border-radius` mixin. That concludes our variables Sass file. We went over a bunch of code there, so let's see what it all looks like together. I've also included some CSS comments in the following code to help remind you what each section does:

```
/* variables */  
  
/* color palette */  
$red: #e74c3c;  
$red2: #c0392b;  
$blue: #3498db;  
$blue2: #2980b9;  
$green: #2ecc71;  
$green2: #27ae60;  
$yellow: #f1c40f;  
$yellow2: #f39c12;  
$purple: #9b59b6;  
$purple2: #8e44ad;  
$white: #fff;  
$off-white: #f5f5f5;  
$grey: #ccc;  
$dark-grey: #333;  
$black: #000;
```

```
/* background colors */
$primary-background: $white;
$secondary-background: $off-white;
$inverse-background: $black;

/* typography */
$body-copy: helvetica, arial, verdana, sans-serif;
$heading-copy: helvetica, arial, verdana, sans-serif;
$base-font-size: 16px;
$font-size: 1em;
$base-line-height: 1.75;

/* text colors */
$primary-text: $black;
$light-text: $grey;
$cloud-text: $black;
$inverse-text: $white;
$code-text: $red;
$pre-text: $blue;

/* links */
$primary-link-color: $purple;
$primary-link-color-hover: $purple2;
$primary-link-color-active: $purple2;

/* border */
$border-color: $grey;
$border-size: 1px;
$border-type: solid;
$border-focus: $purple;

/* layout */
$margin: 1em;
$padding: 1em;

/* border-radius mixin */
@mixin border-radius($radius) {
    -webkit-border-radius: $radius;
    -moz-border-radius: $radius;
    -ms-border-radius: $radius;
    border-radius: $radius;
}
```

Now that we have all our variables and mixins set up, let's go ahead and start to learn how to apply them. We'll continue to build on the button example we started earlier by extending it into a custom look and feel.

Customizing components

Let's first start by customizing a single component; later on I'll talk about creating a theme where you customize all the components in Bootstrap. To get started, we'll build on the button component we started to work on earlier. In this next step we are going to expand on the CSS we have added to fully customize the component. What you want to do is overwrite all the CSS classes and properties that you want to change. In some cases, this might only be a few things but in other scenarios you may want to change quite a bit.

Customizing the button component

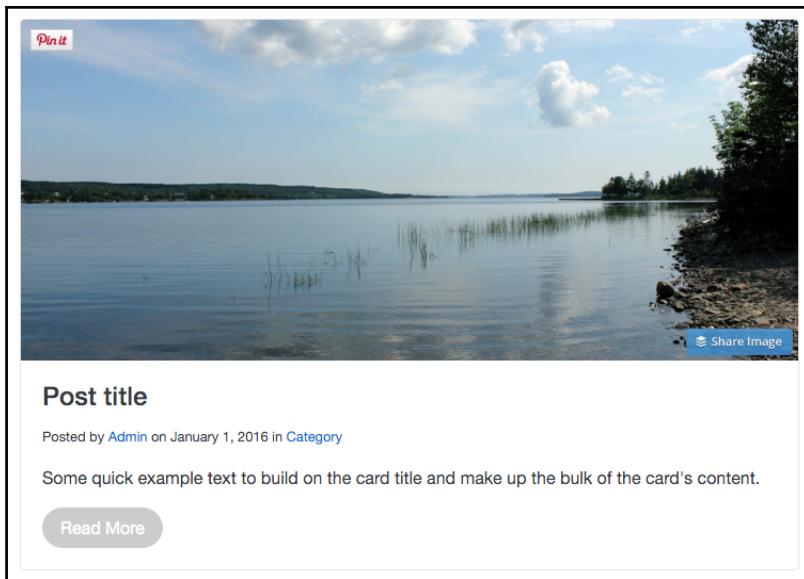
To start, open up `_buttons.scss` located in `/css/components` in our project directory. The first thing we need to customize is the base `.btn` CSS class. Once we have applied some changes there, we'll add more CSS to control the look and feel of the different button variations. Insert the following CSS at the top of the file for the base button class:

```
.btn {  
  background-color: $grey;  
  border-color: $grey;  
  
  @include border-radius(20px);  
}
```

To keep things simple, I'm only going to overwrite a few properties. You're totally free to get more creative and change additional properties to make your buttons look different from the Bootstrap default link. Let's break down what I've done:

- First I've set the `background-color` and `border-color` to use the `$grey` from our color palette. This is a good time to point out that if you want to do a full theme you need to overwrite all the Bootstrap default colors on all components to match your color palette.
- Next I've inserted the `border-radius` mixin and given it a value of `20px`. This will make the buttons really rounded. I'm going for this look so you can clearly see that the button has been customized.

Once you have saved these changes, go to the terminal and run the `harp compile` command from the root of the project directory. Then fire up the server and open the home page of the project that has a bunch of buttons on it. Your buttons should now look like this:



Now that might not look too useful, but it's important that we customize the base `.btn` class first; now we'll continue building the component out by applying our color palette to all of the different button variations.

Extending the button component to use our color palette

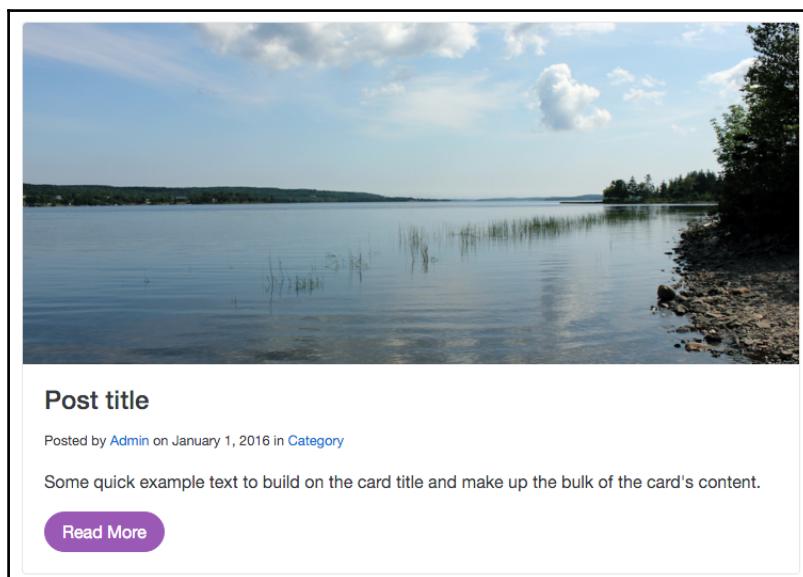
In this next section, we will extend the button component further by applying our color palette to all the different Bootstrap button variations. Before we get to all the different button types, let's start by customizing the `.btn-primary` variation. Enter the following code in the `_buttons.scss` file after the base `.btn` styles:

```
.btn-primary {  
    background-color: $purple;  
    border-color: $purple;  
}  
  
.btn-primary:hover,  
.btn-primary:active {  
    background-color: $purple2;  
    border-color: $purple2;  
}
```

There are a few different things going on so let's review them all:

- There are two sections of CSS for each button variation. The first is the static state of the button. The second is the hover and active states of the button.
- For the static state we use the `.btn-primary` class and insert the `background-color` and `border-color` properties. I want to make my primary button purple so I've inserted the `$purple` Sass variable to overwrite the Bootstrap default color.
- For the other states, we have `.btn-primary:hover` and `.btn-primary:active`. In this case, I'm using the second purple color variable which is `$purple2`. On the hover or active button there will be a slightly darker shade of purple.

Save the file, run a `harp compile` in the terminal, and then open up the home page in your browser. If everything was coded correctly, your buttons should now look like this:



As you can see, the primary button is now purple! It's as simple as that; you can start to apply a custom look and feel to the button component. Let's build out the rest of the button color variations by entering the following code into the `_buttons.scss` file:

```
.btn-secondary {  
  background-color: $off-white;  
  border-color: $off-white;
```

```
}

.btn-secondary:hover,
.btn-secondary:active {
    background-color: $grey;
    border-color: $grey;
}

.btn-success {
    background-color: $green;
    border-color: $green;
}

.btn-success:hover,
.btn-success:active {
    background-color: $green2;
    border-color: $green2;
}

.btn-info {
    background-color: $blue;
    border-color: $blue;
}

.btn-info:hover,
.btn-info:active {
    background-color: $blue2;
    border-color: $blue2;
}

.btn-warning {
    background-color: $yellow;
    border-color: $yellow;
}

.btn-warning:hover,
.btn-warning:active {
    background-color: $yellow2;
    border-color: $yellow2;
}

.btn-danger {
    background-color: $red;
    border-color: $red;
}

.btn-danger:hover,
.btn-danger:active {
```

```
background-color: $red2;
border-color: $red2;
}
```

That's a bunch of code but it should be fairly easy to understand. I've simply followed the same steps I completed for the primary button for every other button variation. Along the way, I've replaced the default Bootstrap color values with our custom color palette. Once you're done, all of your buttons should now look like this:



We've now successfully customized the entire button component. As I mentioned earlier, there may be additional things you might want to do to the buttons. However, at the very least, we've done enough to show how you can make the component your own. The next step in this process is to go through every Bootstrap component one by one and apply the same customization process. We call this writing your own Bootstrap theme.

Writing a theme

Creating your own Bootstrap theme is a bit of an undertaking. The good news is that once you've done it you can reuse a ton of the code for future themes. That's where the real power in making your code modular comes into play. Instead of starting over from scratch each time, you can reuse old code and just extend it. In the last section, we learned how to customize the button component that was the start of our own theme. Let's first start by looking at some common Bootstrap components that you'll want to customize for your own themes.

Common components that need to be customized

There are many ways that you can theme Bootstrap. In some cases, you may only need to customize a few components to get a unique look and feel going. However, you may want to do a more thorough theming process so that your theme doesn't resemble the default Bootstrap look at all. In this section, let's start by listing some of the common components you will most likely want to customize.

Next we'll go through the process of writing the code to customize a few so you get an idea as to how it works. Here's a list of components that I would recommend customizing:

- Buttons
- Drop-downs
- Alerts
- Navbar
- Typography
- Tables

This list is just a starting place. If you want to create a unique theme, you should really try to customize all Bootstrap components. At the very least, you should change them to use your custom color palette, typography, and layout styles. We've already covered buttons so let's jump into customizing the drop-down component, which is an extension of the button.

Theming the drop-down component

The drop-down component requires a medium-sized amount of customization so it's a good starting place to get an idea of what is involved in this process. It also builds on the code we wrote for the button so it's a natural second step. It's important to note that some components will require a good amount of CSS to customize them, while others will only need a little bit. Let's start by creating a new Sass file for drop-downs. From your project folder, create a new file called `_dropdown.scss` in the `css/components` directory. You can leave the file blank for now, just save it.

Once you've created the new Sass file for the drop-down component, we need to import it into our main theme is called `custom.scss`. Open up the custom style sheet in your text editor and insert the following line of code after the `@import` for the button component:

```
@import "components/_dropdown.scss";
```

Now we are ready to start coding our custom drop-down styles. Open up `_dropdown.scss` in your text editor and let's insert this first section of CSS:

```
.dropdown-menu {  
  color: $primary-text;  
}
```

As with the buttons in the previous section, I'm only going to change the most basic properties to demonstrate how you can customize the component. Feel free to customize additional properties to get a more unique look and feel.

Let's break down what is happening here. The drop-down component is made up of the base `.dropdown-menu` CSS class. This controls how the menu will look. Here I've simply changed the text color to use for the `$primary-text` variable.

We also need to do some work on the list of links that appear in our drop-down menu. Insert the following CSS after the first section you just entered:

```
.dropdown-item:focus,  
.dropdown-item:hover {  
    color: $primary-text;  
    background-color: $secondary-background;  
}
```

Let me break down what is happening here:

- These CSS classes control the hover and focus states for each list item in our drop-down menu. Again, I've set it to use our `$primary-text` font color.
- When you hover on a list item, the background color changes. I've changed that background color to use our `$secondary-background` color variable. In this case you should use the background color variable, not a customized color variable. The reason for this is it's easier to keep track of what background colors you are using as you progress through the writing of your code.

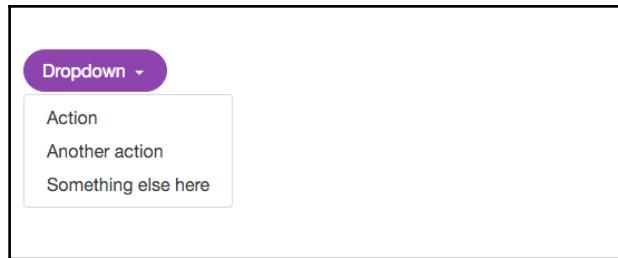
The last thing we need to do is update the actual drop-down button trigger with some additional code. Enter the last part of CSS into the file:

```
.open > .btn-primary.dropdown-toggle:focus {  
    background-color: $purple2;  
    border-color: $purple2;  
}
```

When the drop-down button trigger is clicked the `.open` CSS class will dynamically be inserted into the HTML code. This initiates a unique variation on the button class, a drop-down toggle focus. That may sound complicated but what you need to know is that you need to set this selector to our `$purple2` color so it matches the rest of the button.

I've overwritten the `background-color` and `border-color` properties to use `$purple2` from our color palette.

That's it, the drop-down component has now been themed to match our look and feel. If you preview it in the browser it should look like this when the menu is open:



Now that we've finished with the drop-down component let's move on to learning how to theme the alerts component.

Customizing the alerts component

The alerts component in Bootstrap is fairly easy to theme. As with the button component, it comes in a few variations. Let's start by coding up the CSS for the default color method. Create a new file called `_alerts.scss` and save it to the `css/components` directory. Don't forget to import it into `custom.scss` with the following line of code:

```
@import "components/_alerts.scss";
```

Once you've set up the file, let's get started with the code for the success alert component:

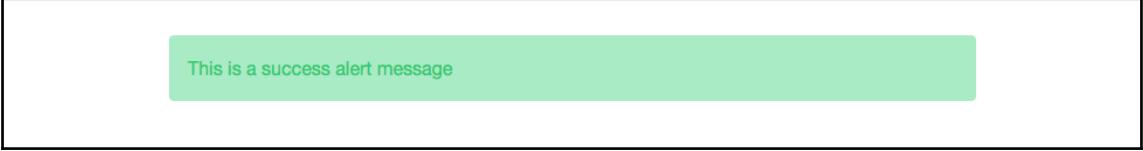
```
.alert-success {  
  color: $green;  
  background-color: lighten( $green, 30% );  
  border-color: lighten( $green, 30% );  
}
```

What you're now seeing should start to look familiar. However, I have introduced something new that I need to explain:

- This is the success alert so it should be green in color. The first thing I've done is change the text color to use the green from our palette with the `$green` variable.
- For the `background-color` and `border-color` properties, I'm using something new, a Sass function. In this case, I want a green color that is slightly lighter than my text. Instead of introducing another green color variable, I can use a Sass function to lighten the base `$green` variable color.

- To create the function, you use the `lighten` keyword. Inside the brackets you need to include the variable name you want to target, in this case `$green`, and finally include a percentage value for how much to lighten it by. This is a nice little trick to save you having to create more variables.

Once you code this up it should look like this in the browser:

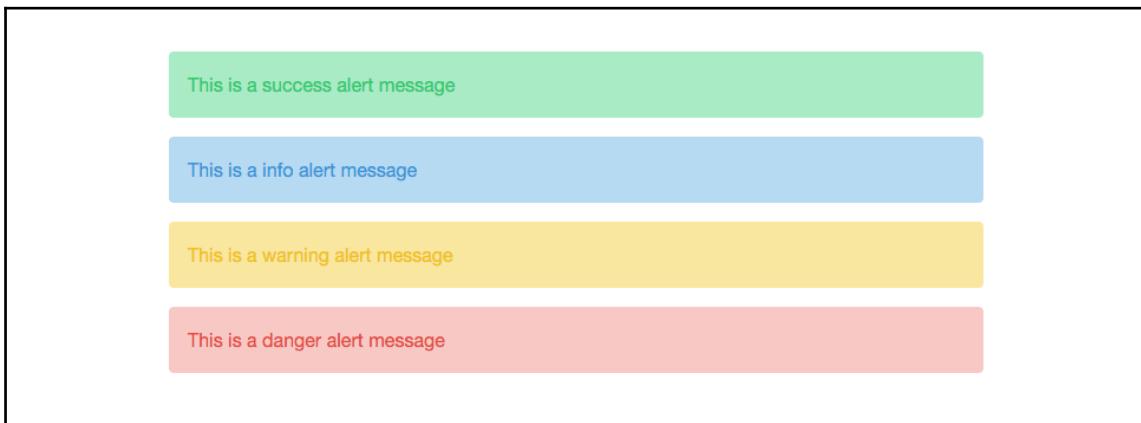


This is a success alert message

As you can see, we are using the green color values from our color palette. Let's continue and customize the colors for the rest of the alert bar variations. Enter the following code into the `_alerts.scss` file:

```
.alert-info {  
  color: $blue;  
  background-color: lighten( $blue, 30% );  
  border-color: lighten( $blue, 30% );  
}  
  
.alert-warning {  
  color: $yellow;  
  background-color: lighten( $yellow, 30% );  
  border-color: lighten( $yellow, 30% );  
}  
  
.alert-danger {  
  color: $red;  
  background-color: lighten( $red, 30% );  
  border-color: lighten( $red, 30% );  
}
```

The other alerts follow the same pattern as the success version. They should look like this in the browser when you are done:



As you can see, the alerts are now using our color palette. Let's move on to the last component that I will show you how to customize, which is typography.

Customizing the typography component

The typography component isn't difficult to customize. We'll build off the base variables we set up to apply them to the appropriate HTML tags. As we did with our other components, start with creating a new file called `_typography.scss` and save it to the `css/components` directory. Once you do this, import the file into `custom.scss` with the following line of code:

```
@import "components/_typography.scss";
```

Let's start customizing the type by applying some styles to the base header tags:

```
h1, h2, h3, h4, h5, h6 {  
  font-family: $heading-copy;  
  color: $primary-text;  
}
```

Here I've simply used the `$heading-copy` variable and applied it to all the HTML heading tags. This will allow our custom heading typeface to be used for all headers. I've also added the `$primary-text` variable so that our headers are using the correct text color. Next let's take a look at a few miscellaneous text styles that you will likely want to overwrite:

```
small {  
    color: $light-text;  
}  
  
pre {  
    color: $pre-text;  
}  
  
code {  
    color: $code-text;  
}
```

As we did with our base variables, I'm now applying some of them on actual selectors. Let's break it down:

- For the `<small>` HTML tag, I want it to look more subtle so I've set the text color to use the `$light-text` variable.
- I purposely set up color text variables for the HTML `<pre>` and `<code>` tags. I've now applied the `$pre-text` and `$code-text` variables to these tags.

That covers some of the basic typography styles you're going to want to customize. There are more you could add but I will let you explore these on your own. That also goes for all the Bootstrap components. We have only scratched the surface of the level of customizing you can do for your Bootstrap theme. However, I think I've given you a good introduction to what you need to do for coding your own Bootstrap themes.

Summary

That brings this chapter to a close. We've covered a ton of new content in this chapter including: the basics of Sass, how to use Sass in Bootstrap, how to create a library of Sass variables, how to apply those variables to customize Bootstrap components, and, finally, how to start writing your own Bootstrap theme. In the final chapter, I'll provide some advice on moving from Bootstrap version 3 to 4.

9

Migrating from Version 3

Version 4 of Bootstrap is a major update. Almost the entire framework has been rewritten to improve code quality, add new components, simplify complex components, and make the tool easier to use overall. We've seen the introduction of new components such as Cards and the removal of a number of basic components that weren't heavily used. In some cases, Cards present a better way of assembling a layout than a number of the removed components. Let's jump into this chapter by showing some specific class and behavioral changes to Bootstrap in version 4.

Browser support

Before we jump into the component details, let's review the new browser support. If you are currently running on version 3 and support some older browsers, you may need to adjust your support level when migrating to Bootstrap 4. For desktop browsers, Internet Explorer version 8 support has been dropped. The new minimum Internet Explorer version that is supported is version 9.

In terms of mobiles, iOS version 6 support has been dropped. The minimum iOS supported is now version 7. The Bootstrap team has also added support for Android v5.0 Lollipop's browser and **WebView**. Earlier versions of the Android Browser and WebView are not officially supported by Bootstrap.

Big changes in version 4

Let's continue by going over the biggest changes to the Bootstrap framework in version 4.

Switching to Sass

Perhaps the biggest change in Bootstrap 4 is the switch from Less to Sass. This will also likely be the biggest migration job you will need to take care of. The good news is you can use the sample code we've created in the book as a starting place. Luckily, the syntax for the two CSS pre-processors is not that different. If you haven't used Sass before, there isn't a huge learning curve that you need to worry about. Let's cover some of the key things you'll need to know when updating your stylesheets for Sass.

Updating your variables

The main difference in variables is the symbol used to denote one. In Less we use an @ symbol for our variables, while in Sass you use the \$ symbol. Here are a couple of examples for you:

```
/* LESS */
@red: #c00;
@black: #000;
@white: #fff;

/* SASS */
$red: #c00;
$black: #000;
$white: #fff;
```

As you can see, that is pretty easy to do. A simple find and replace should do most of the work for you. However, if you are using @import in your stylesheets, make sure this retains an @ symbol.

Updating @import statements

Another small change in Sass is how you import different stylesheets using the @import keyword. First, let's take a look at how you do this in Less:

```
@import "components/_buttons.less";
```

Now let's compare how we do this using Sass:

```
@import "components/_buttons.scss";
```

As you can see, it's almost identical. You just need to make sure you name all your files with the `.scss` extension. Then update your file names in the `@import` to use `.scss` and not `.less`.

Updating mixins

One of the biggest differences between Less and Sass is mixins. Here we'll need to do a little more heavy lifting when we update the code to work for Sass. First, let's take a look at how we would create a border-radius, or round corner, mixin in Less:

```
.border-radius (@radius: 2px) {  
  -moz-border-radius: @radius;  
  -ms-border-radius: @radius;  
  border-radius: @radius;  
}
```

In Less, all elements that use the `border-radius` mixin will have a border radius of 2px. That is added to a component, like this:

```
button {  
  .border-radius  
}
```

Now let's compare how you would do the same thing using Sass. Check out the mixin code:

```
@mixin border-radius($radius) {  
  -webkit-border-radius: $radius;  
  -moz-border-radius: $radius;  
  -ms-border-radius: $radius;  
  border-radius: $radius;  
}
```

There are a few differences here that you need to note:

- You need to use the `@mixin` keyword to initialize any mixin
- We don't actually define a global value to use with the mixin

To use the mixin with a component, you would code it like this:

```
button {  
  @include border-radius(2px);  
}
```

This is also different from Less in a few ways:

- First, you need to insert the `@include` keyword to call the mixin
- Next, you use the mixin name you defined earlier, in this case, `border-radius`
- Finally, you need to set the value for the `border-radius` for each element, in this case, `2px`

Personally, I prefer the Less method as you can set the value once and then forget about it. However, since Bootstrap has moved to Sass, we have to learn and use the new syntax. That concludes the main differences that you will likely encounter. There are other differences and if you would like to research them more, I would check out this page:

<http://sass-lang.com/guide>.

Additional global changes

The change to Sass is one of the bigger global differences in version 4 of Bootstrap. Let's take a look at a few others you should be aware of.

Using REM units

In Bootstrap 4, px has been replaced with rem for the primary unit of measure. If you are unfamiliar with rem it stands for **root em**. Rem is a relative unit of measure where pixels are fixed. Rem looks at the value for font-size on the root element in your stylesheet. It then uses your value declaration, in rem, to determine the computer pixel value. Let's use an example to make this easier to understand:

```
html {  
  font-size: 24px;  
}  
  
p {  
  font-size: 2rem;  
}
```

In this case, the computed font-size for the `<p>` tag would be **48px**. This is different from the **em** unit because **ems** will be affected by wrapping elements that may have a different size. However, **rem** takes a simpler approach and just calculates everything from the root HTML element. It removes the size cascading that can occur when using **ems** and nested, complicated elements. This may sound confusing, but it is actually easier to use em units. Just remember your root font-size and use that when figuring out your rem values.

What this means for migration is that you will need to go through your stylesheet and change any px or em values to use ems. You'll need to recalculate everything to make sure it fits the new format if you want to maintain the same look and feel for your project.

Other font updates

The trend for a long while has been to make text on a screen larger and easier to read for all users. In the past, we used tons of small typefaces that might have looked cool but were hard to read for anyone visually challenged. To that end, the base font-size for Bootstrap has been changed from **14px** to **16px**. This is also the standard size for most browsers and makes the readability of text better. Again, from a migration standpoint, you'll need to review your components to ensure they still look correct with the increased font size. You may need to make some changes if you have components that were based on the **14px** default font-size in Bootstrap 3.

New grid size

With the increased use of mobile devices, Bootstrap 4 includes a new smaller grid tier for small screen devices. The new grid tier is called extra small and is configured for devices under **480px** in width. For the migration story this shouldn't have a big effect. What it does do is allow you a new breakpoint if you want to further optimize your project for smaller screens.

That concludes the main global changes to Bootstrap that you should be aware of when migrating your projects. Next, let's take a look at components.

Migrating components

With the release of Bootstrap 4, a few components have been dropped and a couple of new ones have been added. The most significant change is the new Cards component. Let's start by breaking down this new option.

Migrating to the Cards component

With the release of the Cards component, the Panels, Thumbnails, and Wells components have been removed from Bootstrap 4. Cards combines the best of these elements into one and even adds some new functionality that is really useful. If you are migrating from a Bootstrap 3 project, you'll need to update any Panels, Thumbnails, or Wells to use the Cards component instead. Since the markup is a bit different, I would recommend just removing the old components altogether, and then recoding them using the same content as Cards.

Using icon fonts

The GLYPHICONS icon font has been removed from Bootstrap 4. I'm guessing this is due to licensing reasons as the library was not fully open source. If you don't want to update your icon code, simply download the library from the GLYPHICONS website at:

<http://glyphicon.com/>

The other option would be to change the icon library to a different one such as Font Awesome. If you go down this route, you'll need to update all of your `<i>` tags to use the proper CSS class to render the icons. There is a quick reference tool that will allow you to do this called **GlyphSearch**. This tool supports a number of icon libraries and I use it all the time. Check it out at: <http://glyphsearch.com/>.

Those are the key components you need to be aware of. Next let's go over what's different in JavaScript.

Migrating JavaScript

The JavaScript components have been totally rewritten in Bootstrap 4. Everything is now coded in ES6 and compiled with Babel, which makes it easier and faster to use. On the component side, the biggest difference is the Tooltips component. The Tooltip is now dependant on an external library called **Tether**, which you can download from:

<http://github.hubspot.com/tether/>.

If you are using Tooltips, make sure you include this library in your template. The actual markup looks to be the same for calling a Tooltip but you must include the new library when migrating from version 3 to 4.

Miscellaneous migration changes

Aside from what I've gone over already, there are a number of other changes you need to be aware of when migrating to Bootstrap 4. Let's go through them all below.

Migrating typography

The `.page-header` class has been dropped from version 4. Instead, you should look at using the new display CSS classes on your headers if you want to give them a heading look and feel.

Migrating images

If you've ever used responsive images in the past, the class name has changed. Previously, the class name was `.image-responsive` but it is now named `.image-fluid`. You'll need to update that class anywhere it is used.

Migrating tables

For the table component, a few class names have changed and there are some new classes you can use.

If you would like to create a responsive table, you can now simply add the class `.table-responsive` to the `<table>` tag. Previously, you had to wrap the class around the `<table>` tag. If migrating, you'll need to update your HTML markup to the new format.

The `.table-condensed` class has been renamed to `.table-sm`. You'll need to update that class anywhere it is used.

There are a couple of new table styles you can add called `.table-inverse` or `.table-reflow`.

Migrating forms

Forms are always a complicated component to code. In Bootstrap 4, some of the class names have changed to be more consistent. Here's a list of the differences you need to know about:

- `control-label` is now `.form-control-label`
- `input-lg` and `.input-sm` are now `.form-control-lg` and `.form-control-sm`
- The `.form-group` class has been dropped and you should instead use `.form-control`

You likely have these classes throughout most of your forms. You'll need to update them anywhere they are used.

Migrating buttons

There are some minor CSS class name changes that you need to be aware of:

- `btn-default` is now `.btn-secondary`
- The `.btn-xs` class has been dropped from Bootstrap 4

Again, you'll need to update these classes when migrating to the new version of Bootstrap. There are some other minor changes when migrating on components that aren't as commonly used. I'm confident my explanation will cover the majority of use cases when using Bootstrap 4. However, if you would like to see the full list of changes, please visit:

<http://v4-alpha.getbootstrap.com/migration/>.

Summary

That brings the final chapter of the book to a close! Thank you for taking the time to read it and I hope I've successfully brought you up-to-speed on how to use Bootstrap 4.

Module 3: Mastering Bootstrap 4

Chapter 1: Revving Up Bootstrap

6

Introducing our demo project	7
What Bootstrap 4 Alpha 4 has to offer	10
Layout	10
Content styling	10
Components	11
Mobile support	11
Utility classes	12
Cross-browser compatibility	12
Sass instead of Less	12
From pixel to root em	13
No more support for Internet Explorer 8	14
A new grid tier	14
Bye-bye GLYPHICONS	14
Bigger text: no more panels, wells, and thumbnails	15
New and improved form input controls	15
Customization	16
Setting up our project	16
Summary	24

Chapter 2: Making a Style Statement

25

The grid system	25
Containers	27
container	27
container-fluid	31
Rows	33
Columns	34
Nesting	38
Pulling and pushing	41
Offsetting	42
Image elements	44
Responsive images	46
Image modifiers	48
Responsive utilities	50

Helper classes	54
Context	54
Centering and floating	55
Toggling visibility	58
Text alignment and transformation	59
Summary	60
Chapter 3: Building the Layout	61
Splitting it up	62
Adding Bootstrap components	65
Jumbotron	65
Tabs	68
Carousel	73
Cards	77
Navbar	79
Styling	87
Summary	91
Chapter 4: On Navigation, Footers, Alerts, and Content	92
Fixating the navbar	93
Improving navigation using Scrollspy	94
Customizing scroll speed	96
Icons	99
A note on icons in Bootstrap 3	103
Using and customizing alerts	103
Creating a footer	108
Creating and customizing forms	111
Form validation	118
Progress indicators	119
Adding content using media objects	120
Figures	122
Quotes	122
Abbreviations	124
Summary	124
Chapter 5: Speeding Up Development Using jQuery Plugins	125
Browser detection	126
Enhanced pagination using bootpag	130
Displaying images using Bootstrap Lightbox	142
Improving our price list with DataTables	147
Summary	152

Chapter 6: Customizing Your Plugins	153
Anatomy of a plugin	154
JavaScript	154
Setup	154
Class definition	155
Data API implementation	158
jQuery	159
Sass	160
Customizing plugins	162
Customizing Bootstrap's jQuery alert plugin	162
The markup	163
Extending alert's style sheets	165
Extending alert's functionality with JavaScript	166
Customizing Bootstrap's jQuery carousel plugin	169
The markup	170
Extending carousel's functionality with JavaScript	171
Extending carousel's style sheets	173
Writing a custom Bootstrap jQuery plugin	175
The idea – the A11yHCM plugin	175
The a11yHCM.js file	178
The markup	181
Adding some style	183
Summary	187
Chapter 7: Integrating Bootstrap with Third-Party Plugins	189
Building a testimonial component with Salvattore	190
Introducing Salvattore	193
Integrating Salvattore with Bootstrap	199
Adding Animate.css to MyPhoto	202
Bouncing alerts	203
Animating a Salvattore grid	207
Hover	208
Adding Hover to MyPhoto	208
Making the navbar grow	209
Awesome Hover icons	209
Salvattore Hover	212
Summary	215
Chapter 8: Optimizing Your Website	216
CSS optimization	217
Inline styles	217
Long identifier and class names	219

Shorthand rules	221
Grouping selectors	222
Rendering times	222
Minifying CSS and JavaScript	223
Introducing Grunt	224
Minification and concatenation using Grunt	226
Running tasks automatically	228
Stripping our website of unused CSS	230
Processing HTML	233
Deploying assets	236
Stripping CSS comments	237
JavaScript file concatenation	238
Summary	240
Chapter 9: Integrating with AngularJS and React	241
Introducing AngularJS	242
Setting up AngularJS	242
Improving the testimonials component	243
Making testimonials dynamic	245
Making a Promise with \$q	247
Creating an AngularJS directive	248
Writing the testimonials template	250
Testing the testimonial directive	251
Importing the Salvatorre library	252
Introducing React	253
Setting up React	254
Making a Gallery component in React	256
Using carousel in React	259
Summary	264
Index	265

Module 3

Mastering Bootstrap 4

*Learn how to build beautiful and highly customizable web interfaces by leveraging
the power of Bootstrap 4*

1

Revving Up Bootstrap

Bootstrap is a web development framework that helps developers build web interfaces. Originally conceived at Twitter in 2011 by Mark Otto and Jacob Thornton, the framework is now open source and has grown to be one of the most popular web development frameworks to date. Being freely available for private, educational, and commercial use meant that Bootstrap quickly grew in popularity. Today, thousands of organizations rely on Bootstrap, including NASA, Walmart, and Bloomberg. According to BuiltWith.com, over 10% of the world's top 1 million websites are built using Bootstrap (<http://trends.builtwith.com/docinfo/Twitter-Bootstrap>). As such, knowing how to use Bootstrap will be an important skill and will serve as a powerful addition to any web developer's tool belt.

The framework itself consists of a mixture of JavaScript and CSS, and provides developers with all the essential components required to develop a fully functioning web user interface. Over the course of this book, we will be introducing you to all of the most essential features that Bootstrap has to offer by teaching you how to use the framework to build a complete website from scratch. As CSS and HTML alone are already the subject of entire books in themselves, we assume that you, the reader, has at least a basic knowledge of HTML, CSS, and JavaScript.

We begin this chapter by introducing you to our demo website, `MyPhoto`. This website will accompany us throughout this book, and serve as a practical point of reference. Therefore, all lessons learned will be taught within the context of `MyPhoto`.

We will then discuss the Bootstrap framework, listing its features and contrasting the current release to the last major release (Bootstrap 3).

Last but not least, this chapter will help you set up your development environment. To ensure equal footing, we will guide you towards installing the right build tools, and precisely detail the various ways in which you can integrate Bootstrap into a project. More advanced readers may safely skip this last part and continue to [Chapter 2, Making a Style Statement](#).

To summarize, this chapter will do the following:

- Introduce you to what exactly we will be doing
- Explain what is new in the latest version of Bootstrap, and how the latest version differs to the previous major release
- Show you how to include Bootstrap in our web project

Introducing our demo project

This book will teach you how to build a complete Bootstrap website from scratch. Starting with a simple layout in [Chapter 2, Making a Style Statement](#) and [Chapter 3, Building the Layout](#), we will build and improve the website's various sections as we progress through each chapter. The concept behind our website is simple: to develop a landing page for photographers. Using this landing page, (hypothetical) users will be able to exhibit their wares and services. While building our website, we will be making use of the same third-party tools and libraries that you would if you were working as a professional software developer. We chose these tools and plugins specifically because of their widespread use. Learning how to use and integrate them will save you a lot of work when developing websites in the future. Specifically, the tools that we will use to assist us throughout the development of MyPhoto are **Bower**, **node package manager (npm)**, and **Grunt**.

From a development perspective, the construction of MyPhoto will teach you how to use and apply all of the essential user interface concepts and components required to build a fully functioning website. Among other things, you will learn how to do the following:

- Use the Bootstrap grid system to structure the information presented on your website
- Create a fixed, branded, navigation bar with animated scroll effects

- Use an image carousel for displaying different photographs, implemented using Bootstrap's `carousel.js` and `jumbotron` (`jumbotron` is a design principle for displaying important content). It should be noted that carousels are becoming an increasingly unpopular design choice; however, they are still heavily used and are an important feature of Bootstrap. As such, we do not argue for or against the use of carousels as their effectiveness depends very much on how they are used, rather than on whether they are used.
- Build custom tabs that allow users to navigate across different contents
- Use and apply Bootstrap's modal dialogs
- Apply a fixed page footer
- Create forms for data entry using Bootstrap's input controls (text fields, text areas, and buttons) and apply Bootstrap's input validation styles
- Make best use of Bootstrap's context classes
- Create alert messages and learn how to customize them
- Rapidly develop interactive data tables for displaying product information
- Use drop-down menus, custom fonts, and icons

In addition to learning how to use Bootstrap 4, the development of `MyPhoto` will introduce you to a range of third-party libraries such as `Scrollspy` (for scroll animations), `SalvattoreJS` (a library for complementing our Bootstrap grid), `Animate.css` (for beautiful CSS animations, such as fade-in effects at <https://daneden.github.io/animate.css/>), and `Bootstrap DataTables` (for rapidly displaying data in tabular form).

The website itself will consist of different sections:

- A **Welcome** section
- An **About** section
- A **Services** section
- A **Gallery** section
- A **Contact Us** section

The development of each section is intended to teach you how to use a distinct set of features found in third-party libraries. For example, by developing the **Welcome** section, you will learn how to use Bootstrap's `jumbotron` and `alert` dialogs along with different font and text styles, while the **About** section will show you how to use cards. The **Services** section of our project introduces you to Bootstrap's custom tabs. That is, you will learn how to use Bootstrap's tabs to display a range of different services offered by our website.

Following on from the **Services** section, you will need to use rich imagery to really show off the website's sample services. You will achieve this by really mastering Bootstrap's responsive core along with Bootstrap's carousel and third-party jQuery plugins. Last but not least, the **Contact Us** section will demonstrate how to use Bootstrap's form elements and helper functions. That is, you will learn how to use Bootstrap to create stylish HTML forms, how to use form fields and input groups, and how to perform data validation.

Finally, toward the end of the book, you will learn how to optimize your website, and integrate it with the popular JavaScript frameworks AngularJS (<https://angularjs.org/>) and React (<http://facebook.github.io/react/>). As entire books have been written on AngularJS alone, we will only cover the essentials required for the integration itself.

Now that you have glimpsed a brief overview of MyPhoto, let's examine Bootstrap 4 in more detail, and discuss what makes it so different to its predecessor. Take a look at the following screenshot:



Figure 1.1: A taste of what is to come: the MyPhoto landing page

What Bootstrap 4 Alpha 4 has to offer

Much has changed since Twitter's Bootstrap was first released on August 19th, 2011. In essence, Bootstrap 1 was a collection of CSS rules offering developers the ability to lay out their website, create forms, buttons, and help with general appearance and site navigation. With respect to these core features, Bootstrap 4 Alpha 4 is still much the same as its predecessors. In other words, the framework's focus is still on allowing developers to create layouts, and helping to develop a consistent appearance by providing stylings for buttons, forms, and other user interface elements. How it helps developers achieve and use these features, however, has changed entirely. Bootstrap 4 is a complete rewrite of the entire project, and, as such, ships with many fundamental differences to its predecessors. Along with Bootstrap's major features, we will be discussing the most striking differences between Bootstrap 3 and Bootstrap 4 in the sub-sections that follow.

Layout

Possibly the most important and widely used feature is Bootstrap's ability to lay out and organize your page. Specifically, Bootstrap offers the following:

- Responsive containers
- Responsive breakpoints for adjusting page layout in response to differing screen sizes
- A 12 column grid layout for flexibly arranging various elements on your page
- Media objects that act as building blocks and allow you to build your own structural components
- Utility classes that allow you to manipulate elements in a responsive manner. For example, you can use the layout utility classes to hide elements, depending on screen size

We will be discussing each of these features in detail in [Chapter 2, Making a Style Statement](#) and [Chapter 3, Building the Layout](#).

Content styling

Just like its predecessor, Bootstrap 4 overrides the default browser styles. This means that many elements, such as lists or headings, are padded and spaced differently. The majority of overridden styles only affect spacing and positioning; however, some elements may also have their border removed. The reason behind this is simple: to provide users with a clean slate upon which they can build their site.

Building on this clean slate, Bootstrap 4 provides styles for almost every aspect of your web page such as buttons (*Figure 1.2*), input fields, headings, paragraphs, special inline texts, such as keyboard input (*Figure 1.3*), figures, tables, and navigation controls. Aside from this, Bootstrap offers state styles for all input controls, for example, styles for disabled buttons or toggled buttons. Take a look at the following screenshot:



Figure 1.2: The six button styles that come with Bootstrap 4 are btn-primary, btn-secondary, btn-success, btn-danger, btn-link, btn-info, and btn-warning

Take a look at the following screenshot:



Figure 1.3: Bootstrap's content styles. In the preceding example, we see inline styling for denoting keyboard input

Components

Aside from layout and content styling, Bootstrap offers a large variety of reusable components that allow you to quickly construct your website's most fundamental features. Bootstrap's UI components encompass all of the fundamental building blocks that you would expect a web development toolkit to offer: modal dialogs, progress bars, navigation bars, tooltips, popovers, a carousel, alerts, drop-down menus, input groups, tabs, pagination, and components for emphasizing certain contents.

Let's have a look at the following modal dialog screenshot:

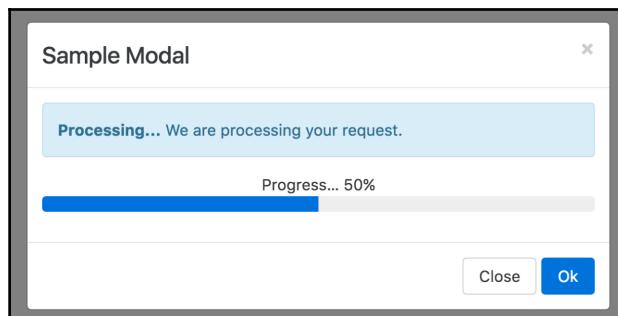


Figure 1.4: Various Bootstrap 4 components in action. In the preceding screenshot we see a sample modal dialog, containing an info alert, some sample text, and an animated progress bar.

Mobile support

Similar to its predecessor, Bootstrap 4 allows you to create mobile-friendly websites without too much additional development work. By default, Bootstrap is designed to work across all resolutions and screen sizes, from mobile, to tablet, to desktop. In fact, Bootstrap's *mobile-first* design philosophy implies that its components must display and function correctly at the smallest screen size possible. The reasoning behind this is simple. Think about developing a website without consideration for small mobile screens. In this case, you are likely to pack your website full of buttons, labels, and tables. You will probably only discover any usability issues when a user attempts to visit your website using a mobile device only to find a small web page that is crowded with buttons and forms. At this stage, you will be required to rework the entire user interface to allow it to render on smaller screens. For precisely this reason, Bootstrap promotes a bottom-up approach, forcing developers to get the user interface to render correctly on the smallest possible screen size, before expanding upwards.

Utility classes

Aside from ready-to-go components, Bootstrap offers a large selection of utility classes that encapsulate the most commonly needed style rules. For example, rules for aligning text, hiding an element, or providing contextual colors for warning text.

Cross-browser compatibility

Bootstrap 4 supports the vast majority of modern browsers, including Chrome, Firefox, Opera, Safari, Internet Explorer (version 9 and onwards; Internet Explorer 8 and below are not supported), and Microsoft Edge.

Sass instead of Less

Both Less and Sass (**Syntactically Awesome Stylesheets**) are CSS extension languages. That is, they are languages that extend the CSS vocabulary with the objective of making the development of many, large, and complex style sheets easier. Although Less and Sass are fundamentally different languages, the general manner in which they extend CSS is the same, both rely on a preprocessor. As you produce your build, the preprocessor is run, parsing the Less/Sass script and turning your Less or Sass instructions into plain CSS.

Less is the official Bootstrap 3 build, while Bootstrap 4 has been developed from scratch, and is written entirely in Sass. Both Less and Sass are compiled into CSS to produce a single file, `bootstrap.css`. It is this CSS file that we will be primarily referencing throughout this book (with the exception of [Chapter 3, Building the Layout](#)). Consequently, you will not be required to know Sass in order to follow this book. However, we do recommend that you take a 20 minute introductory course on Sass if you are completely new to the language. Rest assured, if you already know CSS, you will not need more time than this. The language's syntax is very close to normal CSS, and its elementary concepts are similar to those contained within any other programming language.

From pixel to root em

Unlike its predecessor, Bootstrap 4 no longer uses **pixel (px)** as its unit of typographic measurement. Instead, it primarily uses **root em (rem)**. The reasoning behind choosing rem is based on a well known problem with px; websites using px may render incorrectly, or not as originally intended, as users change the size of the browser's base font. Using a unit of measurement that is relative to the page's root element helps address this problem, as the root element will be scaled relative to the browser's base font. In turn, a page will be scaled relative to this root element.

Typographic units of measurement

Simply put, typographic units of measurement determine the size of your font and elements. The most commonly used units of measurement are px and em. The former is an abbreviation for pixel, and uses a reference pixel to determine a font's exact size. This means that, for displays of **96 dots per inch (dpi)**, 1 px will equal an actual pixel on the screen. For higher resolution displays, the reference pixel will result in the px being scaled to match the display's resolution. For example, specifying a font size of 100 px will mean that the font is exactly 100 pixels in size (on a display with 96 dpi), irrespective of any other element on the page.



Em is a unit of measurement that is relative to the parent of the element to which it is applied. So, for example, if we were to have two nested div elements, the outer element with a font size of 100 px and the inner element with a font size of 2 em, then the inner element's font size would translate to 200 px (as in this case $1 \text{ em} = 100 \text{ px}$). The problem with using a unit of measurement that is relative to parent elements is that it increases your code's complexity, as the nesting of elements makes size calculations more difficult.



The recently introduced rem measurement aims to address both em's and px's shortcomings by combining their two strengths; instead of being relative to a parent element, rem is relative to the page's root element.

No more support for Internet Explorer 8

As was already implicit in the preceding feature summary, the latest version of Bootstrap no longer supports Internet Explorer 8. As such, the decision to only support newer versions of Internet Explorer was a reasonable one, as not even Microsoft itself provides technical support and updates for Internet Explorer 8 anymore (as of January 2016). Furthermore, Internet Explorer 8 does not support rem, meaning that Bootstrap 4 would have been required to provide a workaround. This in turn would most likely have implied a large amount of additional development work, with the potential for inconsistencies. Lastly, responsive website development for Internet Explorer 8 is difficult, as the browser does not support CSS media queries. Given these three factors, dropping support for this version of Internet Explorer was the most sensible path of action.

A new grid tier

Bootstrap's grid system consists of a series of CSS classes and media queries that help you lay out your page. Specifically, the grid system helps alleviate the pain points associated with horizontal and vertical positioning of a page's contents and the structure of the page across multiple displays. With Bootstrap 4, the grid system has been completely overhauled, and a new grid tier has been added with a breakpoint of 480 px and below. We will be talking about tiers, breakpoints, and Bootstrap's grid system extensively in Chapter 2, *Making a Style Statement*.

Bye-bye GLYPHICONS

Bootstrap 3 shipped with a nice collection of over 250 font icons, free of use. In an effort to make the framework more lightweight (and because font icons are considered bad practice), the GLYPHICON set is no longer available in Bootstrap 4.

Bigger text: no more panels, wells, and thumbnails

The default font size in Bootstrap 4 is 2 px bigger than in its predecessor, increasing from 14 px to 16 px. Furthermore, Bootstrap 4 replaced panels, wells, and thumbnails with a new concept: **cards**. To readers unfamiliar with the concept of wells, a well is a UI component that allows developers to highlight text content by applying an inset shadow effect to the element to which it is applied. A panel also serves to highlight information, but by applying padding and rounded borders. Cards serve the same purpose as their predecessors, but are less restrictive as they are flexible enough to support different types of content, such as images, lists, or text. They can also be customized to use footers and headers. Take a look at the following screenshot:

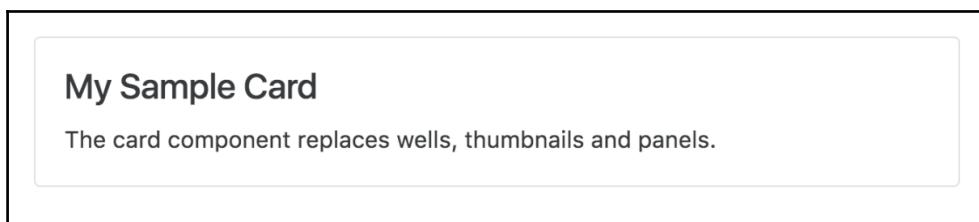


Figure 1.5: The Bootstrap 4 card component replaces existing wells, thumbnails, and panels

New and improved form input controls

Bootstrap 4 now allows for input control sizing, as well as classes for denoting block and inline level input controls. However, one of the most anticipated new additions is Bootstrap's input validation styles, which used to require third-party libraries or a manual implementation, but are now shipped with Bootstrap 4 (see *Figure 1.6*). Take a look at the following screenshot:

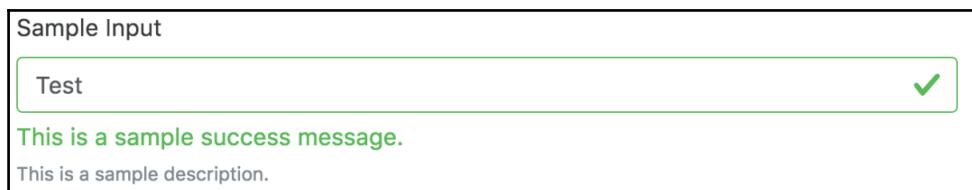


Figure 1.6: The new Bootstrap 4 input validation styles, indicating the successful processing of input

Last but not least, Bootstrap 4 also offers custom forms in order to provide even more cross-

browser UI consistency across input elements (*Figure 1.7*). As noted in the Bootstrap 4 Alpha 4 documentation, the input controls are “*built on top of semantic and accessible markup, so they’re solid replacements for any default form control*” (source: <http://v4-alpha.getbootstrap.com/components/forms/>). Take a look at the following screenshot:



Figure 1.7: Custom Bootstrap input controls that replace the browser defaults in order to ensure cross-browser UI consistency

Customization

The developers behind Bootstrap 4 have put specific emphasis on customization throughout the development of Bootstrap 4. As such, many new variables have been introduced that allow for the easy customization of Bootstrap. Using the `$enabled--` Sass variables, one can now enable or disable specific global CSS preferences.

Setting up our project

Now that we know what Bootstrap has to offer, let us set up our project:

1. Create a new project directory named `MyPhoto`. This will become our project root directory.
2. Create a blank `index.html` file and insert the following HTML code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
    initial-scale=1, shrink-to-fit=no">
    <meta http-equiv="x-ua-compatible" content="ie=edge">
    <title>MyPhoto</title>
</head>
<body>
    <div class="alert alert-success">
        Hello World!
    </div>
</body>
</html>
```

Note the three `meta` tags. The first tag tells the browser that the document in question is `utf-8` encoded. Since Bootstrap optimizes its content for mobile devices, the subsequent `meta` tag is required to help with viewport scaling. The last `meta` tag forces the document to be rendered using the latest document rendering mode available if viewed in Internet Explorer.

3. Open the `index.html` in your browser. You should see just a blank page with the words **Hello World**.

Now it is time to include Bootstrap. At its core, Bootstrap is a glorified CSS style sheet. Within that style sheet, Bootstrap exposes very powerful features of CSS with an easy-to-use syntax. It being a style sheet, you include it in your project as you would with any other style sheet that you might develop yourself. That is, open the `index.html` and directly link it to the style sheet.

Viewport scaling

The term “viewport” refers to the available display size to render the contents of a page. The `viewport` `meta` tag allows you to define this available size. Viewport scaling using `meta` tags was first introduced by Apple and, at the time of writing, is supported by all major browsers.

Using the `width` parameter, we can define the exact width of the user's viewport. For example, `<meta name="viewport" content="width=320px">` will instruct the browser to set the viewport's width to `320px`. The ability to control the viewport's width is useful when developing mobile-friendly websites; by default, mobile browsers will attempt to fit the entire page onto their viewports by zooming out as far as possible. This allows users to view and interact with websites that have not been designed to be viewed on mobile devices. However, as Bootstrap embraces a mobile-first design philosophy, a zoom out will, in fact, result in undesired side-effects. For example, breakpoints (which we will discuss in Chapter 2, *Making a Style Statement*) will no longer work as intended, as they now deal with the zoomed-out equivalent of the page in question. This is why explicitly setting the viewport width is so important. By writing `content="width=device-width, initial-scale=1, shrink-to-fit=no"`, we are telling the browser the following:

- We want to set the viewport's width equal to the actual device's screen width.
- We do not want any zoom, initially.



- We do not wish to shrink the content to fit the viewport.

For now, we will use the Bootstrap builds hosted on Bootstrap's official **Content Delivery Network (CDN)**. This is done by including the following HTML tag into the head of your HTML document (the head of your HTML document refers to the contents between the <head> opening tag and the </head> closing tag):

```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-alpha.4/css/bootstrap.min.css">
```

Bootstrap relies on jQuery, a JavaScript framework that provides a layer of abstraction in an effort to simplify the most common JavaScript operations (such as element selection and event handling). Therefore, before we include the Bootstrap JavaScript file, we must first include jQuery. Both inclusions should occur just before the </body> closing tag:

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.4
/jquery.min.js">
</script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-alpha.4
/js/bootstrap.min.js"></script>
```

Note that, while these scripts could, of course, be loaded at the top of the page, loading scripts at the end of the document is considered best practice to speed up page loading times and to avoid JavaScript issues preventing the page from being rendered. The reason behind this is that browsers do not download all dependencies in parallel (although a certain number of requests are made asynchronously, depending on the browser and the domain). Consequently, forcing the browser to download dependencies early on will block page rendering until these assets have been downloaded. Furthermore, ensuring that your scripts are loaded last will ensure that once you invoke **Document Object Model (DOM)** operations in your scripts, you can be sure that your page's elements have already been rendered. As a result, you can avoid checks that ensure the existence of given elements.

What is a Content Delivery Network?



The objective behind any Content Delivery Network (CDN) is to provide users with content that is highly available. This means that a CDN aims to provide you with content, without this content ever (or rarely) becoming unavailable. To this end, the content is often hosted using a large, distributed set of servers. The BootstrapCDN basically allows you to link to the Bootstrap style sheet so that you do not have to host it yourself.

Save your changes and reload the `index.html` in your browser. The **Hello World** string should now contain a green background:

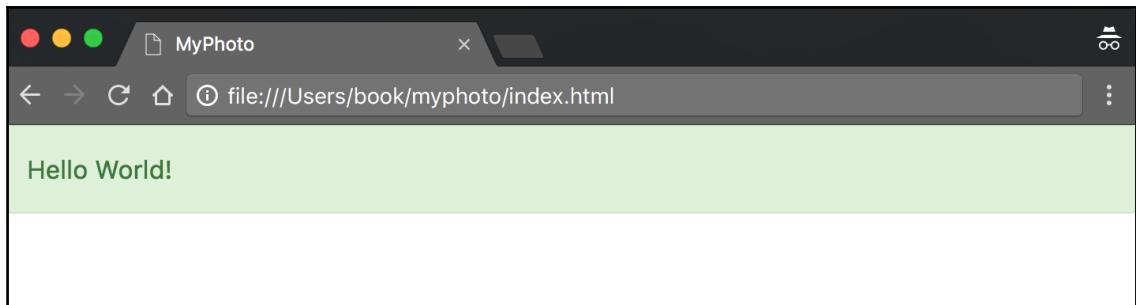


Figure 1.8: Our “Hello World” styled using Bootstrap 4

Now that the Bootstrap framework has been included in our project, open your browser's developer console (if using Chrome on Microsoft Windows, press *Ctrl + Shift + I*; on Mac OS X you can press *cmd + alt + I*). As Bootstrap requires another third-party library Tether, for displaying popovers and tooltips, the developer console will display an error (*Figure 1.6*). Take a look at the following screenshot:

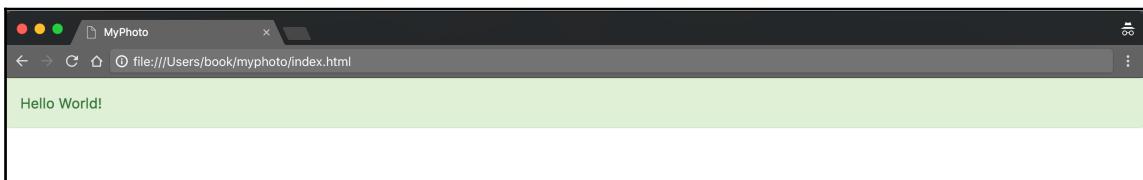


Figure 1.9: Chrome's Developer Tools can be opened by going to View, selecting Developer, and then clicking on Developer Tools. At the bottom of the page, a new view will appear. Under the Console tab, an error will indicate an unmet dependency.

Tether is available via the CloudFare CDN, and consists of both a CSS file and a JavaScript file. As before, we should include the JavaScript file at the bottom of our document while we reference Tether's style sheet from inside our document head:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-
        scale=1, shrink-to-fit=no">
    <meta http-equiv="x-ua-compatible" content="ie=edge">
    <title>MyPhoto</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-alpha.4/css/bootstrap.min.css">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/
        libs/tether/1.3.1/css/tether.min.css">
</head>
<body>
```

```
<div class="alert alert-success">
    Hello World!
</div>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/
2.1.4/jquery.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/tether/
1.3.1/js/tether.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-
alpha.4/js/bootstrap.min.js"></script>
</body>
</html>
```

While CDNs are an important resource, there are several reasons why, at times, using a third-party CDN may not be desirable:

- CDNs introduce an additional point of failure, as you now rely on third-party servers.
- The privacy and security of users may be compromised, as there is no guarantee that the CDN provider does not inject malicious code into the libraries that are being hosted. Nor can one be certain that the CDN does not attempt to track its users.
- Certain CDNs may be blocked by the Internet Service Providers of users in different geographical locations.
- Offline development will not be possible when relying on a remote CDN.
- You will not be able to optimize the files hosted by your CDN. This loss of control may affect your website's performance (although typically you are more often than not offered an optimized version of the library through the CDN).

Instead of relying on a CDN, we could manually download the jQuery, Tether, and Bootstrap project files. We could then copy these builds into our project root and link them to the distribution files. The disadvantage of this approach is the fact that maintaining a manual collection of dependencies can quickly become very cumbersome, and next to impossible as your website grows in size and complexity. As such, we will not manually download the Bootstrap build. Instead, we will let Bower do it for us. Bower is a package management system, that is, a tool that you can use to manage your website's dependencies. It automatically downloads, organizes, and (upon command) updates your website's dependencies. To install Bower, head over to <http://bower.io/>.

How do I install Bower?



Before you can install Bower, you will need two other tools: Node.js and Git. The latter is a version control tool, in essence; it allows you to manage different versions of your software. To install Git, head over to <http://gi>



[t-scm.com/](https://nodejs.org/) and select the installer appropriate for your operating system. NodeJS is a JavaScript runtime environment needed for Bower to run. To install it, simply download the installer from the official NodeJS website: <https://nodejs.org/> Once you have successfully installed Git and NodeJS, you are ready to install Bower. Simply type the following command into your terminal:

```
npm install -g bower
```

This will install Bower for you, using the JavaScript package manager npm, which happens to be used by, and is installed with, NodeJS.

Once Bower has been installed, open up your terminal, navigate to the project root folder you created earlier, and fetch the bootstrap build:

```
bower install bootstrap#v4.0.0-alpha.4
```

This will create a new folder structure in our project root:

```
|__bower_components
|__bootstrap
|__Gruntfile.js
|__LICENSE
|__README.md
|__bower.json
|__dist
|__fonts
|__grunt
|__js
|__less
|__package.js
|__package.json
```

We will explain all of these various files and directories later on in this book. For now, you can safely ignore everything except for the `dist` directory inside `bower_components/bootstrap/`. Go ahead and open the `dist` directory. You should see three sub directories:

- `css`
- `fonts`
- `js`

The name `dist` stands for distribution. Typically, the distribution directory contains the production-ready code that users can deploy. As its name implies, the `css` directory inside `dist` includes the ready-for-use style sheets. Likewise, the `js` directory contains the JavaScript files that compose Bootstrap. Lastly, the `fonts` directory holds the font assets that come with Bootstrap.

To reference the local Bootstrap CSS file in our `index.html`, modify the `href` attribute of the link tag that points to the `bootstrap.min.css`:

```
<link rel="stylesheet" href="bower_components/bootstrap/dist/css  
/bootstrap.min.css">
```

Let's do the same for the Bootstrap JavaScript file:

```
<script src="bower_components/bootstrap/dist/js/bootstrap.min.js"></script>
```

Repeat this process for both jQuery and Tether. To install jQuery using Bower, use the following command:

```
bower install jquery
```

Just as before, a new directory will be created inside the `bower_components` directory:

```
|__bower_components  
|__jquery  
|__AUTHORS.txt  
|__LICENSE.txt  
|__bower.json  
|__dist  
|__sizzle  
|__src
```

Again, we are only interested in the contents of the `dist` directory, which, among other files, will contain the compressed jQuery build `jquery.min.js`.

Reference this file by modifying the `src` attribute of the script tag that currently points to Google's `jquery.min.js` by replacing the URL with the path to our local copy of jQuery:

```
<script src="bower_components/jquery/dist/jquery.min.js"></script>
```

Last but not least, repeat the steps already outlined for Tether:

```
bower install tether
```

Once the installation completes, a similar folder structure than the ones for Bootstrap and jQuery will have been created. Verify the contents of `bower_components/tether/dist` and replace the CDN Tether references in our document with their local equivalent.

The final `index.html` should now look as follows:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,
    shrink-to-fit=no">
    <meta http-equiv="x-ua-compatible" content="ie=edge">
    <title>MyPhoto</title>
    <link rel="stylesheet" href="bower_components/bootstrap/dist/css
    /bootstrap.min.css">
    <link rel="stylesheet" href="bower_components/tether/dist/css
    /tether.min.css">
</head>
<body>
    <div class="alert alert-success">
        Hello World!
    </div>
    <script src="bower_components/jquery/dist/jquery.min.js"></script>
    <script src="bower_components/tether/dist/js/tether.min.js">
    </script>
    <script src="bower_components/bootstrap/dist/js/bootstrap.min.js">
    </script>
</body>
</html>
```

Refresh the `index.html` in your browser to make sure that everything works.

What IDE and browser should I be using when following the examples in this book?

While we recommend a JetBrains IDE or Sublime Text along with Google Chrome, you are free to use whatever tools and browser you like. Our taste in IDE and browser is subjective on this matter. However, keep in mind that Bootstrap 4 does not support Internet Explorer 8 or below. As such, if you do happen to use Internet Explorer 8, you should upgrade it to the latest version.

Summary

Aside from introducing you to our sample project *MyPhoto*, this chapter was concerned with outlining Bootstrap 4, highlighting its features, and discussing how this new version of Bootstrap differs to the last major release (Bootstrap 3). The chapter provided an overview of how Bootstrap can assist developers in the layout, structuring, and styling of pages. Furthermore, we noted how Bootstrap provides access to the most important and widely used user interface controls through the form of components that can be integrated into a page with minimal effort. By providing an outline of Bootstrap, we hope that the framework's intrinsic value in assisting in the development of modern websites has become apparent to the reader. Furthermore, during the course of the wider discussion, we highlighted and explained some important concepts in web development, such as typographic units of measurement or the definition, purpose, and justification of the use of Content Delivery Networks. Last but not least, we detailed how to include Bootstrap and its dependencies inside an HTML document. This sets the scene for *Chapter 2, Making a Style Statement* in which we will introduce you to the Bootstrap grid system by building a general layout for the first page of our sample project.

2

Making a Style Statement

In the previous chapter, we showed you a first glimpse of `MyPhoto`, the Bootstrap website that we are going to build up throughout the following chapters. Now it is time to get our hands dirty, and actually start building the first section of this website. A first pass at an element of the **Services** section that presents the list of print sizes available to order. We will achieve this by building a grid using Bootstrap's grid system, creating image elements within the grid system, applying image modifiers, and leveraging Bootstrap's utility classes to create visual indicators and optimized layouts specific to different display resolutions.

By the end of this chapter, through code examples and studying the Bootstrap source code, you will have gained a deep understanding of the following:

- Bootstrap's grid system
- Responsive images within Bootstrap
- Bootstrap's helper classes
- Bootstrap's responsive utilities

The grid system

Bootstrap's grid system is arguably its most impressive and most commonly used feature. Therefore, mastering it is essential for any Bootstrap developer as the grid system removes many of the pain-points associated with page layouts, especially responsive page layouts. The grid system solves issues such as the horizontal and vertical positioning of a page's contents and the structure of the page across multiple display widths.

As already noted in Chapter 1, *Revving up Bootstrap*, Bootstrap 4 is mobile-first. As such, it should come as no surprise that the grid system is optimized for smaller viewports and scales up to suit larger viewports, as opposed to scaling down to smaller viewports.



What is a viewport?

A viewport is the available display size to render the contents of a page. For example, the size of your browser window, minus the toolbars, scrollbars, and so on, on your display is your viewport. As already noted in Chapter 1, *Revving Up Bootstrap*, mobile devices may indicate their viewport to be larger than it actually is, in order to allow for the display of websites that have not been optimized for display on mobile devices. As a consequence, websites that take mobile viewports into consideration, may often not render as intended. As a remedy, the `viewport` meta tag was introduced by Apple on iOS, and has since been uniformly adopted by all other major browsers. The `viewport` meta tag allows you to define the viewport's display size.

The grid is a structure that consists of three distinct, but fundamentally linked, parts: an all encapsulating **container**, split into horizontal **rows** which are themselves split into 12 equal **columns**. We will take an in depth look into the three building blocks of Bootstrap's grid system:

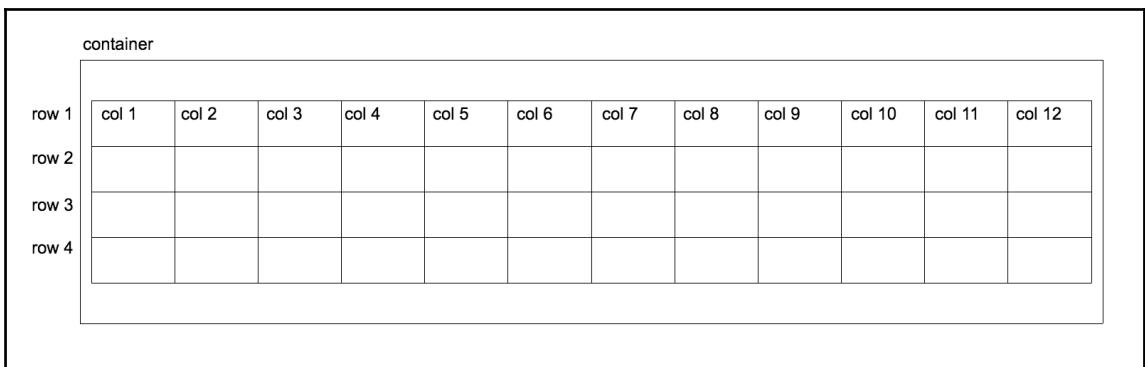


Figure 2.1: The Bootstrap grid structure: a container (outermost box) containing a table-like structure consisting of rows and 12 columns. It is important to note that rows must be contained inside the container. Likewise, columns can only exist within the context of rows. While grids can be used to construct tables, they are not tables in themselves. Unlike tables, independent rows may consist of a different number of columns. So, for example, row 1 may consist of 12 columns, while row 2 may contain only three columns.

Flexbox support



Flexbox is a CSS box model which allows for simple implementation of complex layouts, as opposed to the CSS2 layout modules such as block, inline, table, and positioned. Flexbox is designed to allow a layout to make the most use out of the available space, through a set of simple rules. Bootstrap 4 allows the developer to configure the framework to use flexbox for certain components, by changing one variable in `_variables.scss`—`$enable-flex`. Set `$enable-flex` to `true`, recompile Bootstrap and a number of Bootstrap components will have their `display` property set to `flex`. This includes the grid system itself, input groups, and the media component. You can find out more about flexbox at <https://www.w3.org/TR/css-flexbox-1/>.

Containers

Containers are at the core of Bootstrap's grid system, and practically the parent of all Bootstrap pages. A container is exactly what it sounds like. It encapsulates all other content within a section of a page, providing the base for how the section is rendered. You can think of a container as representing a canvas in a browser window for your content to be displayed on a canvas that can transform based on its environment. Unless explicitly specified, your content will never creep outside of this canvas, regardless of the viewport. A container can apply to the entire contents of a page, where you would have one root container element, or to different sections of a page, where you would have numerous container elements on the page.

There are two types of container classes provided by Bootstrap: `container` and `container-fluid`.

container

The `container` class renders the contents of the page to a fixed width. This width is typically based upon the width of the viewport, leveraging CSS media queries to determine which width is most suitable.

What are media queries?



Media queries are expressions, which resolve to a Boolean value. They are used to trigger @media rules that define styles for different media types.

See https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Media_queries for further information.

The grid system has five core **breakpoints** it references, which are defined in `_variables.scss`. These are, extra-small (`xs`), small (`sm`), medium (`md`), large (`lg`) and extra-large (`xl`).

What are Breakpoints?



Breakpoints in relation to web development layouts are predefined vertical and horizontal dimensions at which the style rules change. As these rules break, they trigger another set of rules optimized for those dimensions. These rules are triggered by media queries, querying the dimensions of the viewport. For example, `@media (min-width: 768px)` will trigger a set of rules when the viewport is more than 768px wide.

Let's take a look at `_variables.scss`:

```
$grid-breakpoints: (
  // Extra small screen / phone
  xs: 0,
  // Small screen / phone
  sm: 544px,
  // Medium screen / tablet
  md: 768px,
  // Large screen / desktop
  lg: 992px,
  // Extra large screen / wide desktop
  xl: 1200px
) !default;
```

Here, Bootstrap is defining the five breakpoints' minimum and maximum width variables, and the associated display types. Bootstrap will reference these variables throughout all its Sass code as the breakpoints can now be accessed as properties of `$grid-breakpoints`. We can also see the variables for the various container sizes, associated with the appropriate breakpoints. Look at the following code:

```
// Grid containers
//
// Define the maximum width of `container` for different screen sizes.
```

```
$container-max-widths: (
    sm: 576px,
    md: 720px,
    lg: 940px,
    xl: 1140px
) !default;
// Grid columns
//
// Set the number of columns and specify the width of the gutters.
$grid-columns: 12 !default;
$grid-gutter-width: 1.875rem !default; // 30px
```

For example, `container-tablet` is set to `750px: 720px` plus the value of `grid-gutter-width`, which is `30px`. As you can see from the comments in the code, `container-**` is associated directly with `screen-**`. Then, these sizes are leveraged via media queries in `_grid.scss` to set the desired width of the container. Let's take a look inside `_grid.scss` at the `.container` class:

```
.container {
    @include make-container();
    @include make-container-max-widths();
}
```

Let's break this down.

The `make-container()` and `make-container-max-widths()` are mixins with rules to center the container within the viewport and set `max-width` rules, respectively.

What is a mixin?



A mixin in this context is a set of predefined style rules encapsulated in a variable, which can be used within another rules definition. This is great for code maintenance and **don't repeat yourself (DRY)** principles.

You will also find `make-container` and `make-container-max-widths` within `_grid.scss`. The `make-container` mixin centralizes the alignment of the container using margin and padding rules. Have a look at the following code:

```
@mixin make-container($gutter: $grid-gutter-width) {
    margin-left: auto;
    margin-right: auto;
    padding-left: ($gutter / 2);
    padding-right: ($gutter / 2);
    @if not $enable-flex {
        @include clearfix();
```

```
    }
}
```

The `make-container-max-widths` mixin is more complex. The mixin loops through the global `$breakpoint` variable, synonymous with `$grid-breakpoints`, and sets a `max-width` rule for each breakpoint, using media queries. Take a look at the following code:

```
// For each breakpoint, define the maximum width of the
// container in a media query
@mixin make-container-max-widths($max-widths: $container-max-widths) {
  @each $breakpoint, $container-max-width in $max-widths {
    @include media-breakpoint-up($breakpoint) {
      max-width: $container-max-width;
    }
  }
}
```

The completed code then looks like the following:

```
@media (min-width: 544px) {
  .container {
    max-width: 576px;
  }
}
@media (min-width: 768px) {
  .container {
    max-width: 720px;
  }
}
@media (min-width: 992px) {
  .container {
    max-width: 940px;
  }
}
@media (min-width: 1200px) {
  .container {
    max-width: 1140px;
  }
}
```

There are four media queries, defining the horizontal breakpoint to trigger a width style rule. For example, `@media (min-width: 768px)` instructs the browser to only set the `width` property to the `max-width` of the container to `720px` for viewports wider than or equal to `768px`. This property is then superseded by the `@media (min-width: 992px)` rule when the viewport is wider than or equal to `992px`.

In the vast majority of cases, the width of the contents of the page is fixed to the width of the container. There are cases where the width of the container is ignored. One such case is Bootstrap's `navbar` class, in which the `navbar` element is allowed to fill the entire horizontal width of the viewport. We will come across this scenario in a later chapter.

Now that we have seen how the container is constructed and the theory behind the container, let us see it in practice. A container is generally a `div` with a `container` class in the body of the markup, wrapping around the page's main content. For example:

```
<body>
    <div class="container">
        <h1>Help, I'm trapped in a container!</h1>
    </div>
    <div>
        <h1>I'm free!</h1>
    </div>
</body>
```

Take a look at the following screenshot:



Figure 2.2: Using the `container` class

container-fluid

The other type of container, `container-fluid`, differs from `container` in two distinct ways:

- It takes up the full-width of the viewport, except for 15 pixels padding left and right
- It doesn't concern itself with breakpoints

The `container-fluid` allows the page to be fully responsive to all widths, providing smoother transitions. When responding to breakpoints, `container` snaps the layout to the appropriate width, while `container-fluid` progressively alters the layout.

The only difference in the markup is that instead of the `container` class being applied to the container `div`, the `container-fluid` class is applied. Look at the following code snippet:

```
<body>
  <div class="container-fluid">
    <h1>Help, I'm trapped in a container!</h1>
  </div>
  <div>
    <h1>I'm free!</h1>
  </div>
</body>
```

Take a look at the following screenshot:



Help, I'm trapped in a container!
I'm free!

Figure 2.3: Using the `container-fluid` class

Note that the container element now sits 15 pixels from the edge of the browser. When we use `container`, the container already has a hard-coded width defined. This width is based on the viewport. For example, at a resolution of 1200 px wide, the container would be 1140 px wide. At a resolution of 1280 pixels, the container would remain at 1170 px wide, because the container only responds to certain breakpoints. When we use `container-fluid`, the container width is dynamic, because `container-fluid` responds to every horizontal change and bases the width solely on the padding values from the `make-container` mixin. `container`, on the other hand, responds only at specific widths. `container-fluid` is the approach to take when building a page which needs to work across all display sizes and forms, especially when building mobile-first applications.

The `container` ensures that our contents will always display within a defined area on the page. But what about positioning content within the container? This is where rows come into play.

Box sizing

In CSS, every element is represented as a rectangle, or box. Each box has a number of properties associated with it to define how the element is rendered. This is the CSS Box Model. The `box-sizing` property of an element defines how the Box Model should calculate the width and height of elements.



The default value for `box-sizing` in the CSS box model is `content-box`. The `content-box` property only includes the content of an element when calculating the size of that element.

Bootstrap 4 defaults the value of `box-sizing` to `border-box`. The `border-box` property includes the padding and border, as well as the content of the element in the calculation of the height and width of the element. Note that the margin is not included in the calculation. The third possible value for `box-sizing` is `padding-box`. The `padding-box` property, as the name suggests, only uses the content and the padding in calculating the size of an element.

Rows

A `row` is used to define a selection of elements that should be dealt with as a horizontal group. As such, rows reside within a container element. The power of the row lies in being able to stack content vertically. Almost like containers within a container, or defining a section of the page. Creating a row is as simple as applying the `row` class to the desired element:

```
<body>
  <div class="container">
    <h1>Help, I'm trapped in a container!</h1>
    <div class="row">
      <div>Section 1</div>
    </div>
    <div class="row">
      <div>Section 2</div>
    </div>
    <div class="row">
      <div>Section 3</div>
    </div>
  </div>
<div>
```

```
<h1>I'm free!</h1>
</div>
</body>
```

Take a look at the following screenshot:



Figure 2.4: Using rows

The true power of rows only becomes apparent when they are used with columns.

Columns

Arguably, columns are the most important piece of the grid system. Rows exist within containers, and those rows are split up into 12 equal columns. Before we get into the nitty-gritty details, let's take a look at an example, by taking the first step into creating the print sizes section of MyPhoto. There will be 12 print sizes offered. Let's list those sizes horizontally:

```
<div class="container">
  <h1>Our Print Sizes</h1>
  <div class="row">
    <div class="col-sm-1">6x5</div>
    <div class="col-sm-1">8x10</div>
    <div class="col-sm-1">11x17</div>
    <div class="col-sm-1">12x18</div>
    <div class="col-sm-1">16x20</div>
    <div class="col-sm-1">18x24</div>
    <div class="col-sm-1">19x27</div>
    <div class="col-sm-1">20x30</div>
    <div class="col-sm-1">22x28</div>
    <div class="col-sm-1">24x36</div>
    <div class="col-sm-1">27x39</div>
    <div class="col-sm-1">27x40</div>
  </div>
</div>
```

As usual, we have our container. Within that container, we have a row, and within that row we have twelve individual elements with the col-sm-1. This produces a very neat list of evenly spaced elements in a single row on the page. Observe the following screenshot:

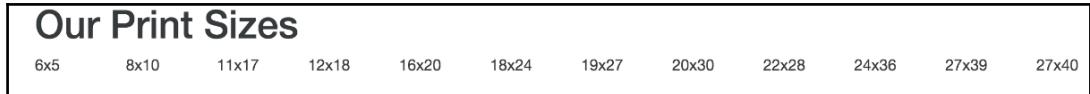


Figure 2.5: Using columns to display print sizes

Let's break down col-xs-1 and explain each part individually:

- col: This means that we want this element to act as a column.
- sm: This is a reference to all viewports above or equal to 544px. This class means we apply this rule for all viewports equal to or larger than 544px.
- 1: This means that the element takes up one column width of the row (1/12 of the row width).

Because col-sm-1 references viewports larger than 544px, smaller viewports (such as phones) revert to a stacked view. Take a look at the following screenshot:



Figure 2.6: Viewports smaller than 544px revert to a stacked view when using col-sm-1

Columns are split up into five distinct breakpoints:

- col-xs-: This is for viewports below 544px (extra-small)
- col-sm-: This is for viewports of 544px or greater (small)
- col-md-: This is for viewports of 768px or greater (medium)
- col-lg-: This is for viewports of 992px or greater (large)

- `col-xl-`: This is for viewports of 1200px or greater (extra-large)

The Bootstrap 3 column breakpoints

Bootstrap 3 did not have `col-xl`. Furthermore, its four distinct breakpoints were:



- `col-xs-`: This was for viewports below 768px (extra-small)
- `col-sm-`: This was for viewports of 768px or greater (small)
- `col-md-`: This was for viewports of 992px or greater (medium)
- `col-lg-`: This was for viewports of 1200px or greater (large)

These classes are then appended with the number of columns an element should cover. Let's split the print sizes into five separate categories, namely: **Small**, **Medium**, **Large**, and **Extra Large**. As we know, a row is split into 12 columns. We have four print size categories, so we divide the number of columns by the number of elements, and that is the number of columns we want the element to cover. So, we append the number 3 to the `col-sm-` classname:

```
<div class="container">
    <h1>Our Print Sizes</h1>
    <div class="row">
        <div class="col-sm-3">Small</div>
        <div class="col-sm-3">Medium</div>
        <div class="col-sm-3">Large</div>
        <div class="col-sm-3">Extra Large</div>
    </div>
</div>
```

Check out the following screenshot:



Figure 2.7: Print Categories split it into even columns across the grid system

But again, on an extra small viewport we are going to see the elements stacked. That is, the elements will appear one on top of the other. But what if we do not want this to happen? What if we would like the elements to take up a different number of columns as the viewport size changes? Well luckily Bootstrap allows you to define the column widths for all breakpoints, and it will decide which rule to apply. Let's try the following:

```
<div class="container">
  <h1>Our Print Sizes</h1>
  <div class="row">
    <div class="col-xs-6 col-sm-3">Small</div>
    <div class="col-xs-6 col-sm-3">Medium</div>
    <div class="col-xs-6 col-sm-3">Large</div>
    <div class="col-xs-6 col-sm-3">Extra Large</div>
  </div>
</div>
```

We have retained `col-sm-3`, but now we have included `col-xs-6`. This means that for viewports below 544px wide, we want each element to take up 6 columns. This will result in the first two elements displaying on one line, and the last two below that.

On a viewport of 544px or wider, the categories appear in one horizontal row (as previously suggested, this is a drastic change from Bootstrap 3; with the previous version of Bootstrap, using the code, categories would appear in a horizontal row for viewports of 768px or wider). Look at the following screenshot:



Figure 2.8: The print sizes at a resolution above 544px

On a viewport of less than 544px wide, the categories are split across two rows. Observe the following screenshot:

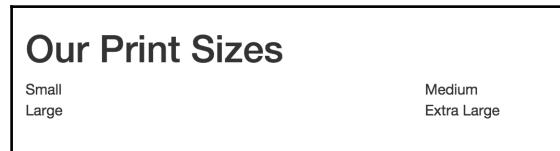


Figure 2.9: The print sizes at a resolution below 544px

Nesting

Not only does the grid system split rows horizontally into columns, it also allows you to split the columns vertically, by supporting nested rows. These nested rows themselves are split into 12 columns within the space provided by the parent column. There is nothing special needed in terms of mark up to achieve row inception. All that is needed to achieve this is to nest the elements appropriately and apply the row and column classes.

Let's organize our print sizes into the relevant categories. We want 12 size options, split equally into the four size categories. Each category will contain one row element with each print size taking up one column in the grid. Let's try the following:

```
<div class="container">
    <h1>Our Print Sizes</h1>
    <div class="row">
        <div class="col-xs-6 col-sm-3">
            <h5>Small</h5>
            <div class="row">
                <div class="col-sm-4">6x5</div>
                <div class="col-sm-4">8x10</div>
                <div class="col-sm-4">11x17</div>
            </div>
        </div>
        <div class="col-xs-6 col-sm-3">
            <h5>Medium</h5>
            <div class="row">
                <div class="col-sm-4">12x18</div>
                <div class="col-sm-4">16x20</div>
                <div class="col-sm-4">18x24</div>
            </div>
        </div>
        <div class="col-xs-6 col-sm-3">
            <h5>Large</h5>
            <div class="row">
                <div class="col-sm-4">19x27</div>
                <div class="col-sm-4">20x30</div>
                <div class="col-sm-4">22x28</div>
            </div>
        </div>
        <div class="col-xs-6 col-sm-3">
            <h5>Extra Large</h5>
            <div class="row">
                <div class="col-sm-4">24x36</div>
                <div class="col-sm-4">27x39</div>
                <div class="col-sm-4">27x40</div>
            </div>
        </div>
    </div>
```

```
</div>  
</div>
```

Check out the following screenshot:

Our Print Sizes											
Small		Medium				Large			Extra Large		
6x5	8x10	11x17	12x18	16x20	18x24	19x27	20x30	22x28	24x36	27x39	27x40

Figure 2.10: The print sizes using nesting

Within our category columns, we have nested a row. We split each row into three equal columns for viewports larger than or equal to 544px wide, using `col-sm-4`, to display the print sizes. Simple as that. Typically, it is good practice to ensure that the sum total of columns defined within the nested rows doesn't exceed the 12 columns allocated, as Bootstrap applies widths based on the assumption of 12 columns. Exceeding the 12 columns may result in unequal or unexpected column widths. However, on some occasions you may want to force a column onto another line at certain resolutions. For example, text content of columns may slightly overlap at certain resolutions.

In that case, we would like to force certain columns onto another line at a small resolution. To do this, we add `col-md-*` classes, and give the columns requiring a new line at 544px the class `col-sm-12`. Let's force the third size in the **Large** category onto its own line and force all **Extra Large** sizes onto separate lines. Let's try the following:

```
<div class="col-xs-6 col-sm-3">  
    <h5>Large</h5>  
    <div class="row">  
        <div class="col-sm-4">19x27</div>  
        <div class="col-sm-4">20x30</div>  
        <div class="col-sm-12 col-md-4">22x28</div>  
    </div>  
</div>  
<div class="col-xs-6 col-sm-3">  
    <h5>Extra Large</h5>  
    <div class="row">  
        <div class="col-sm-12 col-md-4">24x36</div>  
        <div class="col-sm-12 col-md-4">27x39</div>  
        <div class="col-sm-12 col-md-4">27x40</div>  
    </div>  
</div>
```

Observe the following screenshot:

Our Print Sizes							
Small		Medium			Large		Extra Large
6x5	8x10	11x17	12x18	16x20	18x24	19x27	20x30
						22x28	24x36
							27x39
							27x40

Figure 2.11: The print sizes with the “Extra Large” category forced onto a separate line for viewports below 544px

Nice and neat. If you have been paying attention, then you will have noticed that we do not actually need to define the resolutions below **Medium** if we want the elements to have separate lines, as this is the default behavior. We would only need to define it if we wanted a resolution below that (such as `xs`) to have a different behavior. So, this does the trick:

```
<div class="col-xs-6 col-sm-3">
    <h5>Large</h5>
    <div class="row">
        <div class="col-sm-4">19x27</div>
        <div class="col-sm-4">20x30</div>
        <div class="col-md-4">22x28</div>
    </div>
</div>
<div class="col-xs-6 col-sm-3">
    <h5>Extra Large</h5>
    <div class="row">
        <div class="col-md-4">24x36</div>
        <div class="col-md-4">27x39</div>
        <div class="col-md-4">27x40</div>
    </div>
</div>
```

Columns and flexbox



If the grid system has flexbox enabled, by setting `$enable-flex` to true as described previously, it is possible to have Bootstrap automatically set the column sizes to equal width. To do this simply use `col-*`, where * is the breakpoint. An example would be `col-xs`. Given two sibling elements in a row, both with the class `col-xs`, then both of those columns will automatically be given the same width.

The grid system also lets you order your columns independently of how they are ordered in the markup. Bootstrap 4 achieves this through the `pull-*-*` and `push-*-*` classes. These classes took the form of `col-*-*pull-*` and `col-*-*push-*` in Bootstrap 3.

Pulling and pushing

The `pull-*-*` and `push-*-*` classes allow for columns to be moved horizontally along their parent row. For instance, perhaps you wanted the **Extra Large** category to appear as the first category in certain resolutions. You would simply dynamically apply the appropriate push and pull classes to the appropriate columns. In this case, apply `push-sm-9` to the **Extra Large** column, as you are pushing the column 9 columns left, and `pull-sm-3` to the rest, as you are pulling those three columns to the right. Take a look at the following code snippet:

```
<div class="container">
    <h1>Our Print Sizes</h1>
    <div class="row">
        <div class="col-xs-6 col-sm-3 push-sm-3">
            <h5>Small</h5>
            <div class="row">
                <div class="col-sm-4">6x5</div>
                <div class="col-sm-4">8x10</div>
                <div class="col-sm-4">11x17</div>
            </div>
        </div>
        <div class="col-xs-6 col-sm-3 push-sm-3">
            <h5>Medium</h5>
            <div class="row">
                <div class="col-sm-4">12x18</div>
                <div class="col-sm-4">16x20</div>
                <div class="col-sm-4">18x24</div>
            </div>
        </div>
        <div class="col-xs-6 col-sm-3 push-sm-3">
            <h5>Large</h5>
            <div class="row">
                <div class="col-sm-4">19x27</div>
                <div class="col-sm-4">20x30</div>
                <div class="col-md-4">22x28</div>
            </div>
        </div>
        <div class="col-xs-6 col-sm-3 pull-sm-9">
            <h5>Extra Large</h5>
            <div class="row">
                <div class="col-md-4">24x36</div>
```

```
<div class="col-md-4">27x39</div>
<div class="col-md-4">27x40</div>
</div>
</div>
</div>
```

Observe the following screenshot:

Our Print Sizes											
Extra Large			Small			Medium			Large		
24x36	27x39	27x40	6x5	8x10	11x17	12x18	16x20	18x24	19x27	20x30	22x28

Figure 2.12: Using Bootstrap's pull-*-* to re-arrange the Extra Large category column

You may have noticed that in the markup, we have only applied sm pull and push classes, even though we have xs classes applied. The reason for that is simple. The push and pull classes only work on groups of columns that exist on a single horizontal plane. Pushing **Extra Large** 9 columns to the left will just force them out of the viewport completely. Pushing **Extra Large** 6 columns will only push the columns to the position of **Large**, an unfortunate shortcoming of this feature.

Offsetting

One neat feature of the grid system is how it allows you to create empty space within your row by using columns. If you wanted to list the categories and sizes, but for some reason you wanted to leave the space for **Medium** empty, in other grid systems you might need to add the empty elements to the markup to get the desired effect. For example:

```
<div class="container">
    <h1>Our Print Sizes</h1>
    <div class="row">
        <div class="col-xs-6 col-sm-3 push-sm-3">
            <h5>Small</h5>
            <div class="row">
                <div class="col-sm-4">6x5</div>
                <div class="col-sm-4">8x10</div>
                <div class="col-sm-4">11x17</div>
            </div>
        </div>
        <div class="col-xs-6 col-sm-3 push-sm-3">
        </div>
        <div class="col-xs-6 col-sm-3 push-sm-3">
            <h5>Large</h5>
        </div>
    </div>
</div>
```

```
<div class="row">
    <div class="col-sm-4">19x27</div>
    <div class="col-sm-4">20x30</div>
    <div class="col-md-4">22x28</div>
</div>
</div>
<div class="col-xs-6 col-sm-3 pull-sm-9">
    <h5>Extra Large</h5>
    <div class="row">
        <div class="col-md-4">24x36</div>
        <div class="col-md-4">27x39</div>
        <div class="col-md-4">27x40</div>
    </div>
</div>
</div>
</div>
```

Observe the following screenshot:

Our Print Sizes					
Extra Large	Small			Large	
24x36	27x39	27x40	6x5	8x10	11x17
				19x27	20x30
					22x28

Figure 2.13: Adding spacing between columns

While it has the desired effect, it is adding markup simply for the sake of layout, which isn't really what we want to do if we can avoid it. Bootstrap allows us to avoid it via the `offset` classes. The `offset` classes follow the same convention as the rest of the `column` classes, `offset-*-*-*`. Now, we can remove the empty layout elements and simply add the `offset` classes to the **Large** columns. Take a look at the following code:

```
<div class="col-xs-6 col-sm-3 push-sm-3">
    <h5>Small</h5>
    <div class="row">
        <div class="col-sm-4">6x5</div>
        <div class="col-sm-4">8x10</div>
        <div class="col-sm-4">11x17</div>
    </div>
</div>
<div class="col-xs-6 col-xs-offset-6 col-sm-3 offset-sm-3 push-sm-3">
    <h5>Large</h5>
    <div class="row">
        <div class="col-sm-4">19x27</div>
        <div class="col-sm-4">20x30</div>
        <div class="col-md-4">22x28</div>
    </div>
</div>
```

```
</div>
<div class="col-xs-6 col-sm-3 pull-sm-9">
    <h5>Extra Large</h5>
    <div class="row">
        <div class="col-md-4">24x36</div>
        <div class="col-md-4">27x39</div>
        <div class="col-md-4">27x40</div>
    </div>
</div>
```

Voila. The same result with less code. The goal we all aim to achieve.

With containers, rows, and columns, we can now reason about our layout more easily. By splitting a viewport into understandable chunks and concepts, the grid system gives us a structure to apply our content.

Image elements

As a next step, let us add an image to each column in our grid. Each image will act as a category heading, as well as allow us to display our photographic wares. The images used in the following part, and throughout the rest of the book, are provided with this book. Take a look at the following code:

```
<div class="col-xs-6 col-sm-3 push-sm-3">
    
    <h5>Small</h5>
    <div class="row">
        <div class="col-sm-4">6x5</div>
        <div class="col-sm-4">8x10</div>
        <div class="col-sm-4">11x17</div>
    </div>
</div>
<div class="col-xs-6 col-sm-3 push-sm-3">
    
    <h5>Medium</h5>
    <div class="row">
        <div class="col-sm-4">12x18</div>
        <div class="col-sm-4">16x20</div>
        <div class="col-sm-4">18x24</div>
    </div>
</div>
<div class="col-xs-6 col-sm-3 push-sm-3">
    
    <h5>Large</h5>
    <div class="row">
```

```
<div class="col-sm-4">19x27</div>
<div class="col-sm-4">20x30</div>
<div class="col-md-4">22x28</div>
</div>
</div>
<div class="col-xs-6 col-sm-3 pull-sm-9">
    
    <h5>Extra Large</h5>
    <div class="row">
        <div class="col-md-4">24x36</div>
        <div class="col-md-4">27x39</div>
        <div class="col-md-4">27x40</div>
    </div>
</div>
```

And that results in the following screenshot as seen in *Figure 2.14*:

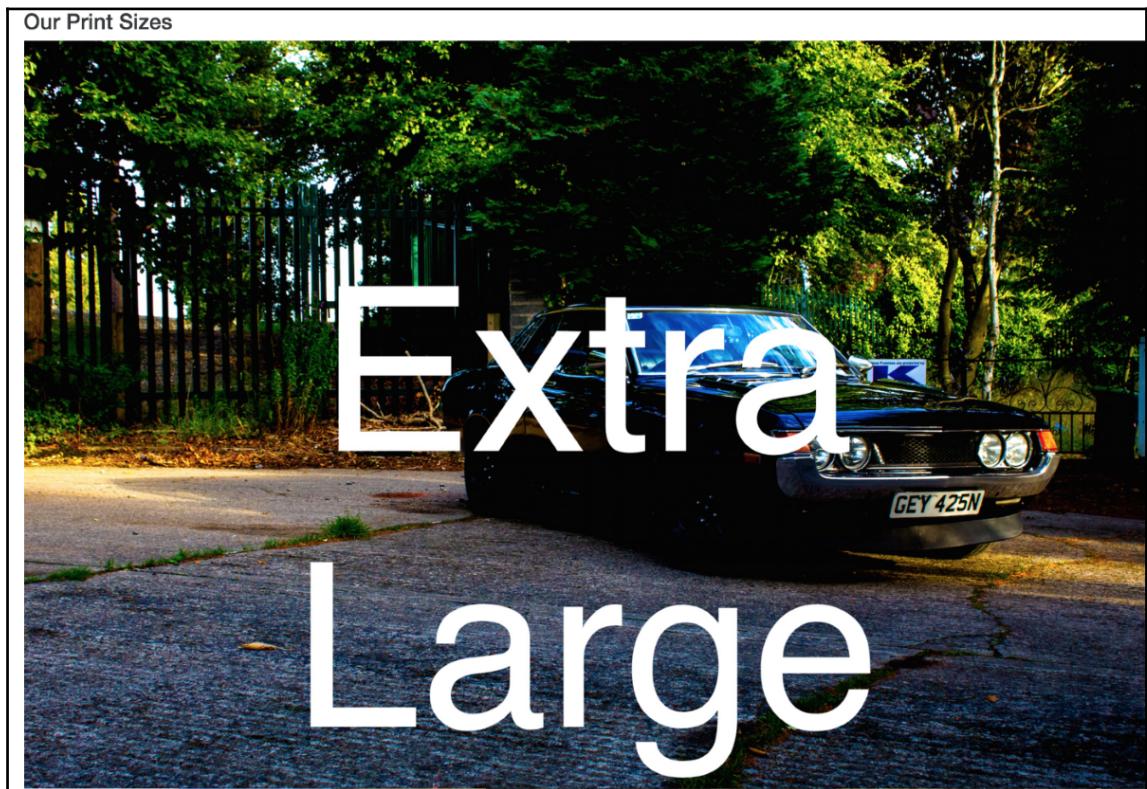


Figure 2.14: An unexpected outcome: Adding an image to the column in each grid results in the images failing to respect the boundaries of parent columns

That isn't what we expected. As you can see, images do not respect the boundaries of the parent column. Obviously, we can fix this with some styling, but we don't need to do that from scratch. Bootstrap comes with a class to handle this case, called `img-fluid`.

Responsive images

It is as straightforward as you would hope, just apply the `img-fluid` class to the element:

```
<div class="col-xs-6 col-sm-3 push-sm-3">
    
    <h5>Small</h5>
    <div class="row">
        <div class="col-sm-4">6x5</div>
        <div class="col-sm-4">8x10</div>
        <div class="col-sm-4">11x17</div>
    </div>
</div>
<div class="col-xs-6 col-sm-3 push-sm-3">
    
    <h5>Medium</h5>
    <div class="row">
        <div class="col-sm-4">12x18</div>
        <div class="col-sm-4">16x20</div>
        <div class="col-sm-4">18x24</div>
    </div>
</div>
<div class="col-xs-6 col-sm-3 push-sm-3">
    
    <h5>Large</h5>
    <div class="row">
        <div class="col-sm-4">19x27</div>
        <div class="col-sm-4">20x30</div>
        <div class="col-md-4">22x28</div>
    </div>
</div>
<div class="col-xs-6 col-sm-3 pull-sm-9">
    
    <h5>Extra Large</h5>
    <div class="row">
        <div class="col-md-4">24x36</div>
        <div class="col-md-4">27x39</div>
        <div class="col-md-4">27x40</div>
    </div>
</div>
```

Take a look at the following screenshot:

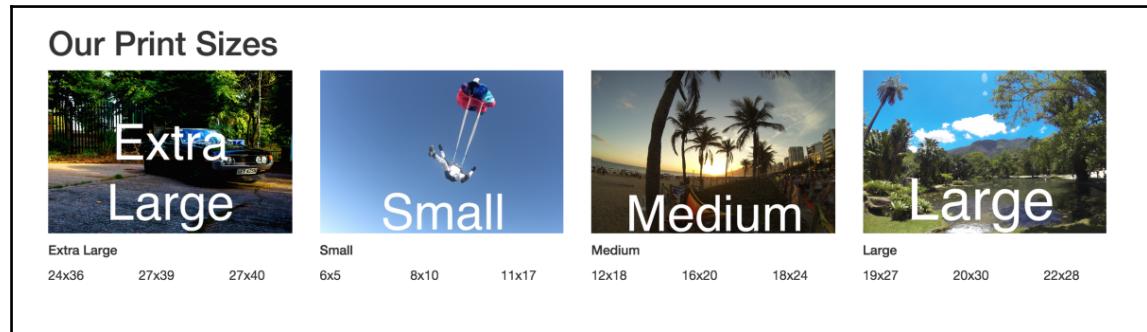


Figure 2.15: Making images responsive using Bootstrap's `img-fluid` class results in images that respect the boundaries of parent elements

That is more like it. `img-fluid` is exceedingly simple in itself, essentially just adding a `max-width: 100%` rule to the image element. Now, the `img` element will respect the boundaries of its parent.

Responsive images in Bootstrap 3



Images in Bootstrap 3 were made responsive using the `img-responsive` class. The `img-fluid` class in Bootstrap 4 is, in essence, the equivalent of `img-responsive`, just with a different name.

However, this simple approach also means that the feature is very basic. The browser still downloads the full resolution image, even though it may only be rendered at a fraction of the size. There are other libraries and services which help resolve the responsive images problem.

Bootstrap does, however, provide a neat mixin to help mitigate issues with retina displays. The `img-retina` mixin basically extends `background-image` and `background-size` rules, by allowing for two images and two sizes to be defined one for standard displays, and one for retina. `img-retina` takes the form:

```
.img-retina(std-res-img, hi-res-img, standard-width, standard-height)
```

For standard displays, `img-retina` will set the `background-image` to `std-res-img`, with the defined width and height. For retina display, `img-retina` will set `background-image` to `hi-res-img`, with the defined width and height values doubled.

For example, if we wanted to make sure that the **Extra Large** image loaded at high resolution on retina displays, we could give it a class `extra-large-image`, and apply that to a `div`:

```
<div class="extra-large-image"></div>
```

We would define `extra-large-image` as:

```
.extra-large-image {  
    .img-retina('/images/extra-large_std-res.png',  
    '/images/extra-large_hi-res.png', 700px, 400px)  
}
```

This will result in `/images/extra-large_std-res.png` being loaded with the dimensions `700 x 400` at standard resolution, and `/images/extra-large_hi-res.png` being loaded at `1400 x 800` on retina displays.

Image modifiers

Bootstrap also comes with some useful built-in image modifiers namely `img-rounded`, `img-thumbnail`, and `img-circle`. Let's apply these to the images in our example:

```
<div class="container">  
    <h1>Our Print Sizes</h1>  
    <div class="row">  
        <div class="col-xs-6 col-sm-3 push-sm-3">  
              
            <h5>Small</h5>  
            ...  
        </div>  
        <div class="col-xs-6 col-sm-3 push-sm-3">  
              
            <h5>Medium</h5>  
            ...  
        </div>  
        <div class="col-xs-6 col-sm-3 push-sm-3">  
              
            <h5>Large</h5>  
            ...  
    </div>
```

```
</div>
<div class="col-xs-6 col-sm-3 pull-sm-9">
    
    <h5>Extra Large</h5>
    ...
</div>
</div>
</div>
```

Take a look at the following screenshot:

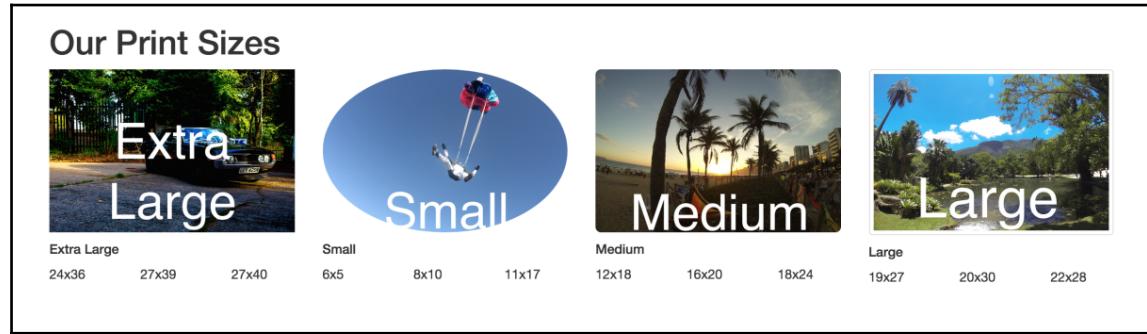


Figure 2.16: Applying Bootstrap's image modifiers: `img-rounded`, `img-circle` and `img-thumbnail`

You may notice that in the previously mentioned code, for the **Small** and **Medium** images, we have kept `img-fluid`, but removed `img-fluid` from **Large**. This is because `img-thumbnail` actually uses `img-fluid` as a mixin, while `img-circle` and `img-rounded` pay zero respect to the parent column width, so the `img-fluid` class is necessary. These images scale nicely down to `xs` displays, but it does look a little cluttered on a small viewport.

Take a look at the following screenshot:

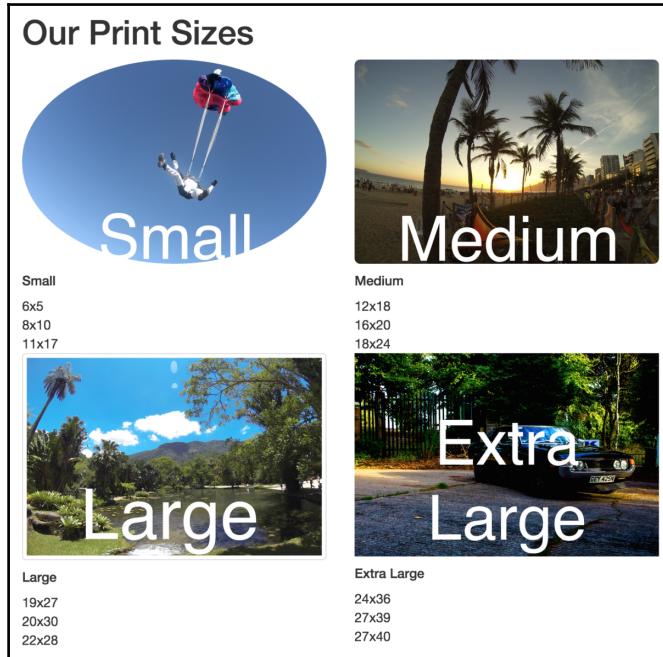


Figure 2.17: Dealing with smaller viewports by utilizing Bootstrap's responsive utilities

Bootstrap provides some really useful responsive utilities to tackle the issue of cluttered viewports.

Responsive utilities

Responsive utilities are a group of media query-based classes that control when an element should be hidden or shown depending on the viewport. One popular use case for this is controlling display specific navigations. For example, a page may have a navigation bar on large displays and have a hidden navigation on small displays which is only displayed when a user chooses to open the navigation.

Let's look at a quick example with our **Print Size** page. Add the `hidden-xs-down` class to the `img` elements:

```
<div class="container">
  <h1>Our Print Sizes</h1>
  <div class="row">
```

```
<div class="col-xs-6 col-sm-3 push-sm-3">
  
  <h5>Small</h5>
  ...
</div>
<div class="col-xs-6 col-sm-3 push-sm-3">
  
  <h5>Medium</h5>
  ...
</div>
<div class="col-xs-6 col-sm-3 push-sm-3">
  
  <h5>Large</h5>
  ...
</div>
<div class="col-xs-6 col-sm-3 pull-sm-9">
  
  <h5>Extra Large</h5>
  ...
</div>
</div>
</div>
```

Take a look at the following screenshot:

Our Print Sizes	
Small	Medium
6x5	12x18
8x10	16x20
11x17	18x24
Large	Extra Large
19x27	24x36
20x30	27x39
22x28	27x40

Figure 2.18: Hiding elements based on the viewport size using the hidden-xs-down class

The `hidden-xs-down` property hides the element only when the display is extra-small, according to Bootstrap's grid system.

Hiding in Bootstrap 3



Bootstrap 3 only offers `hidden-*` (where * refers to the viewport size. For example, `hidden-xs`). As such, to hide an element only when the display is extra-small, we would add the class `hidden-xs` to our element.

The `hidden` classes stick to the conventions found in the grid system, along with `hidden-xs-*`, there are `hidden-sm-*`, `-md-*`, and `-lg-*` classes. In the grid system, `col-md` targets all **Medium** displays and below. Likewise, `hidden-md-down` will target only **Medium** displays or displays smaller than **Medium** (that is, small and extra small).

Aside from `hidden-xs-down`, Bootstrap also offers the `hidden-*up` class, which hides an element if the viewport is at or above the threshold. For example, `hidden-lg-up` would hide the element when the display is **Large** or **Extra Large** (that is, unless the viewport is extra small, small or medium)

Let's apply the hidden classes so that:

- Images are hidden on `xs`, `sm`, and `md` displays.
- The page title is hidden on `md` or smaller displays
- Category titles are hidden on `lg` and bigger displays.

Observe the following code:

```
<div class="container">
    <h1 class="hidden-lg-down">Our Print Sizes</h1>
    <div class="row">
        <div class="col-xs-6 col-sm-3 push-sm-3">
            
            <h5 class="hidden-lg-up">Small</h5>
            ...
        </div>
        <div class="col-xs-6 col-sm-3 push-sm-3">
            
            <h5 class="hidden-lg-up">Medium</h5>
            ...
        </div>
        <div class="col-xs-6 col-sm-3 push-sm-3">
            
            <h5 class="hidden-lg-up">Large</h5>
            ...
        </div>
    </div>
</div>
```

```
</div>
<div class="col-xs-6 col-sm-3 pull-sm-9">
    
    <h5 class="hidden-lg-up">Extra Large</h5>
    ...
</div>
</div>
</div>
```

Viewing the page on a viewport smaller than 544px, the categories will be displayed over two rows, with the category title text instead of images. Have a look at the following screenshot:

Small	Medium
6x5	12x18
8x10	16x20
11x17	18x24
Large	Extra Large
19x27	24x36
20x30	27x39
22x28	27x40

Figure 2.19: Screenshot depicting our page with the title and image elements hidden at displays smaller than sm

Viewing the page on a viewport larger than 768px (md) and smaller than 992px (lg), the categories will be displayed over one row with both the category title text and images, as in Figure 2.20. Viewing the page on a viewport larger than 992px will remove the category title text. Take a look at the following screenshot:

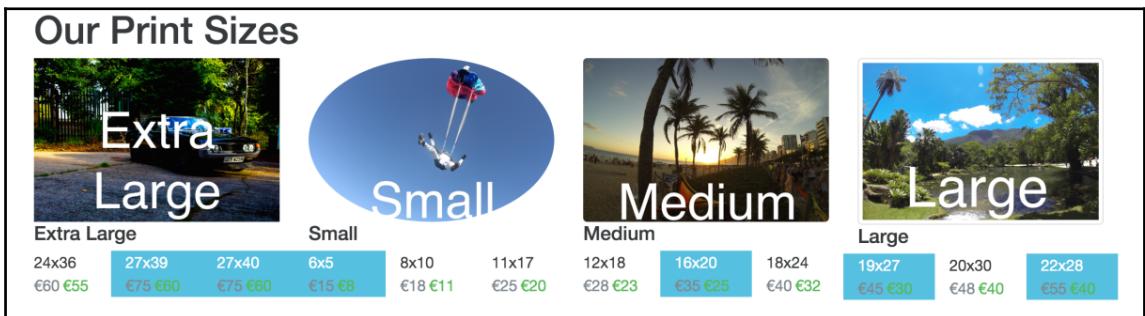


Figure 2.20: Screenshot depicting our page with the title element hidden for lg displays

Helper classes

Bootstrap also provides some utility classes, which Bootstrap refers to as helper classes. The helper classes provide some basic styles to accentuate information on a page. Their purpose is to give the user some context to the information they are receiving and to provide the developer with styling techniques, outside of the grid system.

Context

Bootstrap defines six context types to give a visual indicator to the user of what type of information is being conveyed: muted, primary, success, info, warning, and danger, as well as providing the developer with simple classes to assign context to elements via text color, `text-<context>` or background color, `bg-<context>`.

Let's apply some context to our example. We will add two prices to each size: a regular price and a special offer price. We will apply a `success` context to the special offer price and a `muted` context to the regular price. Print sizes with prices reduced by €10 or more will be given an `info` context background. The code should look similar to the following code snippet:

```
<div class="container">
    <h1 class="hidden-lg-down">Our Print Sizes</h1>
    <div class="row">
        <div class="col-xs-6 col-sm-3 push-sm-3">
            
            <h5 class="hidden-lg-down">Small</h5>
            <div class="row">
                <div class="col-sm-4 bg-info">6x5
                    <div class="row">
                        <div class="col-sm-3 text-muted">€15</div>
                        <div class="col-sm-3 text-success">€8</div>
                    </div>
                </div>
                <div class="col-sm-4">8x10
                    <div class="row">
                        <div class="col-sm-3 text-muted">€18</div>
                        <div class="col-sm-3 text-success">€11</div>
                    </div>
                </div>
                <div class="col-sm-4">11x17
                    <div class="row">
                        <div class="col-sm-3 text-muted">€25</div>
                        <div class="col-sm-3 text-success">€20</div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
```

```
</div>
</div>
</div>
</div>
<div class="col-xs-6 col-sm-3 push-sm-3">

...
</div>
```

Take a look at the following screenshot:

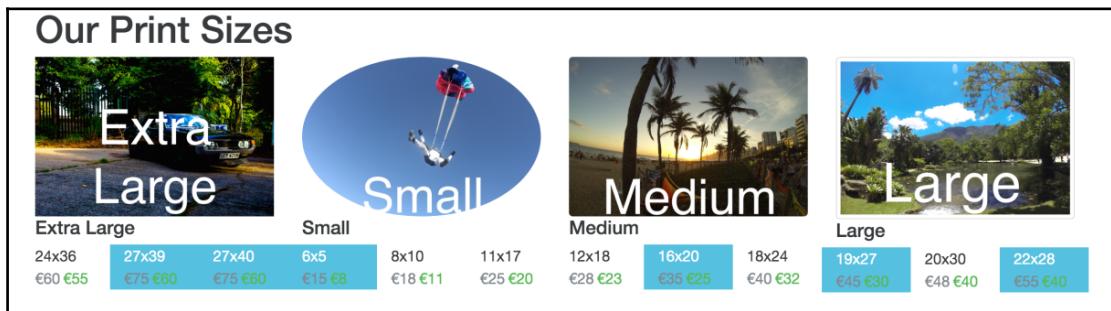


Figure 2.20: Bootstrap's context classes in action: note the changes in color for the various dimension and price blocks

As you can see, regular prices are gray, the special offer prices are green, and sizes with large discounts have a blue background.

Centering and floating

Bootstrap also provides handy classes for centering and floating elements.

Bootstrap 3 had a `center-block` class, which used margins to align an element centrally and sets the `display` property to `block`. This comes in very useful outside of the grid system. This has been removed from Bootstrap 4, in favor of two other classes, `d-block` and `m-x-auto`. The `d-block` class is responsible for setting the element's `display` property to `block`, while the `m-x-auto` class sets the margin properties to `auto`. As you may imagine, Bootstrap 4 also comes with the `d-inline` and `d-inline-block` classes, as well as an array of `m-*-*` classes for various margin options. More than that, there are also various padding options, with the `p-*-*` convention. Let us update our example with a new heading, **Special Offers**, inside a `div` we want to take up 50% of the viewport and we'll give it a primary background using the contextual classes.

Let's see it without the use of `m-x-auto` first. Observe the following code:

```
<div style="width:50%">
    <h1 class="bg-primary">Special Offers</h1>
</div>
<div class="container">
    <h1 class="hidden-lg-down">Our Print Sizes</h1>
    <div class="row">
```

Take a look at the following screenshot:

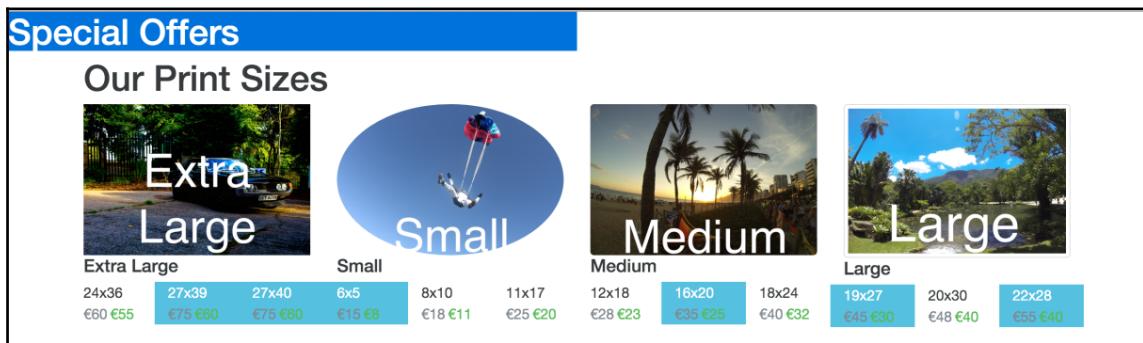


Figure 2.21: Applying Bootstrap's bg-primary class

Pretty ugly. Let's make it a little less ugly by centering the heading using `center-block` and `text-xs-center`. In Bootstrap 3, we would have used `center-block`, but as the element already has the `display: block` property, we can just use `m-x-auto` instead to control the margins. This can be done in the following manner:

```
<div class="m-x-auto" style="width:50%">
    <h1 class="bg-primary text-xs-center">Special Offers</h1>
</div>
```

For the time being, we have also used inline styles here to set the width of **Special Offers** to 50%. Inline styles should in general be avoided as they decrease maintainability of the code base. Ideally, we should use a class here to apply this style rule. We will address this in a later chapter. Let's check out the result in the following screenshot:

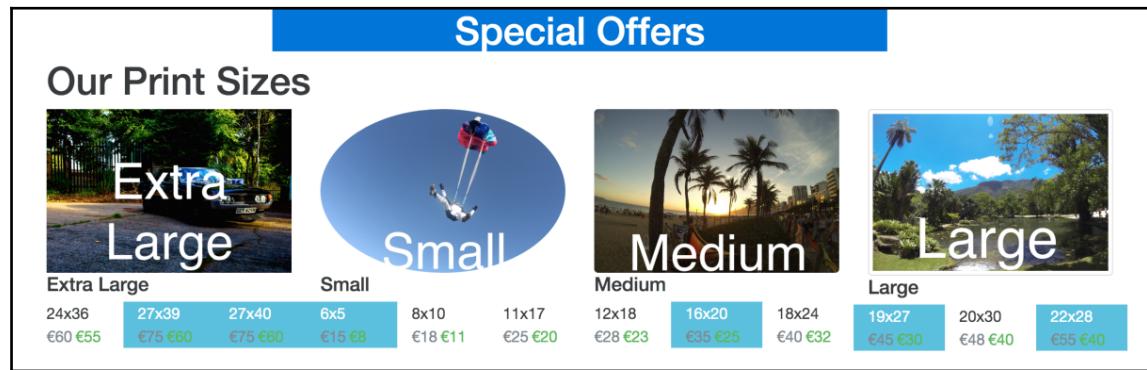


Figure 2.22: Applying Bootstrap's m-x-auto, text-xs-center and bg-primary classes

Nice, a little less ugly.

Along with this, Bootstrap gives us useful classes for floating elements left and right: `pull-*-right` and `pull-*-left`. These classes follow the same semantics as the previously discussed `hidden` classes and simply result in the appropriate `float` property being applied to the element together with an `!important` rule. The rules only get applied when the display is at or above the given breakpoint (the supported breakpoints are: `xs`, `sm`, `md`, `lg`, and `xl`).

Let's add a caveat to the page, indicating that **Terms and Conditions Apply**. We will use the `pull-xs-right` class to float it to the far right, the contextual `danger` class to indicate that it is worth noting to the user, and Bootstrap's `label` class to make the text look a little prettier. Since `pull-xs-right` floats elements to the right when the display is extra small or larger, the **Terms and Conditions** element will be floating to the right for all display sizes. Take a look at the following code:

```
<div class="bg-danger pull-xs-right">
    <p class="label">Temperatures may not be accurate</p>
</div>
```

Take a look at the following screenshot:

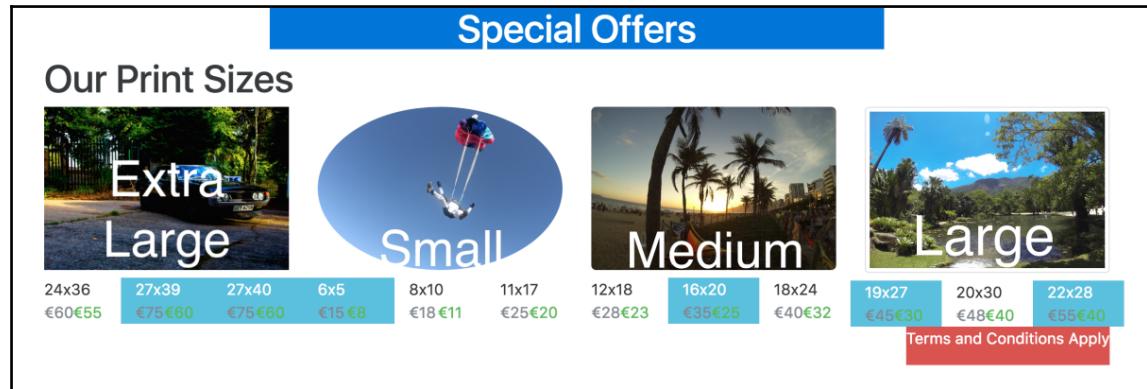


Figure 2.23: Using pull-xs-right to align an element to the right, for all display sizes

Nice and neatly docked to the right-hand side.

Aligning elements in Bootstrap 3



Bootstrap 3 only offers pull-left and pull-right, independent of display size. To align elements conditionally one would need to write custom CSS rules.

Toggling visibility

None of these utility classes are very complex. Nor would a developer not be able to implement the required logic themselves. However, they greatly help speed up development. The classes Bootstrap provides to toggle content visibility are a clear example of this. The `invisible` class simply sets the elements `visibility` property to `hidden`, with an `!important` rule. However, there are a few very useful display utility classes that are slightly more complex such as `text-hide`, `sr-only`, and `sr-only-focusable`.

The `sr-only` class hides an element except from screen-readers. `sr-only-focusable` extends this functionality to become visible when the element becomes focused, for instance in the case of keyboard access.

The `text-hide` class allows the developer to couple an image with a text element. The text will be read by a screen-reader, but by using `text-hidden`, it is not visible. Let's see this in action with the images in our example. Replace the `hidden-**` classes we added to our category text elements with `text-hide`. Have a look here:

```
<h5 class="text-hide">Small</h5>
```

Give it a try with a screen-reader to see (actually, hear!) the results.

Text alignment and transformation

Using the grid system, we learned how to lay out a website's basic building blocks. However, generally we need an additional level of control that allows us to more finely align the content contained within these building blocks. This level of control is provided Bootstrap's text alignment classes:

- `text-justify`: This justifies the text, so that it fills an entire area evenly (see *Figure 2.24*)
- `text-*-left`: This aligns the text in the element to which this class is applied to the left on viewports of size * or wider, where * can be one of the following: `xs`, `sm`, `md`, `lg`, and `xl`
- `text-*-right`: This aligns the text in the element to which this class is applied to the right on viewports of size * or wider, where * can be one of the following: `xs`, `sm`, `md`, `lg`, and `xl`
- `text-*-center`: This aligns the text in the element to which this class is applied to the center on viewports of size * or wider, where * can be one of the following: `xs`, `sm`, `md`, `lg`, `xl`

This can be observed in the following screenshot:

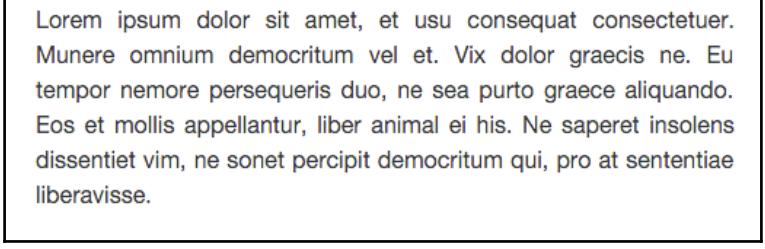


Figure 2.24 shows a rectangular box containing justified text. The text is in Latin and reads:
Lorem ipsum dolor sit amet, et usu consequat consectetur.
Munere omnium democritum vel et. Vix dolor graecis ne. Eu
tempor nemore persequeris duo, ne sea purto graece aliquando.
Eos et mollis appellantur, liber animal ei his. Ne saperet insolens
dissentiet vim, ne sonet percipit democritum qui, pro at sententiae
liberavisse.

Figure 2.24: An example of using the `text-justify` class to justify text

Aside from text alignment classes, Bootstrap also provides classes for transforming text contained within elements to which the classes are applied. Specifically:

- Text can be converted to all lower case by applying the `text-lowercase` class
- Text can be converted to all upper case by applying the `text-uppercase` class
- The first letter in each word can be capitalized by applying the `text-capitalize` class

The appearance (that is, its font weight) of text can be modified by applying `font-weight-normal`, `font-weight-bold`, and `font-italic`.

Summary

In this chapter, we covered the most important aspect of Bootstrap. We learned how Bootstrap's grid system operates, gaining an understanding of its many features. We have seen how we can leverage Bootstrap to manage responsive images, as well as how to target specific content to specific displays. On top of that, we have learned how to use the extremely useful Bootstrap helper classes. We also discovered a very nice feature of all the layout specific utility classes: the fact that they can also be used as mixins, and are indeed used as mixins within other Bootstrap classes. Leveraging these classes when we start writing custom style rules for `MyPhoto` will surely come in useful. As such, our understanding of Bootstrap is now quite deep, so let us go forward to the next chapter and build the layout for our `MyPhoto` page.

3

Building the Layout

In Chapter 2, *Making a Style Statement*, we learned about some of Bootstrap's core features, such as the grid system, and built out the print sizes component of MyPhoto. In this chapter, we are going to put this knowledge to further use by building out the layout of MyPhoto.

First, we will split the page into five sections, as we discussed in the previous chapters. These would be **Welcome**, **Services**, **Gallery**, **About**, and **Contact Us**, along with a footer placeholder. Here, we will use the grid system to create distinct sections on the page. We will add content to these sections using the Bootstrap components jumbotron, tabs, carousel, and wells.

Furthermore, we will discover how to integrate Bootstrap's **navbar** component into our page to provide single-page, app-like navigation, while also using it to surface Bootstrap **modal windows**.

As we learn about and implement these different components, we will also introduce how to customize the components to integrate with the styling conventions of MyPhoto.

In this chapter, we shall cover the following topics:

- Learning how to lay out a page using Bootstrap's grid system
- Learning how to integrate Bootstrap jumbotron, wells, tabs, and carousel
- Learning how to integrate Bootstrap's navbar
- Learning how to create and surface Bootstrap's modal windows
- Learning how to extend Bootstrap's component styles

Splitting it up

The MyPhoto webpage consists of five sections: **Welcome**, **Services**, **Gallery**, **About**, and **Contact Us**. The first thing we want to do is split our page into these distinct sections.

Each section will be a distinct container, so we can practically treat each section as a standalone component on the page. We want the container to take up 100% of the available horizontal space. Therefore, we will apply the `container-fluid` class. As we learned earlier, `container-fluid` allows its contents to be responsive across all resolutions, relying on the percentage width of the page, unlike the standard `container` class, which uses predefined horizontal breakpoints. To provide a visual cue that these are separate parts of the page, we will apply Bootstrap's **contextual background classes** to each section. We will also include a `footer` element in the page. The `footer` will contain miscellaneous bits of information, like legal information, but for now we will just include a placeholder:

```
<body>
  <div class="container-fluid bg-primary">
    <div class="row">
      <h3>Welcome</h3>
    </div>
  </div>
  <div class="container-fluid bg-info">
    <div class="row">
      <h3>Services</h3>
    </div>
  </div>
  <div class="container-fluid bg-success">
    <div class="row">
      <h3>Gallery</h3>
    </div>
  </div>
  <div class="container-fluid bg-warning">
    <div class="row">
      <h3>About</h3>
    </div>
  </div>
  <div class="container-fluid bg-danger">
    <div class="row">
      <h3>Contact Us</h3>
    </div>
  </div>
  <footer>Footer placeholder</footer>
</body>
```

So, we have five distinct containers, each with a distinct contextual background class and an inner row element, with an appropriate heading, along with a footer, producing the following screenshot:



Figure 3.1: Five distinct containers, each with a different contextual background

Okay, great. All sections of our site are in place. Let's add some custom CSS to give each section some space to breathe. Create a file, `styles/myphoto.css`, and add the following rules to set the minimum height of the section:

```
.myphoto-section {  
    min-height: 500px;  
}
```

The `myphoto-section` class will set the height of the section to a minimum height of `500px`. Add this class to each section, except for the footer:

```
<div class="container-fluid myphoto-section bg-primary">  
    <div class="row">  
        <h3>Welcome</h3>  
    </div>  
</div>
```

The addition of the `myphoto-section` class results in the following screenshot:



Figure 3.2: Five distinct containers with a minimum height of 500 pixels each

Now each section is easily distinguishable. The page bears a striking resemblance to a rainbow, though. Let's apply a color scheme here. We will apply a classic dark/light style, alternating between dark and light with each section. Update `myphoto.css` with the following classes:

```
.bg-myphoto-dark {  
    background-color: #504747;  
    color: white;  
}  
.bg-myphoto-light {  
    background-color: white;  
    color: #504747;  
}
```

These new classes adhere to the Bootstrap `.bg` naming convention and offer contrasting styles, a dark background with light text, and vice versa. Apply these to the container elements, in an even-odd manner:

```
<div class="container-fluid myphoto-section bg-myphoto-light">  
    <div class="row">  
        <h3>Welcome</h3>  
    </div>  
</div>  
<div class="container-fluid myphoto-section bg-myphoto-dark">  
    <div class="row">  
        <h3>Services</h3>  
    </div>  
</div>  
...
```

The **Welcome** section should now appear with a light background, the **Services** section with a dark background, and so on. Take a look at the following screenshot:



Figure 3.3: Five sections with alternating background colors

Pretty classy, right? Now, we have a responsive page, split into individual sections, powered by the grid system, with our own styling. Nothing wild here, but a nice start. Let's go ahead and add some content.

Adding Bootstrap components

MyPhoto is looking pretty bare right now, so let's take a look at integrating some Bootstrap components into our sections. First, let us add Bootstrap's JavaScript library. Bootstrap's JS relies upon jQuery UI, so let us add that too. Note that Bootstrap JS is not required for all the components we are going to use here, but it is best to get the setup out of the way. Install jQuery using Bower:

```
bower install jquery
```

Add the following code to the head of your page:

```
<script src="bower_components/jquery/dist/jquery.min.js"></script>
<script src="bower_components/bootstrap/dist/js/bootstrap.min.js">
</script>
```

With that, we're ready to use any Bootstrap component in our page.

Jumbotron

The first component we are going to integrate is Bootstrap's jumbotron.

The jumbotron component is essentially a very simple visual cue to draw attention to certain content. It is generally used as a splash to offer immediate information to the user. That is exactly what we will be doing with it.

In the **Welcome** section of our page, nest a `jumbotron` element into the `container` element:

```
<div class="container-fluid myphoto-section bg-myphoto-light">
    <div class="container">
        <div class="jumbotron">
            <h1>Welcome to MyPhoto</h1>
            <p>Photographs you can cherish!</p>
        </div>
    </div>
</div>
```

We have nested a container element in the container-fluid **Welcome** element. We do this so that our content is centered within the section, allowing for gutter space on the left and right. Within that container, we add a jumbotron element. The jumbotron class simply gives its element a hero-banner type style to attract the eye. The jumbotron itself acts in a similar way to a container, encompassing its own content. Here, we add a header and a paragraph to provide some context to the user. Take a look at the following screenshot:

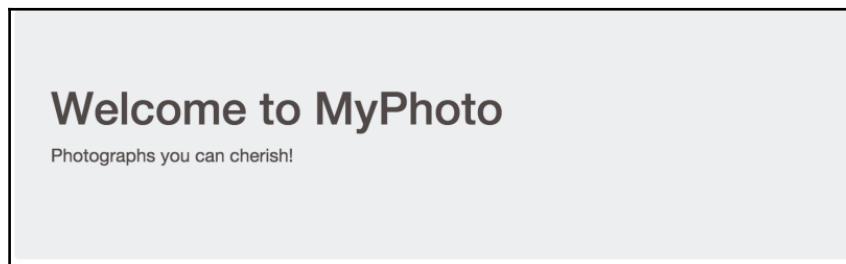


Figure 3.4: The Bootstrap jumbotron class to create a banner welcoming the user to MyPhoto

Oh, not great. The jumbotron has worked as expected, but it is not grabbing our attention as we would hope against a light background. Being a photographer's page, maybe we should add some pictorial content here? We can use jumbotron to display as a hero-image. Add a new class to `myphoto.css`:

```
.jumbotron-welcome {  
    background-image: url('/images/skyline.png');  
    background-size: cover;  
    color: white;  
}
```

This class will simply apply a background image and a light font color to an element. Apply the class to our jumbotron element:

```
<div class="jumbotron jumbotron-welcome">
```

Take a look at the following screenshot:

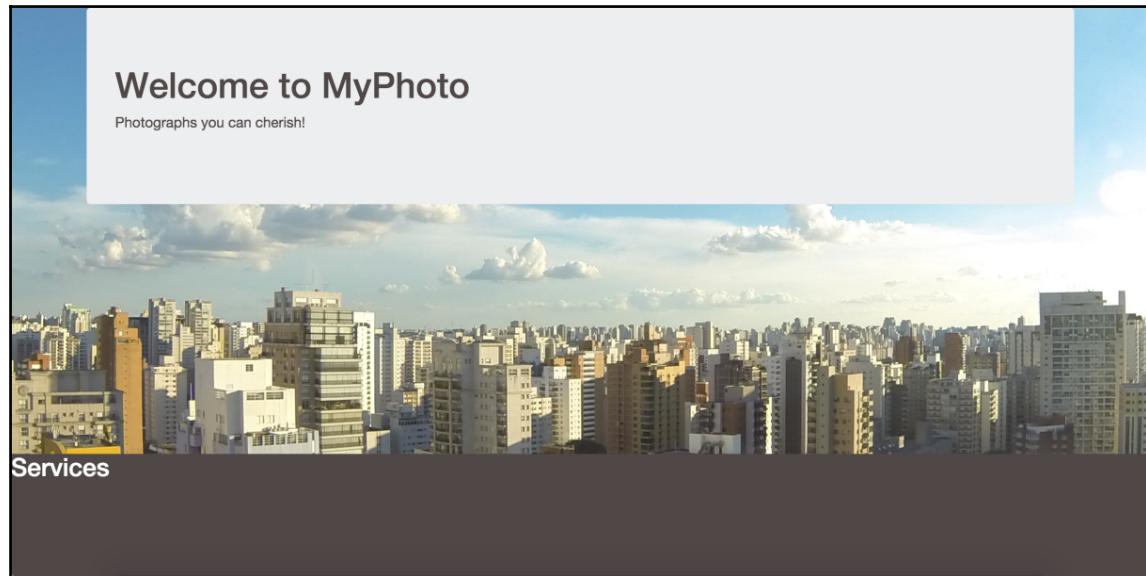


Figure 3.5: Using Sao Paulo's skyline as a background image for MyPhoto's welcome banner

Pretty nice. Let's flip that, and apply the background image to the **Welcome** section container element, removing the `bg-myphoto-light` class. Add the following class to `myphoto.css`:

```
.bg-myphoto-welcome {  
    background-image: url('/images/skyline.png');  
    background-size: cover;  
    color: #504747;  
}
```

Then, update the **Welcome** section to the following:

```
<div class="container-fluid myphoto-section bg-myphoto-welcome">  
    <div class="container">  
        <div class="jumbotron">  
            <h1>Welcome to MyPhoto</h1>  
            <p>Photographs you can cherish!</p>  
        </div>  
    </div>  
</div>
```

Our new class again applies the background image. It also applies a dark font color, which will be inherited by the jumbotron. We have removed the `jumbotron-welcome` class from the `jumbotron` element.

Excellent! We have added some photographic content and our jumbotron attracts attention.

Tabs

MyPhoto offers a trio of services. We built the UI for one of these, the **Print** service, in Chapter 2, *Making a Style Statement*. The UI displayed the prices for various print sizes. Obviously, we will want to integrate the **Print** service into the **Services** section of the MyPhoto page. We also need to include the other two services, namely, **Events**, where a photographer may take photographs of a customer's event such as a wedding or a conference, and **Personal**, where a photographer may take family photos and so on.

Rather than build the services into three separate interfaces, we are going to develop them using Bootstrap's `tabs` component.

Tabs are a part of Bootstrap's `nav` family of components. Tabs provide an elegant way of displaying and grouping related content. Under the surface, `tabs` are simply list elements linked to `div` elements, representing the tab contents, with some nice styling applied. Tabs are one of the Bootstrap components that require `jQuery` and `bootstrap.js` to function correctly.

First, we will add the tab list to the **Services** section:

```
<ul class="nav nav-tabs nav-justified">
    <li class="nav-item">
        <a href="#services-events" data-toggle="tab" class="nav-link active">Events</a>
    </li>
    <li class="nav-item">
        <a href="#services-personal" data-toggle="tab" class="nav-link">Personal</a>
    </li>
    <li class="nav-item">
        <a href="#services-prints" data-toggle="tab" class="nav-link">Prints</a>
    </li>
</ul>
```

We're wrapping the **Services** header in a `container` element, so we follow the same convention as the **Welcome** section and leave gutter space around the content. Within that container, we also include the `tabs`. We create an unordered list element (`ul`) with the `nav` base class, and the `nav-tabs` class to apply the tab styling. The list elements are then created, with the tab heading text and a `href` to the relevant tab content. The `nav-item` class is applied to each list item to denote that the item is indeed a navigation item (the `nav-item` class adjusts the margin and float of the list item, and ensures that the list items are not stacked on top of each other, but instead appear next to one another). The anchor element that is used to reference the tab's corresponding section is given a `nav-link` class. This `nav-link` class adjusts the following:

- The anchor's padding, so that it displays nicely as a tab link.
- It removes the default anchor focus and hover text decoration.
- It changes the anchor's display to `inline-block`. This has two important effects on the element:
 - The browser renders any white space contained within the element.
 - The element is placed inline, but maintains the properties of a block-level element.

The `active` class is applied to the default selected list item's anchor element. Last but not least, the anchor's `data-toggle` attribute is set to `tab`. Our **Services** section now looks like this:



Figure 3.6: Using tabs components to build out the MyPhoto's Services section

Navigation tabs in Bootstrap 3



Nav tabs in Bootstrap 3 require slightly more work than when using Bootstrap 4. For one, Bootstrap 3 does not have the `nav-item` and `nav-link` classes. Instead, we create an unordered-list element (`ul`) with the `nav` base class, namely, `nav-tabs` class, to apply the tab styling, and `nav-justified` to center the text within the tab headings. The list elements are then created, with the tab heading text and a `href` to the relevant tab content.



The `active` class is applied to the default selected list item:

```
<ul class="nav nav-tabs nav-justified" role="tablist">
  <li role="presentation" class="active"><a href="#services-events" role="tab" data-toggle="tab">Events</a></li>
  <li role="presentation"><a href="#services-personal" role="tab" data-toggle="tab">Personal</a></li>
  <li role="presentation"><a href="#services-prints" role="tab" data-toggle="tab">Prints</a></li>
</ul>
```

Note that the `nav-justified` class has been removed from Bootstrap 4.

We now have some selectable tabs, with inappropriate styling and no content. Let's resolve the content issue first. We will add `Lorem Ipsum` text to the **Events** and **Personal** tab content, and we will use an `Our Print Sizes` table for the **Prints** tab content:

```
<div class="tab-content bg-myphoto-light">
  <div role="tabpanel" class="tab-pane active" id="services-events">
    <div class="container">
      <div class="row">
        <p>Lorem Ipsum</p>
      </div>
    </div>
  </div>
  <div role="tabpanel" class="tab-pane" id="services-personal">
    <div class="container">
      <div class="row">
        <p>Lorem Ipsum</p>
      </div>
    </div>
  </div>
  <div role="tabpanel" class="tab-pane" id="services-prints">
    <div class="m-x-auto" style="width:50%">
      <h1 class="bg-primary text-xs-center">Special Offers</h1>
    </div>
    <div class="container">
      <h1 class="hidden-lg-down">Our Print Sizes</h1>
      <div class="row">
        <div class="col-xs-6 col-sm-3 push-sm-3">
          <h5 class="text-hide">Small</h5>
          <div class="row">
            <div class="col-sm-4 bg-info">6x5
              <div class="row">
                <div class="col-sm-3 text-muted">€15</div>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
```

```
        <div class="col-sm-3 text-success">€8</div>
    </div>
</div>
<div class="col-sm-4">8x10
    <div class="row">
        <div class="col-sm-3 text-muted">€18</div>
        <div class="col-sm-3 text-success">€11</div>
    </div>
</div>
<div class="col-sm-4">11x17
    <div class="row">
        <div class="col-sm-3 text-muted">€25</div>
        <div class="col-sm-3 text-success">€20</div>
    </div>
</div>
</div>
<div class="col-xs-6 col-sm-3 push-sm-3">
    <h5 class="text-hide">Medium</h5>
    <div class="row">
        <div class="col-sm-4">12x18
            <div class="row">
                <div class="col-sm-3 text-muted">€28</div>
                <div class="col-sm-3 text-success">€23</div>
            </div>
        </div>
        <div class="col-sm-4 bg-info">16x20
            <div class="row">
                <div class="col-sm-3 text-muted">€35</div>
                <div class="col-sm-3 text-success">€25</div>
            </div>
        </div>
        <div class="col-sm-4">18x24
            <div class="row">
                <div class="col-sm-3 text-muted">€40</div>
                <div class="col-sm-3 text-success">€32</div>
            </div>
        </div>
    </div>
</div>
<div class="col-xs-6 col-sm-3 push-sm-3">
    <h5 class="text-hide">Large</h5>
    <div class="row">
        <div class="col-sm-4 bg-info">19x27
            <div class="row">
                <div class="col-sm-3 text-muted">€45</div>
                <div class="col-sm-3 text-success">€30</div>
            </div>
        </div>
    </div>
</div>
```

```
</div>
<div class="col-sm-4">20x30
    <div class="row">
        <div class="col-sm-3 text-muted">€48</div>
        <div class="col-sm-3 text-success">€40</div>
    </div>
</div>
<div class="col-md-4 bg-info">22x28
    <div class="row">
        <div class="col-sm-3 text-muted">€55</div>
        <div class="col-sm-3 text-success">€40</div>
    </div>
    </div>
</div>
<div class="col-xs-6 col-sm-3 pull-sm-9">
    <h5 class="text-hide">Extra Large</h5>
    <div class="row">
        <div class="col-md-4">24x36
            <div class="row">
                <div class="col-sm-3 text-muted">€60</div>
                <div class="col-sm-3 text-success">€55</div>
            </div>
        </div>
        <div class="col-md-4 bg-info">27x39
            <div class="row">
                <div class="col-sm-3 text-muted">€75</div>
                <div class="col-sm-3 text-success">€60</div>
            </div>
        </div>
        <div class="col-md-4 bg-info">27x40
            <div class="row">
                <div class="col-sm-3 text-muted">€75</div>
                <div class="col-sm-3 text-success">€60</div>
            </div>
        </div>
    </div>
</div>
<div class="bg-danger pull-xs-right">
    <p class="label">Terms and Conditions Apply</p>
</div>
</div>
</div>
```

We add a `tab-content` element to wrap the three services, which are declared with a `tab-pane` class. The `tab-pane` classes have an `id` attribute matching the `href` attribute of the corresponding tab heading, and each of the `tab-pane` elements has a `container` element for the contents of the pane. Now, clicking on the the **Prints** tab will surface **Our Print Sizes** chart. We have modified the print sizes chart for its new context within the `tabs` component. Take a look at the following screenshot:

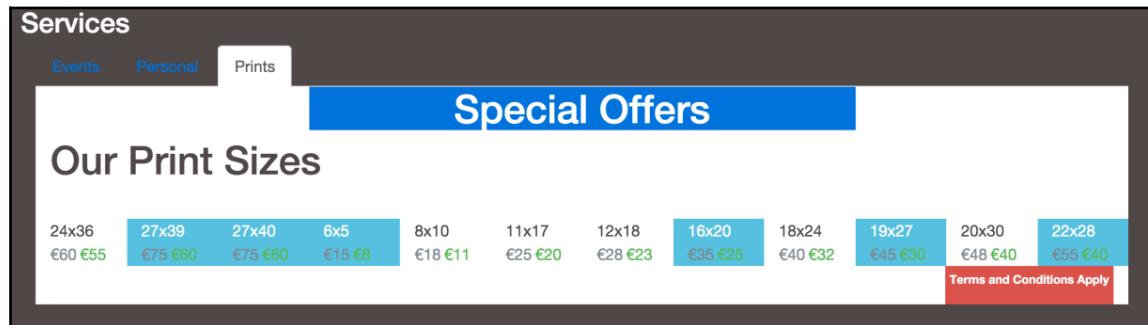


Figure 3.7: Adding tab content using the tab-pane class

Let's quickly add some styling to the tab headings and panels to make them slightly more readable. Add the following classes to `myphoto.css`:

```
.bg-myphoto-dark .nav-tabs a {  
    color: white;  
}
```

Save and refresh. Our **Services** section now looks like the following:



Figure 3.8: Improving the look and feel of our tabs by setting the tab title color to white

With those changes, our **Services** tabs are a little more pleasing to the eye. The contents of the tabs can now be fleshed out within their own specific container elements.

Carousel

To exhibit the photographic wares, we need a gallery. To implement the gallery feature, we are going to integrate Bootstrap's `carousel` component. The carousel acts as a slideshow, with a list of nested elements as the slides.

Let's add a `carousel`, with three slides, to the **Gallery** section:

```
<div class="container-fluid myphoto-section bg-myphoto-light">
  <div class="container">
    <div class="row">
      <h3>Gallery</h3>
      <div id="gallery-carousel" class="carousel slide"
        data-ride="carousel" data-interval="3000">
        <div class="carousel-inner" role="listbox">
          <div style="height: 400px" class="carousel-item active">
            
            <div class="carousel-caption">
              Brazil
            </div>
          </div>
          <div style="height: 400px" class="carousel-item">
            
            <div class="carousel-caption">
              Datsun 260Z
            </div>
          </div>
          <div style="height: 400px" class="carousel-item">
            
            <div class="carousel-caption">
              Skydive
            </div>
          </div>
        </div>
        <a class="left carousel-control" href="#gallery-carousel"
          role="button" data-slide="prev">
          <span class="icon-prev" aria-hidden="true"></span>
        </a>
        <a class="right carousel-control" href="#gallery-carousel"
          role="button" data-slide="next">
          <span class="icon-next" aria-hidden="true"></span>
        </a>
        <ol class="carousel-indicators">
          <li data-target="#gallery-carousel" data-slide-
            to="0" class="active"></li>
          <li data-target="#gallery-carousel" data-slide-
            to="1"></li>
```

```
<li data-target="#gallery-carousel" data-slide-to="2"></li>
</ol>
</div>
</div>
</div>
</div>
```

Take a look at the following screenshot:



Figure 3.9: Using the Bootstrap carousel to display a gallery of images. Note the arrows at both sides of the images. These are the carousel controls that allow users to navigate the image gallery.

Now, we have a fully functioning, three-slide gallery. Let's breakdown what is happening here.

The carousel in Bootstrap 3

When using the carousel feature after switching to Bootstrap 4, having used Bootstrap 3, there are two changes to be wary of. The first is that the `carousel-item` class was just called `item` class in Bootstrap 3. The second is that Bootstrap 4 comes with the `icon-next` and `icon-prev` classes that should be used with the carousel controls. A typical Bootstrap 3 carousel control may look as follows:



```
<a class="left carousel-control" href="#gallery-carousel" role="button" data-slide="prev">
  <span class="glyphicon glyphicon-chevron-left" aria-hidden="true"></span>
</a>
<a class="right carousel-control" href="#gallery-carousel" role="button" data-slide="next">
```



```
<span class="glyphicon glyphicon-chevron-right" aria-hidden="true"></span>
</a>
```

First, we declare the `carousel` parent tag:

```
<div id="gallery-carousel" class="carousel slide" data-ride="carousel" data-interval="4000"></div>
```

We give the `div` an `id` that can be referenced by its nested elements, such as the `carousel` indicators. We use Bootstrap's `carousel` and `slide` classes. The `data-ride="carousel"` attribute indicates that the `carousel` is to automatically initialize on page load. The `data-interval` attribute indicates that the slides should change every 4000 ms. There are other options such as `data-pause`, which will indicate whether or not the `carousel` should pause on hover. The default value is `hover`; to prevent pausing, set this option to `false`. The `data-wrap` is a Boolean attribute, to indicate whether the `carousel` should be a continuous loop, or end once the last slide has been reached. The default value for `data-wrap` is `true`. The `data-keyboard` is also a Boolean attribute, to indicate whether or not the `carousel` is keyboard-controllable. The default value for `data-keyboard` is `true`.

Then, we add the actual slides. The slides are nested within a `carousel-inner` element, to contain the slides. The slides are also `div` elements, with the `carousel-item` class, and the `active` class to indicate which slide to show on initialization:

```
<div style="height: 400px" class="carousel-item active">
  
  <div class="carousel-caption">
    Skydiving
  </div>
</div>
```

Within the `div`, we have an `image` element to act as the main content for the slide, and it has a sibling `carousel-caption` element, which is exactly what it sounds like, a caption for the slide. The `carousel-caption` element can contain nested elements.

Next, we add the right and left arrows for navigating through the slides:

```
<a class="left carousel-control" href="#gallery-carousel" role="button" data-slide="prev">
  <span class="icon-prev" aria-hidden="true"></span>
</a>
```

These are simple anchor tags, leveraging Bootstrap's directional and `carousel-control` classes. The `data-slide` attribute indicates whether we want to cycle backward or forward through the list of slides. The `data-slide` can take the value `prev` for previous, or `next`. Nested within the anchor tag is a span simply applying the `icon-prev` and `icon-next` class as a directional indicator.

Finally, we declare the `carousel-indicators`:

```
<ol class="carousel-indicators">
  <li data-target="#gallery-carousel" data-slide-to="0"
    class="active"></li>
  <li data-target="#gallery-carousel" data-slide-to="1"></li>
  <li data-target="#gallery-carousel" data-slide-to="2"></li>
</ol>
```

The indicators are circles layered on the slideshow, indicating which slide is currently active. For example, if the second slide is active, then the second circle will be filled. It is mandatory to indicate which slide is active on initialization by setting `class="active"` on that element. The `data-slide-to` attribute indicates which slide the circle relates to, so if a user clicks a circle with `data-slide-to="2"`, the third slide becomes active, as the count for the slides begins at 0. Some Bootstrap framework ports, for example Angular Bootstrap, will automatically generate the carousel indicators based on the number of slides the carousel contains, but using Vanilla Bootstrap, the list has to be created and maintained manually.

With that, we now have a fully functioning carousel as our **Gallery**, with very little markup, thanks to the power of Bootstrap.

Cards

The final section we are going to look at in this chapter is the **About** section. Here, we are going to use Bootstrap's `card` component to highlight our `MyPhoto` marketing blurb. Take a look at the following code:

```
<div class="container-fluid myphoto-section bg-myphoto-dark">
  <div class="container">
    <div class="row">
      <h3>About</h3>
      <div class="card bg-myphoto-light">
        <div class="card-block">
          <p>The style of photography will be customised to your personal preference, as if being shot from your own eyes. You will be part of every step of the photography process,</p>
    </div>
  </div>
</div>
```

```
to ensure these photos feel like your own.</p>
<p>Our excellent photographers have many years of experience,
capturing moments from weddings to sporting events of absolute
quality.</p>
<p>MyPhoto also provides superb printing options of all shapes
and sizes, on any canvas, in any form. From large canvas prints
to personalised photo albums, MyPhoto provides it all.</p>
<p>MyPhoto provides a full, holistic solution for all your
photography needs.</p>
</div>
</div>
</div>
</div>
</div>
```

Take a look at the following screenshot:

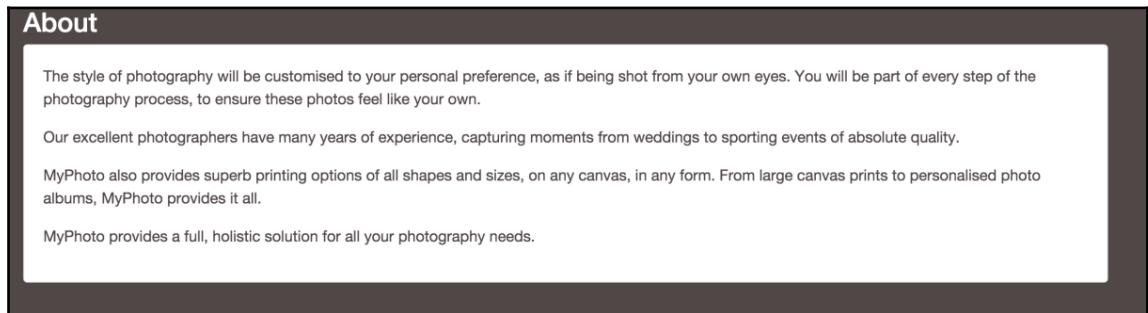


Figure 3.10: Using Bootstrap cards to display About information

There isn't much to say about cards. The `card` class gives an impression of an element being inset, to draw the eye of the user and to provide a visual cue of relatively important content. It achieves this effect by:

- Giving the element a 1 px border, along with a 0.25 rem border radius to which it is applied
- Adjusting the element's bottom margin, forcing some space between the `card` element and any subsequent elements
- Ensuring that the element's position is relative and making the element behave as a block-level element (that is, forcing the element onto a row of its own)

Along with a card, we use `card-block` to provide larger padding around the content area. The card is as effective as it is simple, and complements the content nicely.

Now that our sections have some content, we need to add some navigation and branding. To do this, we are going to leverage Bootstrap's navbar component.

Cards in Bootstrap 3

Cards are a new concept, introduced with Bootstrap 4. Prior to Bootstrap 4, cards did not exist. The feature replaced Bootstrap 3's wells, thumbnails, and panels. To achieve a similar effect to the one demonstrated in *Figure 3.10*, we would use the `well` and `well-lg` classes:



```
<div class="container-fluid myphoto-section bg-myphoto-dark">
  <div class="container">
    <div class="row">
      <h3>About</h3>
      <div class="well well-lg bg-myphoto-light">
        <p>....</p>
        <p>...</p>
      </div>
    </div>
  </div>
</div>
```

Navbar

Bootstrap's navbar is a powerful, responsive component to provide sensible navigation around a website. Before getting into the details, let's add it to our page and see what happens:

```
<nav class="navbar navbar-light">
  <a class="navbar-brand" href="#">MyPhoto</a>
  <ul class="nav navbar-nav">
    <li class="nav-item"><a class="nav-link" href="#welcome">Welcome</a></li>
    <li class="nav-item"><a class="nav-link" href="#services">Services</a></li>
    <li class="nav-item"><a class="nav-link" href="#gallery">Gallery</a></li>
    <li class="nav-item"><a class="nav-link" href="#about">About</a></li>
    <li class="nav-item"><a class="nav-link" href="#contact">Contact Us</a></li>
  </ul>
```

```
</nav>
```

Take a look at the following screenshot:

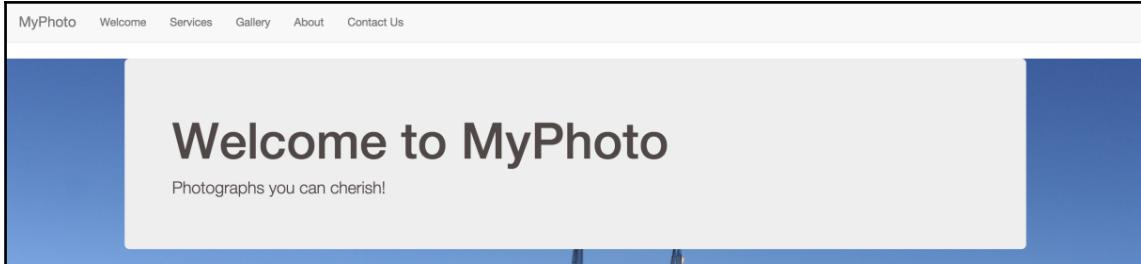


Figure 3.11: Using Bootstrap's navbar to provide navigational links to MyPhoto's Welcome, Services, Gallery, About, and Contact Us sections

With that, we have a navigation for our single page site. Let's break it down.

For accessibility and semantic purposes, we use the `nav` tag as the parent element. We could use the `role="navigation"` attribute on a `div` element, but using `nav` is semantically clearer. We apply the `navbar` class, which is where Bootstrap starts its magic. The `navbar` class sets the dimensions and positioning of the navigation bar. The `navbar-light` property applies some styling to keep in line with Bootstrap's default style guide.

The first element in the `nav` family is a `navbar-brand` element. This element is used to apply styling to differentiate the brand of the page, such as a specific logo or font style, to the other elements within the `nav`.

We then have a `ul` element as a sibling to `navbar-brand`. Within this `ul`, we nest a list of list item elements that contain the anchors to various sections on our page. Each list item element has a `nav-item` class applied to it; each anchor has a `nav-link` class. The former adjusts the element's margin and ensures the element's positioning within the container by setting its `float` property. The latter adjusts the anchor element's styling, removing its text decoration and default colors.

If you click on **Services**, nothing happens. This is because we also need to update the sections with the corresponding `id` for each section. Let's do that now. For example, add `id="services"` to the **Welcome** section:

```
<div class="container-fluid myphoto-section bg-myphoto-welcome" id="services">
```

Now, if we click on **Services** in the `nav`, the browser scrolls us down to the **Services** section.



Navigation in Bootstrap 3

The entire navbar component has been rewritten in Bootstrap 4 in an effort to make the navbar simpler and easier to use. Bootstrap 4 introduced the `nav-item` class for use on the list items, and added the `nav-link` class for use with the anchor element. Neither of these two aforementioned classes exist in Bootstrap 3.

Take a look at the following screenshot:



Figure 3.12: Clicking on Services in the nav, the browser scrolls us down to the Services section. Observe how the string “#services” is appended to the URL in the address bar.

In its current state, the navigation list wraps onto a new line, which wraps when the list is too long for the given resolution. Take a look at the following screenshot:

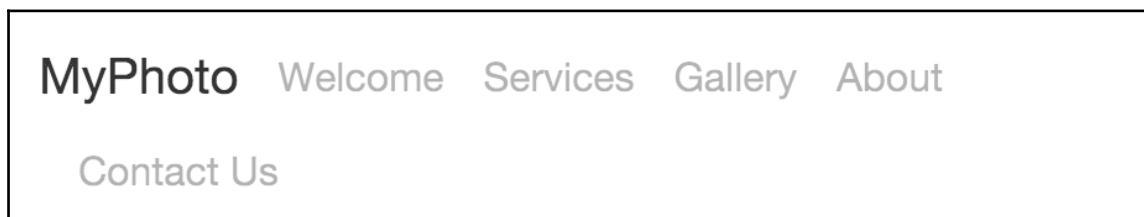


Figure 3.13: The navbar items wrapping onto a new line on smaller resolutions

Not very Bootstrap-like. Of course, Bootstrap provides a straightforward solution for this. Bootstrap's solution is to use the *hamburger* interface at lower resolutions. First, let's add the hamburger. Within the `nav` element, we add the following code:

```
<button class="navbar-toggler hidden-sm-up pull-xs-right"  
type="button" data-toggle="collapse" data-target="#navigation">  
≡  
</button>
```

Take a look at the following screenshot:

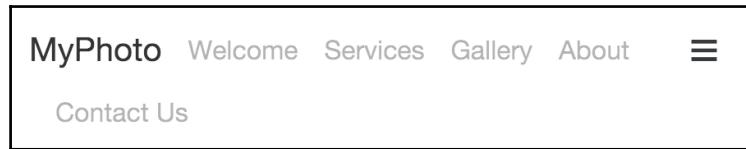


Figure 3.14: Adding a hamburger button to our navigation bar

What is a hamburger?



A hamburger, in web development, refers to a specific style of button or icon, representing a list item. Represented as three horizontal and parallel lines (resembling a hamburger), a hamburger button generally surfaces navigation options when selected. The hamburger has become popular and useful for condensing navigations on small displays.

At lower resolutions, the hamburger now appears on the right-hand side. However, our list of sections still wraps onto a new line and the hamburger button has no effect. In order to hook the hamburger and the links together, first we add an `id` attribute to the parent `div` of the links, and then reference that `id` in a `data-target` attribute in the `hamburger` button element. We also apply the `collapse` and `navbar-collapse` classes to the link's parent element. These classes effectively make the navigation links disappear at smaller resolutions:

```
<div class="collapse navbar-collapse" id="navigation">
  <button type="button" class="navbar-toggle collapsed"
    data-toggle="collapse" data-target="#navigation"
    aria-expanded="false">
```

Take a look at the following screenshot:

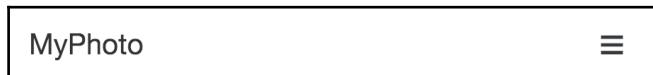


Figure 3.15: Making the navigation links disappear on smaller resolutions

When a user clicks on the hamburger, the navigation menu will be surfaced. Take a look at the following screenshot:

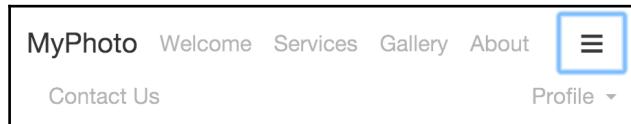


Figure 3.16: Making the navigation links visible on smaller resolutions by clicking the hamburger menu

Now, on smaller viewports, the list of links is no longer visible, the hamburger icon appears, and the list of links appears once the button has been activated. This provides a much better experience on smaller resolutions.

Now that we have our sections and navigation working, let's add a drop-down menu to our navigation to provide the user with the secondary features **Profile**, **Settings**, and **Log Out**.

To do this, we are going to use Bootstrap's dropdown class. As a child of the list of section links, create a new list item element with Bootstrap's `nav-item` class. As previously noted, this defines the element as a Bootstrap navigation item and renders it alongside the other items within our navbar. To the same list item, we also apply the `dropdown` class and `pull-xs-right` class (recall that the latter aligns the list item element to the right of the container). Within the list element, we include an anchor tag, responsible for surfacing the drop-down navigation:

```
<li class="nav-item dropdown pull-xs-right">
  <a href="#" class="nav-link dropdown-toggle" data-toggle="dropdown" role="button"
     aria-haspopup="true" aria-expanded="false">
    Profile <span class="caret"></span>
  </a>
  <div class="dropdown-menu dropdown-menu-right">
    <a class="dropdown-item" href="#" data-toggle="modal"
       data-target="#profile-modal">
      Profile
    </a>
  </div>
</li>
```

Take a look at the following screenshot:

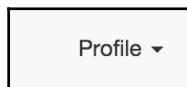


Figure 3.17: A drop-down menu item

Great, it looks nice and neat on the right-hand side of our page. Of course, it doesn't do anything yet. We need to create the menu we want it to surface. As a child of the **Profile** anchor tag, add a `div` element, this time with Bootstrap's `dropdown-menu` class, to denote this is the element we want the dropdown to surface, with four list elements:

```
<li class="nav-item dropdown pull-xs-right">
  <a href="#" class="nav-link dropdown-toggle" data-toggle=
    "dropdown" role="button"
      aria-haspopup="true" aria-expanded="false">
        Profile <span class="caret"></span>
    </a>
    <div class="dropdown-menu dropdown-menu-right">
      <a class="dropdown-item" href="#" data-toggle="modal"
        data-target="#profile-modal">
          Profile
      </a>
      <a class="dropdown-item" href="#" data-toggle="modal" data-target=
        "#settings-modal">
          Settings
      </a>
      <div class="dropdown-divider"></div>
      <a class="dropdown-item" href="#">
          Logout
      </a>
    </div>
  </li>
```

Take a look at the following screenshot:

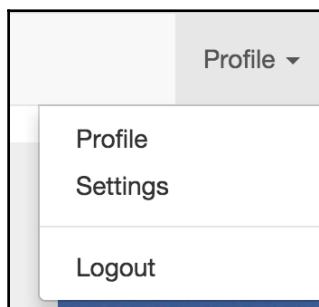


Figure 3.18: The expanded drop-down menu item, listing three sub-items

We also add a `dropdown-menu-right` class to align the drop-down menu to the right. Now, we have a list of links in a drop-down menu, and as you can see, we've leveraged Bootstrap's `dropdown-divider` class to provide a visual cue to the separation between **Profile** functionality, and logging out of the site.

The dropdown class itself simply sets the CSS position property to `relative`. There are several other classes specifying how the drop-down menu, and other items within the drop-down, such as the divider, look and behave. All these style definitions can be found in `bootstrap/scss/_dropdown.scss`. The interactions and behaviors of the `dropdown` component are actually handled and triggered by an out-of-the-box Bootstrap library, `dropdown.js`. The `dropdown.js` uses CSS selectors to hook into dropdown-specific elements, and toggles CSS classes to execute certain behaviors. You can find the `dropdown` library at `bower_components/bootstrap/js/dist/dropdown.js`.

Drop-down menus in Bootstrap 3



Notice the difference in how drop-down menus are created in Bootstrap 4. To implement the same drop-down menu functionality using Bootstrap 3, we would need another unordered-list element as a child of the **Profile** anchor tag. We would then only apply the `dropdown-menu` class. The individual menu items would be represented as list items. Furthermore, Bootstrap 4 introduced the `dropdown-divider` class, replacing the Bootstrap 3 `divider` class. The appearance of the two is the same, however, and both serve as a visual cue to the separation between **Profile** functionality, and logging out of the site.

As nice as the dropdown is, it doesn't actually do anything. Rather than have the user leave the page to use secondary functionality, let's render the functionality in modal windows instead.



What is a modal window?

A modal window in web development is an element that is layered on top of the main webpage content to give the impression of a separate window, which requires user interaction and prevents the user from using the main content.

Bootstrap makes surfacing modal windows extraordinarily simple. It is simply a case of creating the modal markup, using Bootstrap's modal classes, and using HTML attributes, `data-toggle`, `data-target`, and `data-dismiss` to surface and remove the modal.

First, let's update the Profile and Settings element with the data-toggle and data-target attributes:

```
<a class="dropdown-item" href="#" data-toggle="modal" data-target="#profile-modal">  
    Profile  
</a>  
<a class="dropdown-item" href="#" data-toggle="modal" data-target="#settings-modal">  
    Settings  
</a>
```

We set data-toggle="modal" to tell Bootstrap we want to open a modal, and data-target is a reference to the id of the modal we want to surface. Let's go ahead and create these modals.

As a sibling to the nav element, we add the following modal markup:

```
<div class="modal" id="profile-modal" role="dialog">  
    <div class="modal-dialog" role="document">  
        <div class="modal-content">  
            <div class="modal-header">  
                <button type="button" class="close" data-dismiss="modal"  
                    aria-label="Close"><span aria-hidden="true">&times;</span>  
                </button>  
                <h4 class="modal-title" id="profile-modal-label">Profile</h4>  
            </div>  
            <div class="modal-body">  
                Profile  
            </div>  
        </div>  
    </div>  
    <div class="modal" id="settings-modal" role="dialog">  
        <div class="modal-dialog" role="document">  
            <div class="modal-content">  
                <div class="modal-header">  
                    <button type="button" class="close" data-dismiss="modal"  
                        aria-label="Close"><span aria-hidden="true">&times;</span>  
                    </button>  
                    <h4 class="modal-title" id="settings-modal-label">Settings</h4>  
                </div>  
                <div class="modal-body">  
                    Settings  
                </div>  
            </div>  
        </div>  
    </div>
```

```
</div>
```

Let's explain what is happening in this piece of code. We apply the `modal` class to the parent element. The `id` of the element corresponds to the value of the `data-target` attribute of the element we want to use to surface the modal. Within that `div`, we use Bootstrap's modal classes to define nested elements to represent the modal itself (`modal-dialog`), the content of the modal (`modal-content`), the header of the modal (`modal-header`) and the body of the modal (`modal-body`). We also define the title of the modal using `modal-title`, and a **Close** button. The **Close** button includes a `data-dismiss="modal"` attribute, which allows the user to close the modal just by clicking anywhere off the modal. Let's take a look at how the `Profile` modal renders. Take a look at the following screenshot:

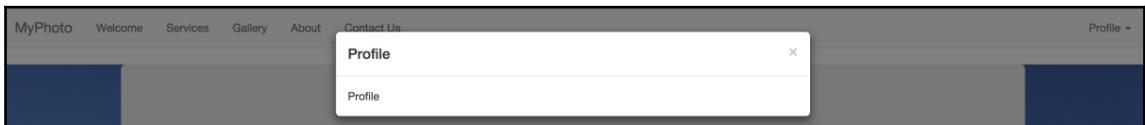


Figure 3.19: Using Bootstrap's modal to display an empty dialog

Pretty nice, considering it required very little markup.

We have a pretty nice navigation in place now. Well, a functional navigation. It does not really fit in well with the rest of our site, though. Let's create our own custom navigation style for `MyPhoto`.

Styling

Bootstrap's navbar comes with two built-in styles: `navbar-light`, which we currently use, and `navbar-dark`. To add a new style, we could add a new class to `_navbar.scss`, add new color variables to `_variables.scss`, recompile Bootstrap, and then apply the class to our `navbar` element. However, applying changes directly to Bootstrap source files is bad practice. For instance, we use Bower to add Bootstrap, along with other third-party components, to our project. If we were to add new styles to Bootstrap source files, we would need to add Bootstrap to our repository and use that version, instead of Bower, or else other developers on your team would not get the changes. Also, if we wanted to use a new version of Bootstrap, even a minor or patch release, it would force us to reapply all our custom changes to that new version of Bootstrap.

Instead, we are going to apply the changes to the MyPhoto stylesheet. Add the following rules to myphoto.css:

```
.navbar-myphoto {  
    background-color: #2A2A2C;  
    border-color: black;  
    border-radius: 0;  
    margin-bottom: 0px;  
}  
.navbar-myphoto.navbar-brand {  
    color: white;  
    font-weight: bold;  
}  
.navbar-myphoto.navbar-nav > li > a {  
    color: white;  
}  
.navbar-myphoto.navbar-nav > li > a:hover {  
    background-color: #2A2A2C;  
    color: gray;  
}  
.navbar-myphoto.navbar-nav > li > a:focus {  
    background-color: #504747;  
    color: gray;  
}  
.navbar-myphoto.navbar-nav > li.active > a {  
    background-color: #504747;  
    color: gray;  
}  
.navbar-myphoto.dropdown-menu {  
    background-color: #504747;  
    border-color: black;  
}  
.navbar-myphoto.dropdown-menu > a {  
    color: white;  
    background-color: #504747;  
}  
.navbar-myphoto.dropdown-menu > a:hover {  
    color: gray;  
    background-color: #504747;  
}  
.navbar-myphoto.dropdown-menu > a:focus {  
    color: gray;  
    background-color: #504747;  
}  
.navbar-myphoto.dropdown-menu > .active > a:focus {  
    color: gray;  
    background-color: #504747;  
}
```

```
.navbar-myphoto > .navbar-toggler {  
    color: white;  
}  
.nav-pills > .active > a, .nav-pills > .active > a:hover {  
    background-color: grey;  
}  
.nav-pills.nav-link.active, .nav-pills.nav-link.active:focus,  
.nav-pills.nav-link.active:hover,  
.nav-pills.nav-item.open.nav-link,  
.nav-pills.nav-item.open.nav-link:focus,  
.nav-pills.nav-item.open.nav-link:hover {  
    background-color: grey;  
}
```

We have added a new class, `navbar-myphoto`. The `navbar-myphoto` uses the same dark background as `bg-myphoto-dark` and removes the `margin-bottom` navbar applied by default. We then have to apply rules for the classes that are usually nested within a `navbar` element. We apply font and background rules across these classes, in their various states, to keep inline with the general style of MyPhoto. These rules could be made more succinct using a preprocessor, but we are not going to add that complexity to this project.

In the markup, replace `navbar-light` with `navbar-myphoto`. Take a look at the following screenshot:

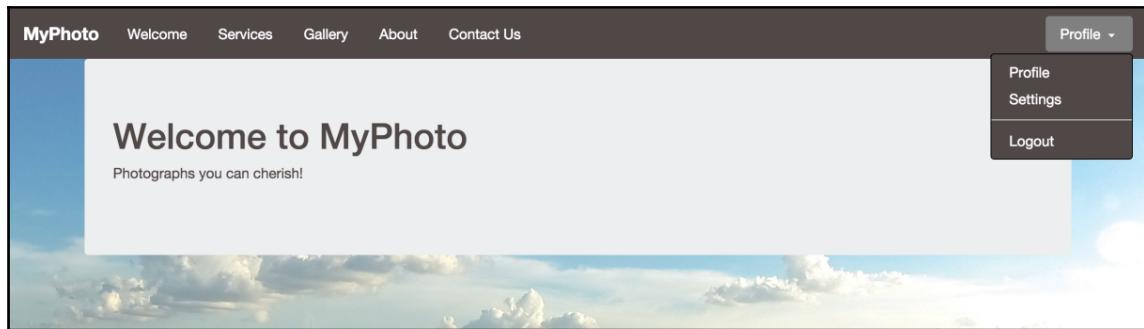


Figure 3.20: The custom-styled navigation bar with drop-down

With these new styles, we now have a style-appropriate navbar for MyPhoto. Pretty neat.

Last but not least, let's try and improve the overall look and feel of our website by adding some nice fonts. We will use two freely available Google Fonts. Specifically, we will use Poiret One (<https://www.google.com/fonts/specimen/Poiret+One>) for our navbar and headers, and Lato (<https://www.google.com/fonts/specimen/Lato>) for everything else. To be able to use these fonts, we must first include them. Insert the following two lines into the head of our `index.html`:

```
<link href="https://fonts.googleapis.com/css?family=Poiret+One" rel="stylesheet" type="text/css">
<link href="https://fonts.googleapis.com/css?family=Lato&subset=latin,latin-ext" rel="stylesheet" type="text/css">
```

Then, insert the following lines of code into `myphoto.css`:

```
h1, h2, h3, h4, h5, h6, .nav, .navbar {
    font-family: 'Poiret One', cursive;
}
body {
    font-family: 'Lato', cursive;
}
```

Take a look at the following screenshot:

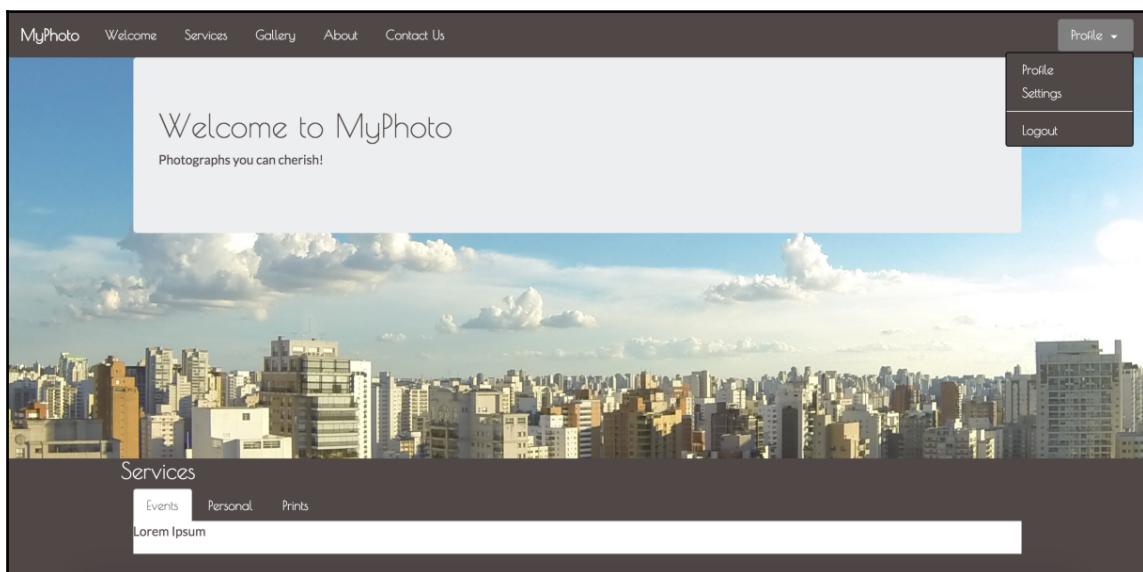


Figure 3.21: MyPhoto styled using Google Fonts: Poiret One and Lato

Summary

In this chapter, we defined the skeleton of our MyPhoto webpage using the knowledge of Bootstrap's grid system we gained from Chapter 2, *Making a Style Statement*, while also integrating some of Bootstrap's core components. We lightly customized those components with our own bespoke CSS to fit in with the design of the page, while also exploring how to link the functionality of these components together. With a functioning MyPhoto page, we can now move on to Chapter 4, *On Navigation, Footers, Alerts, and Content*, and learn how to customize these components even further.

4

On Navigation, Footers, Alerts, and Content

In the previous chapter, we built our website's basic skeleton. Using Bootstrap's grid system, we structured our website into five distinct sections. We then styled these sections and learned how to use Bootstrap's navbar and tab system to make these sections navigable. In this chapter, we will continue adding to the knowledge obtained in [Chapter 3, Building the Layout](#) by leveraging even more Bootstrap components, adding more content and streamlining our website's design. We will begin by improving our navbar. We will first learn how to fix our navbar's position. We will then use a Bootstrap plugin (Scrollspy) to automatically update the navbar tab item appearance based on the user's navigation. Next, we will customize the website's scrolling behavior, making the transition between sections smoother.

Once we have improved our website's navigation, we will focus on improving and customizing our website's overall look and feel. That is, we will learn how to apply and customize alerts, and how to use buttons and brand images. We will also discover how to style different text elements, use media objects, and apply citations and figures.

As we progress through this chapter, we will be examining each of the aforementioned components individually. This way, we will have the chance to see how they are actually composed under the hood.

To summarize, in this chapter we shall do the following:

- Learn how to fixate our navbar
- Use the Bootstrap plugin Scrollspy to improve our website's navigation
- Use icons to customize and improve the overall design of our website
- Introduce Bootstrap alerts and customize them

- Style our website's footer
- Apply buttons to improve our website's overall usability

Fixating the navbar

Our website already looks pretty decent. We have a navigation bar in place, a footer placeholder, and various sections populated with some sample content. But we are not quite there yet. The website's overall user experience is still a bit edgy, and does not yet feel very refined. Take user navigation, for instance. While clicking on a navbar link, indeed does take the user to the correct section, the navbar disappears once we navigate across the sections. This means that the navbar loses its purpose. It no longer facilitates easy navigation across the different sections of our website. Instead, the user will need to scroll to the top of the page every time they wish to use the navbar. To make the navbar persistent, append `navbar-fixed-top` to the `class` attribute of our `nav` element:

```
<nav class="navbar navbar-myphoto navbar-fixed-top">  
  ...  
</nav>
```

Save, refresh, and scroll. Voila! Our navbar now remains fixed at the top of our page. The `navbar-fixed-top` works as follows:

- The element's position is set to `fixed`. This positions the element relative to the browser window (*not* relative to other elements), meaning that the element will remain in the same place, regardless of the positioning of other elements on the page.
- The element's `top` is set to 0. This means that the distance between the navbar and the top of the browser window is 0.

In addition to some minor margin and padding changes, `navbar-fixed-top` also changes the element's `z-index` to 1,030, therefore ensuring that the element will appear above all other elements on the page (that is, all elements that have a `z-index` of less than 1,030).

Did you know?



Did you know that, within web development, another term for persistency is *sticky*? For example, instead of asking “How can I make my navbar persistent?”, you often hear people asking “How do I make my navbar *sticky*?“.

Should you desire to fixate the navbar at the bottom of the page, you can use the `navbar-fixed-bottom` class. This class behaves in exactly the same way as the `navbar-fixed-top` class, except that, instead of setting the element's top to 0, it sets the `bottom` property to 0, thereby ensuring that the element resides at the bottom of the page.

If we wanted to quickly change the color of the navbar without wanting to write a whole bunch of custom rules, then we could apply the `navbar-*` and `bg-*` classes:

- `navbar-dark`: This is used to indicate that the navbar's foreground color should be adjusted to match a dark background. As such, the rule will apply a white foreground color to all navbar items.
- `navbar-light`: This is the opposite of the aforementioned `navbar-dark`, and applies a dark foreground color in order to support a light background.
- `bg-*`: This will set the background color to that of the desired context class (we will cover the various context classes later on in this chapter). For example, `bg-primary`, `bg-success`, and `bg-info`. `bg-inverse` mimics an inverted background, setting the background color to `#373a3c`.

Take a look at the following screenshot:



Figure 4.1: The MyPhoto navbar with three different context styles: `pg-primary`, `bg-warning`, and `bg-danger`.

Improving navigation using Scrollspy

Now that we have fixated the navbar at the top of the page, it will no longer disappear as the user scrolls down the page. But now we are faced with a new problem. As the user clicks on a navbar item, they have no way of telling which section of the site they are visiting, or whether the section has loaded successfully without actually examining the section's content. Wouldn't it be nice if the user could tell which section they are currently visiting by looking at the navbar? To this end, we would need to find a way of highlighting the navbar items based on the user's navigation. For example, if the user is browsing the **Services** section, then we would need to update the color of the **Services** navbar item.

Luckily, Bootstrap comes equipped with a plugin that allows us to automatically update the navbar items based on the user's navigation. Take a look at the following screenshot:



Figure 4.2: Currently, all navbar items are the same color – there is no way for the user to tell where on the page they are currently just by looking at the navbar itself.

Meet Scrollspy. Scrollspy allows us to automatically update navigation targets based on the user's current position within the page. In other words, Scrollspy allows us to denote scrollable areas within our page and update elements to denote when the user enters a specific scrollable area. It also allows us to add scroll animations to our page. We'll see more about this later.

Scrollspy is already included in our installed Bootstrap build, so there is no need for any additional downloads. The first thing that we must do is denote our scrollable area.

By adding `data-spy="scroll"` to the element that contains our contents, we are, in essence, telling Bootstrap to use anything contained within the given element as our scrollable area. Typically, `data-spy="scroll"` is added to the body element of our page:

```
<body data-spy="scroll">  
...  
</body>
```

The `data-spy` attribute is by far the most important attribute when using Scrollspy, as without it, Bootstrap wouldn't know what to do with the remainder of the Scrollspy instructions. It is also important to note that Scrollspy will only work with `container` elements whose position is relative.

Next, we must tell Scrollspy how to identify our navbar element. After all, how else will the plugin know where the navbar items that it has to update are located? It is important that you point to the parent element that contains the list of navbar items. If you point to any other element, the feature will not work. Before we can tell Scrollspy about our list of nav items, we must first be able to uniquely identify them. To do so, we will use our navbar's class name, `navbar`. Alternatively, we could assign our navbar an `id` by adding `id="navbar"` to our `navbar` element. We will then make Scrollspy aware of the navbar by using the `data-target` attribute:

```
<body data-spy="scroll" data-target=".navbar">  
  <nav class="navbar navbar-fixed-top navbar-myphoto">  
    <div class="collapse navbar-toggleable-xs" id="navigation">  
      <ul class="nav nav-pills">  
        ...  
      <ul/>
```

```
</div>
</nav>
...
</body>
```

Note that, when using the `data-target` attribute, class names must be preceded by a `.` (period), while each `id` must be preceded by `#` (hash).

Save and refresh. Now try to navigate between sections and observe the navbar's behavior. Items are automatically highlighted (see *Figure 4.3*) as you change position. Take a look at the following screenshot:

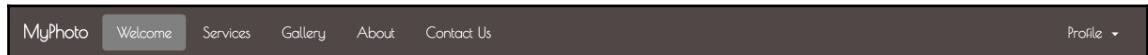


Figure 4.3: Navbar items are highlighted, depending on the user's current position within the page.

Customizing scroll speed

Great! Our navbar items are now automatically updated based on the user's scroll position. Easy, huh? But we're not quite done yet. The transition between sections actually feels quite rough. The page literally jumps from one section to another. Such a jerky movement is not very pleasing. Instead, we should improve the user experience of our website by making this transition between sections smoother. We can quite easily accomplish this by customizing the scroll speed of our page by using jQuery. The first step in this task involves automating the scroll, that is, forcing a scroll event to a target on the page using jQuery, without the user needing to actually perform a scroll operation. The second step involves defining the speed of such a scroll operation.

As it turns out, the developers behind jQuery already thought about both of these steps by providing us with the `animate` method. As its name implies, this method allows us to apply an animation to a given set of HTML elements. Furthermore, we can specify the duration of this animation (or use jQuery's default value). If you take a look at the jQuery documentation for `animate` (<http://api.jquery.com/animate/>), you will see that one of the possible parameters is `scrollTop`. Therefore, by writing `{scrollTop: target}` we can automatically scroll to `target` (`target` being the *target* location of the scroll).

Now, before we can apply our animation, we must ask ourselves on which element the animation should take effect. Well, the nesting of our HTML document takes the form of the following elements:

```
body
|__#welcome
|__#services
|__#gallery
|__#about
|__#contact
```

Furthermore, the scroll serves to navigate between elements. Therefore, it makes sense to apply the animation to the parent element of our HTML document:

```
$( 'body' ).animate({
    scrollTop: target
});
```

Great! So, we have told our browser to apply a scroll animation to the given target. But what exactly is the target? How do we define it? Well, our navbar items are anchor tags, and the anchor's `href` denotes the section to which an internal scroll should apply.

Therefore, we would need to access the individual navbar's `href` target and then use this as the target for our scroll animation. Luckily, this too is easy to do using jQuery. We first need to add an event listener to each navbar item. Then, for each clicked anchor, we extract the element to which its `href` attribute refers. We then determine this element's offset and pass this offset to our scroll animation.

To add an event listener to an element, we use jQuery's `on` method and pass `click` as a parameter to denote that we want to capture an on-click event:

```
$( "nav div ul li a" ).on('click', function(evt) {
```

Note how our selector only identifies anchor tags that are located within an unordered list inside a `div`. This ensures that only navigation menu items are being matched, as our markup should not contain any other anchor tags that are located inside list items belonging to a navigation element.

From within our event listener, we can access the object on which the click was performed using the `this` keyword. Thus, our clicked object will always be an anchor instance, and we can access the contents of its `href`. Specifically, we can access the string that follows the hash symbol within the `href`. To do this, all we have to write is `$(this).prop('hash')` or (better and more concise) `this.hash`

Remember that the string following the hash within a `href` identifies an internal element within the HTML document. We can therefore use the extracted string as a jQuery selector to get jQuery to retrieve the desired instance of the HTML element. All that we need to do then is use jQuery's `offset()` method to calculate our element's coordinates for us:

```
$("nav ul li a").on('click', function(evt) {
    var offset = $(this.hash).offset();
});
```

Voila! We are almost done! Let's piece it all together, wrap the code into a `script` tag, and place it into the head of our HTML document:

```
<script type="text/javascript">
    $(document).ready(function() {
        $("nav div ul li a").on('click', function(evt) {
            var offset = $(this.hash).offset();
            $('body').animate({
                scrollTop: offset.top
            });
        });
    });
</script>
```

Save the document and try it out in your browser. Our code executes correctly. But something still isn't quite right. Did you notice the odd flicker as you clicked on the navbar items? This is because the anchor tag on which we are clicking still tells the browser to jump to the specified internal element. At the same time, we also instruct our browser to animate a scroll to the element. To resolve this duplication, we have to prevent the on-click event from trickling down to the anchor tag once it reaches our event listener. To do this, we call `preventDefault` on the event:

```
$("nav div ul li a").on('click', function(evt) {
    evt.preventDefault();
    //...
});
```

Apply the changes, save, refresh, and try again. Great! Our custom scroll works! But there is one last annoyance. Clicking on the drop-down menu that launches our **Profile** and **Settings** modal dialogs. Did you notice how their anchor tags link to a plain hash symbol?

Let's deal with this corner case by checking whether jQuery's `offset()` method can successfully execute an offset. To do this, we must wrap our call to animate within an `if` statement, so that our final block of code is as follows:

```
<script type="text/javascript" >
$(document).ready(function() {
    $("nav div ul li a").on('click', function(evt) {
        evt.preventDefault();
        var offset = $(this.hash).offset();
        if (offset) {
            $('body').animate({
                scrollTop: offset.top
            })
        }
    });
});
</script>
```

Icons

The customization of our MyPhoto navbar is coming along nicely. We now have a nice scroll animation in place as well as a set of navbar items that update themselves based on the user's scroll position. However, we are not quite there yet. The items in our **Profile** drop-down menu still looks quite plain. Wouldn't it be nice if we could use icons to increase each drop-down menu item's readability? Adding icons to controls and menus helps draw attention to important functionality while clearly outlining a control's intended purpose.

When it comes to icons, a popular choice among web developers is the Font Awesome icon library (<https://fontawesome.github.io/Font-Awesome/>) which is a free collection of over 500 icons that were made to be used with Bootstrap websites. Take a look at the following screenshot:

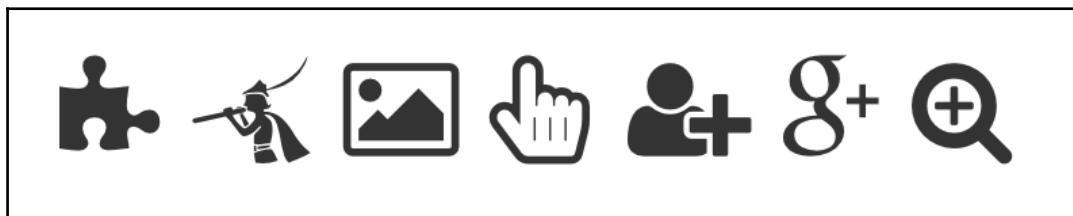


Figure 4.4: Examples of various Font Awesome icons.

To download the icon library, run a bower install as follows:

```
bower install components-font-awesome
```

Once the download completes, you will see that a new sub directory named `components-font-awesome` has been created inside your `bower_components` directory:

```
bower_components/
bootstrap/
components-font-awesome/
|__css/
|__fonts/
|__less/
|__scss/
```

Font Awesome icons ship as fonts. In order to be able to use the downloaded icons, all that you will need to do is include the Font Awesome style sheet that is located inside your `bower_components/components-font-awesome/css` directory. Insert the following link into the head of our HTML document:

```
<link rel="stylesheet" href="bower_components/components-font-awesome
/css/font-awesome.min.css" />
```

The complete head of our HTML document should now look as follows:

```
<head>
    <meta charset="UTF-8">
    <title>ch04</title>
    <link rel="stylesheet" href="bower_components/bootstrap/dist/css
/bootstrap.min.css" />
    <link rel="stylesheet" href="styles/myphoto.css" />
    <link rel="stylesheet" href="bower_components/components-font-
awesome/css/font-awesome.min.css" />
    <script src="bower_components/jquery/dist/jquery.min.js"></script>
    <script src="bower_components/bootstrap/dist/js/bootstrap.min.js">
        </script>
    <script type="text/javascript">
        // Smooth scroll JavaScript code here
    </script>
</head>
```

To use an icon, just apply the icon's class name to an HTML element. The class names for individual icons can be determined by looking at either the Font Awesome documentation, or by using GlyphSearch (<http://glyphsearch.com/>), a handy little search engine that lets you search for icons, preview them, and then copy their class name to use within your HTML document. It is important to note that each (Font Awesome) icon must be a child of a special class, The `fa` class. That is, to use an icon, you must first apply the `fa` class to the selected element, followed by the icon's name. For example, if your icon's class name is `fa-user`, then you would set your element's class attribute to `class="fa fa-user"`:

```
<span class="fa fa-user"></span>
```

Note that, while it is perfectly acceptable to apply a `fa` class to any HTML element, the convention for icons is to use the `<i>` element. That is:

```
<i class="fa fa-user"></i>
```

Having applied a `fa` class to our desired element, we can now style it just as we would any other element. For example, to change its color:

```
<i class="fa fa-user" style="color: red;"></i>
```

Take a look at the following screenshot:

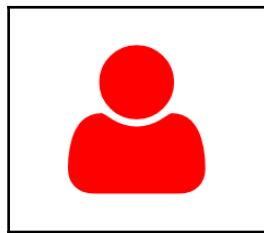


Figure 4.5: Changing the color of a Font Awesome icon.

Now that we know how to include icons in our page, let's go ahead and customize our **Profile** drop-down menu so that the individual drop-down menu items contain both text and a descriptive icon. Feel free to select your own icons, however, appropriate choices for each item would be:

- `fa-user`: This is for **Profile**
- `fa-cog`: This is for **Settings**
- `fa-sign-out`: This is for **Logout**

To add the individual icons to our drop-down menu, simply create a new `<i>` in front of the menu item's text and apply the appropriate class:

```
<ul class="nav navbar-nav navbar-right">
    <li class="dropdown">
        <a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button"
            aria-haspopup="true" aria-expanded="false">Profile <span
            class="caret"></span></a>
        <ul class="dropdown-menu">
            <li>
                <a href="#" data-toggle="modal" data-target="#profile-modal">
                    <i class="fa fa-user"></i> Profile
                </a>
            </li>
            <li>
                <a href="#" data-toggle="modal" data-target="#settings-modal">
                    <i class="fa fa-cogs"></i> Settings
                </a>
            </li>
            <li role="separator" class="divider"></li>
            <li>
                <a href="#">
                    <i class="fa fa-sign-out"></i> Logout
                </a>
            </li>
        </ul>
    </li>
</ul>
```

Take a look at the following screenshot:

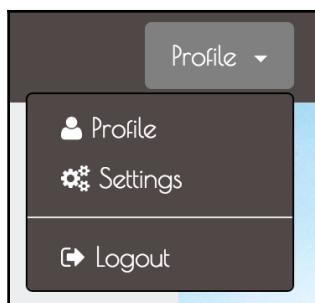


Figure 4.6: Our navbar's new drop-down menu sporting Font Awesome icons.

A note on icons in Bootstrap 3

Bootstrap 3 came with a large library of icons that range from symbols and objects to products. While the icons were provided by a third party (GLYPHICONS), they were free to use (for non-commercial purposes) and fully customizable. In this sense, the term customizable referred to the fact that they came embedded as fonts. This means that you could style them, just as you would with any other text. For example, you can change their color or size by applying CSS in the same way as you would to any other HTML element. Unfortunately, Bootstrap 4 no longer ships with icons.

Using and customizing alerts

Now that we know how to use icons, let's turn to a different topic, namely, alert boxes. Alert boxes are typically used to highlight an important event, or to emphasize an important message. As such, the purpose behind alerts is to provide a content area that immediately stands out, and therefore cannot be easily overseen by the user. For example, imagine that MyPhoto only supports browsers above certain versions. In such a case, a user who visits the site with an unsupported browser version should be notified that their browser is not supported. After all, the website may not function or display correctly when viewed with unsupported software. Bootstrap provides us with the `alert` class, which makes it very easy for us to implement this hypothetical scenario (the JavaScript for browser detection will be presented in [Chapter 5, Speeding Up Development Using jQuery Plugins](#)).

Bootstrap's alert comes with four contexts: **Success**, **Warning**, **Info**, and **Danger**. Each context class styles the element to which it is applied differently, depending on the intended message. See *Figure 4.7*, which lists the four default alert styles that come with Bootstrap:



Figure 4.7: Bootstrap's four contextual alert classes: .alert-success, .alert-warning, .alert-info, and .alert-danger.

Let's go ahead and apply one of these styles to our new unsupported browser alert box. Go ahead and create a new `div` and set its `class` attribute to `alert alert-danger`:

```
<div class="alert alert-danger">
    <strong>Unsupported browser</strong> Internet Explorer 8 and
    lower are not supported by this website.
</div>
```

Now insert this alert `div` inside our **Welcome** section, below the jumbotron:

```
<div class="container-fluid myphoto-section bg-myphoto-welcome"
id="welcome">
    <div class="container">
        <div class="jumbotron">
            <h1>Welcome to MyPhoto</h1>
            <p>Photographs you can cherish!</p>
        </div>
        <div class="alert alert-danger">
            <strong class="alert-heading">Unsupported browser</strong>
            Internet Explorer 8 and lower are not supported by this
            website.
        </div>
    </div>
</div>
```

Save and refresh. Voila! We have just created our very first Bootstrap alert (see *Figure 4.8*). Take a look at the following screenshot:

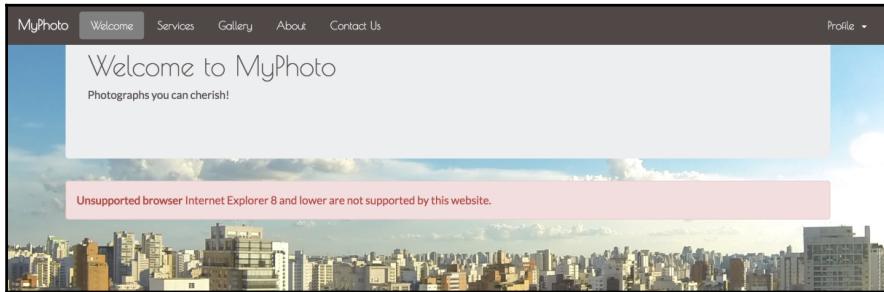


Figure 4.8: Our first dangerous alert dialog.

But something isn't quite right. What if the user knows that their browser is outdated, but still wishes to continue viewing the contents of our **Welcome** section without the invasive alert? Could we provide a way for the user to acknowledge the message and then allow them to continue browsing without its invasive presence? The answer is yes. Bootstrap provides us with a very easy way to make alerts dismissible using the `data-dismiss` attribute:

```
<div class="alert alert-danger alert-dismissible">
  <a href="#" class="close" data-dismiss="alert" aria-label="close">
    &times;</a>
  <strong>Unsupported browser</strong> Internet Explorer 8
  and lower are not supported by this website.
</div>
```

This will add an **X** to the right side of our alert dialog. Adding the `alert-dismissible` class will align the **X** and have it inherit its color. Click this **X**, and see the alert disappear.

This is great. Users can now dismiss our alert. However, what happens when the user jumps straight into a different section of MyPhoto? Currently, the alert is placed inside our **Welcome** section. As such, users viewing other sections of our website will not necessarily be able to see the alert dialog. The solution is to adjust the position of our alert dialog so that it appears stuck to our page, regardless of the section that the user is currently in. How do we do this? First, we will need to take the alert outside of our **Welcome** section, and move it just below our navbar. This will make the alert hidden behind our fixed navbar. To make the alert visible below our navbar, we can simply offset the position of the alert from the top of the page using the CSS `margin-top` property. To then make the alert sticky, that is, fixed below the navbar regardless of which section the user is currently in, we use the CSS `position` property and set it to `fixed`. Lastly, we can adjust the left offset and width of our alert so that it is nicely indented from the left hand side of our page, and stretches

horizontally across the page (note that, for the sake of keeping the sample code short and concise, we are applying inline styles to achieve this. However, as we will discover in Chapter 8, *Optimizing Your Website* we should normally avoid using inline styles whenever we can). Observe the following code:

```
<nav class="navbar navbar-myphoto navbar-fixed-top" role="navigation">
    <!-- Navbar markup -->
</nav>
<div class="alert alert-danger alert-dismissible" style="position: fixed; margin-top: 4em; width: 90%; margin-left: 4em;">
    <a href="#" class="close" data-dismiss="alert" aria-label="close">&times;</a>
    <strong class="alert-heading">Unsupported browser</strong>
    Internet Explorer 8 and lower are not supported by this website.
</div>
```

Take a look at the following screenshot:

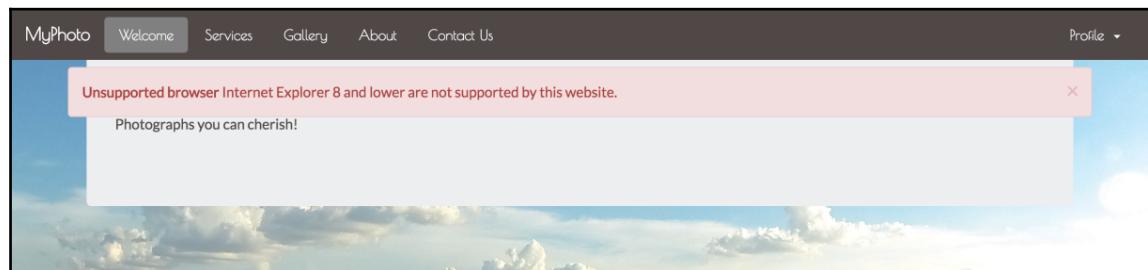


Figure 4.9: Our sticky alert dialog now stretches across the entire page and remains visible across the sections.

Great! Try scrolling down the page and observe how our alert remains fixed below the navbar. Let's add an icon:

```
<div class="alert alert-danger alert-dismissible" style="position: fixed; margin-top: 4em; width: 90%; margin-left: 4em;">
    <a href="#" class="close" data-dismiss="alert" aria-label="close">&times;</a>
    <strong class="alert-heading"><i class="fa fa-exclamation"></i> Unsupported browser</strong>Internet Explorer 8 and lower are not supported by this website.
</div>
```

Our alert is already looking pretty decent. It is positioned nicely below our navbar, is dismissible, and sports a nice little icon. But it somehow doesn't look very dangerous, does it? How about we customize its colors a bit? For example, we could darken the background color slightly, and lighten the foreground color. But how would we go about doing this without modifying the Bootstrap source? Easy! Just apply the desired CSS properties using either an inline style, or, even better, apply it globally throughout our style sheet. Go ahead and open `styles/myphoto.css`. Insert the following CSS snippet, save, and then refresh the MyPhoto page:

```
.alert-danger {  
    background-color: #a94342;  
    color: white;  
}
```

The snippet that you just added to your `myphoto.css` file should be pretty self-explanatory: It applies a white foreground and a dark red background (*Figure 4.10*) to any element that has the class `alert-danger`. Consequently, this foreground and background color will apply to any alert dialog that uses the `alert-danger` context class. Congratulations! You just learned how to customize your first Bootstrap component!

Let's finish by tidying any inline styles that we created (we will talk more about inline styles in Chapter 8, *Optimizing Your Website*; for now just accept that you should avoid using inline styles whenever possible). Create a custom class, `alert-myphoto`, extract the inline styles into it, and add a `z-index` rule to ensure that our warning will appear above all other elements on the page:

```
.alert-myphoto {  
    background-color: #a94342;  
    color: white;  
    position: fixed;  
    margin-top: 4em;  
    width: 90%;  
    margin-left: 4em;  
    z-index: 3000;  
}
```

Take a look at the following screenshot:

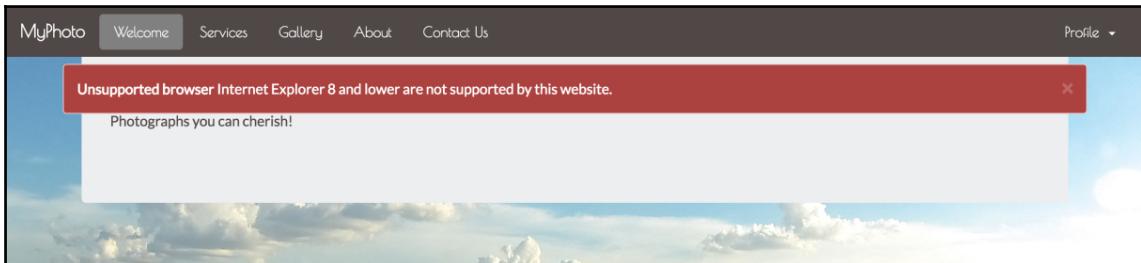


Figure 4.10: Our alert now with a white foreground and a darker background.

Under the hood



Bootstrap defines the background color for the individual context classes using the `$state-<context>-bg` variable, where context refers to one of the four Bootstrap contexts: **Success**, **Danger**, **Info**, or **Warning**. As such, you can globally override the default background color for a given alert by changing the value of `$state-<context>-bg` to your desired background color. For example, to make the background of the danger alert blue, you could write the following:

```
$state-danger-bg: blue
```

However, this is generally not recommended as it will change the background color for all danger alerts across your entire website.

Note that, when adding links to alerts, that you should apply the `alert-link` class to the link element. This will style the element to match the alert's context.

Creating a footer

At the moment, MyPhoto only contains a placeholder in place of a useful footer. Before MyPhoto goes live, it will desperately need a footer that should contain at least three pieces of information:

- A copyright notice
- A link to your website's terms and conditions
- A link to your website's **About** section

Let's go ahead and modify our footer to include these three pieces of information:

```
<footer class="footer">
    <p class="text-muted">&copy; MyPhoto Inc.</p>
    <p class="text-muted">Terms & Conditions</p>
    <p class="text-muted">About Us</p>
</footer>
```

Now open `styles/myphoto.css` and insert the following CSS:

```
footer p {
    display: inline;
}
```

So, what exactly did we just do? We inserted our copyright notice, and some placeholder text for our **Terms and Conditions** and **About Us** link. We embedded each of these three texts inside a paragraph element and applied Bootstrap's `text-muted` context class. The `text-muted` class does precisely what its name implies. It attempts to mute anything containing text by setting the foreground color to `#777` (a very soft, light grey). Take a look at the following screenshot:



© MyPhoto Inc. Terms & Conditions About Us

Figure 4.11: Our footer text: Copyright notice, Terms & Conditions, and About Us

Save and refresh (see *Figure 4.11*). Our footer is already looking better, however, unfortunately the text is somewhat obtrusive. Go ahead and make it smaller:

```
<footer class="footer">
    <p class="text-muted"><small>&copy; MyPhoto Inc.</small></p>
    <p class="text-muted"><small>Terms & Conditions</small></p>
    <p class="text-muted"><small>About Us</small></p>
</footer>
```

Now center the text by applying Bootstrap's `text-xs-center` context class (recall from Chapter 3, *Building the Layout* that the `text-xs-center` will center align text on viewports of size `xs` or wider):

```
<footer class="footer text-xs-center">
    <p class="text-muted"><small>&copy; MyPhoto Inc.</small></p>
    <p class="text-muted"><a href="#"><small>Terms & Conditions</small>
        </a></p>
    <p class="text-muted"><a href="#"><small>About Us</small></a></p>
</footer>
```

Last but not least, we first space the individual paragraphs out by adding a left margin of 10 pixels to each paragraph, and then adjust the foreground color of our footer's links:

```
footer p {  
    display: inline;  
    margin-left: 10px;  
}  
footer a {  
    color: inherit ;  
}
```

© MyPhoto Inc. Terms & Conditions About Us

Figure 4.12: Our completed footer

Under the hood

Bootstrap's text alignment classes are just wrappers for the CSS `text-align` property. That is, the `text-xs-center` class is defined by one line only: `text-align: center !important;`. Likewise, `text-xs-right` and `text-xs-left` are represented by `text-align: right !important` and `text-align: left !important`. Other sizes are defined through CSS media queries. For example, the `bootstrap.css` file defines the text alignment classes for small viewports so that they only apply to viewports that are at least 544 pixels wide:



```
@media (min-width: 544px) {  
    .text-sm-left {  
        text-align: left !important;  
    }  
    .text-sm-right {  
        text-align: right !important;  
    }  
    .text-sm-center {  
        text-align: center !important;  
    }  
}
```

Creating and customizing forms

At long last we are ready to move on to the final part of our landing page, the **Contact Us** form. Note that we will not be writing any of the actual JavaScript that transfers the contents of the **Contact Us** form to the server. Instead, we will learn how to use Bootstrap to lay out our form elements in an elegant and responsive manner.

Typically, **Contact Us** forms require at least three pieces of information from the user—the user's name (so that the recipient of the form data will know *who* they are talking to), the user's e-mail address (so that the recipient of the form data can *reply* to the user), and the actual message (what the user wants to send to the recipient). As such, we will be creating a form that will consist of three inputs—a name field, an e-mail address field, and a text area for the user to write their message.

Let's dig right into it. Start by creating an empty form below the **Contact Us** header:

```
<div class="container-fluid myphoto-section bg-myphoto-light"  
id="contact">  
    <div class="container">  
        <div class="row">  
            <h3>Contact Us</h3>  
            <form>  
                <!--Our form will go here-->  
            </form>  
        </div>  
    </div>  
</div>
```

At the heart of Bootstrap's form layout is the `form-group` class. On its own, all this class does is set a 15 px margin. However, as we will see, combining the form group with form controls gives us a powerful way of controlling our form's appearance.

By default, Bootstrap lays out its form elements vertically. That is, each element is stacked above the other. Let's go ahead and add a name field to our form. Insert the following snippet in between our previously created form tags:

```
<div class="form-group">  
    <label for="name">First and Lastname</label>  
    <input type="name" class="form-control" id="name"  
    placeholder="Name">  
</div>
```

Save and hit refresh. As you can see, the label takes over the entire first row, with the name input field appearing on the second row (*Figure 4.13*). Note how we applied the `form-control` class to our `name` input field. This class is what styles our actual input element.

Among other things, Bootstrap sets its height (to exactly 34 px), pads it, sets the input's font size (to 14 px), and gives the element a nice inset effect by adjusting the border style, border color, and border radius. Removing this class from our input will result in just a simple, plain input box being displayed.

Now, what if we wanted slightly smaller form elements? Well, we could apply our own styles, for example, we could change the font size or height of our input element. However, Bootstrap provides us with `form-group-sm` and `form-group-lg`. The former reduces the height of nested form controls to 30 px, the font size to 12 px, and the line height to 1.5 mm. The `form-group-lg`, on the other hand, makes its containing form controls larger by increasing their font size to 18 px and their height to 46 px. Go ahead and apply either one to our form group. For example:

```
<div class="form-group-sm">
    <label for="name">Name</label>
    <input type="name" class="form-control" id="name"
        placeholder="Name">
</div>
```

Take a look at the following screenshot:

A screenshot of a web page titled "Contact Us". Below the title, there is a label "Name" followed by a long, thin input field. The input field has a placeholder "Name" and is styled with a light gray background and a thin black border.

Figure 4.13: Our name input.

Great! We have created our first form. But note how (*Figure 4.13*) our name field stretches across the entire page, taking up a lot of valuable real estate. Seeing how users will certainly not need all this space for their names, you might wish to align the label and name input field. Luckily, Bootstrap makes this quite easy for us by offering horizontal form layouts and inline forms through the `form-inline` class. Applying it to our `form` element will change the display style of form controls and form groups contained within the form to `inline-block`. This will align the various components within your `form` element. Go ahead and apply it (see *Figure 4.14*):

```
<form class="form-inline">
    <div class="form-group-sm">
        <label for="name">Name</label>
        <input type="name" class="form-control" id="name"
            placeholder="Name">
    </div>
</form>
```

Take a look at the following screenshot:

The screenshot shows a 'Contact Us' form group. It contains two input fields, both labeled 'Name'. The first input field has the placeholder 'Name' and is currently empty. The second input field also has the placeholder 'Name' and is currently empty. The entire form group is enclosed in a black rectangular border.

Figure 4.14: Our vertical name form group.

This is much better. But we are not done quite yet. Try adding a second form group to the form. It will end up on the same line as our name form group. Furthermore, the input fields are now a bit too small to be of practical use. Try customizing them by combining the lessons that you learned in this chapter with the lessons you learned in Chapter 2, *Making a Style Statement*.

While we can save valuable space by aligning labels and input boxes, there is a second approach that we can take to combine input description with the actual input fields: Icons and placeholder texts (see *Figure 4.15*). The idea is that we combine icons and placeholder text to signify the expected input. The advantage behind this approach is that we will not be losing space to label elements.

First, let's remove the previously inserted label element and edit the input's placeholder attribute to contain the string Your name. Our name form group should now consist of a single input element:

```
<div class="form-group">
    <input type="text" class="form-control" placeholder="Your name">
</div>
```

Let's go ahead and duplicate this form group so that we now also have an input for our user's e-mail address:

```
<form>
    <div class="form-group">
        <input type="text" class="form-control" placeholder="Your
name">
    </div>
    <div class="form-group">
        <input type="text" class="form-control" placeholder="Your email
address">
    </div>
</form>
```

Now let's go ahead and select two Font Awesome icons that best describe the expected input. We recommend:

- A user symbol to indicate the user's name: `fa fa-user`
- An @ symbol to indicate the user's e-mail address: `fa fa-at`

Take a look at the following screenshot:

The screenshot shows a contact form with three fields. The first field is labeled 'Your name' with a small user icon to its left. The second field is labeled 'Your email address' with a small '@' icon to its left. The third field is labeled 'Your message' and contains a large text area for input.

Figure 4.15: The finished Contact Us form: labels and placeholder text are combined to describe the expected input for a given input field.

To combine the input element with the icon, we first apply Bootstrap's `input-group` class to the form control. That is, our form control's `class` attribute should now be set to `class="form-group input-group"`. Next, we create the `<i>` tag for our Font Awesome icon, insert it before our input element, and then wrap the icon with a `span` element. We set the `span` class attribute to `class="input-group-addon"`:

```
<span class="input-group-addon"><i class="fa fa-user"></i></span>
```

Under the hood



The `input-group` class modifies the display, position, and `z-index` properties of a `form-control` element. In combination with an `input-group-addon`, this allows you to insert elements to the left or right of input elements.

Apply this to both the name and e-mail inputs and save. You should now see that the icons are aligned to the left of our input elements. To align them to the right, simply move the icon span *after* the input element:

```
<div class="form-group input-group">
    <input type="text" class="form-control" placeholder="Your email
address">
    <span class="input-group-addon"><i class="fa fa-at"></i></span>
</div>
```

Our form is almost done. All that is left is to add a text area for our message (using the `textarea` element) and a **Send** button. To achieve the former, go ahead and create a new `form-group`. However, instead of the `form-group` containing an `input` box, add a `label` and a `textarea`. Just as with the `input` elements, the `textarea` should be a form control (so go ahead and set its `class` attribute to `form-control`):

```
<div class="form-group">
    <label for="name">Your message</label>
    <textarea class="form-control" rows="5" id="message"></textarea>
</div>
```

The last missing part of the puzzle is our **Send** button. Although we won't be writing the event listeners for our **Send** button, we will now explore the various Bootstrap context styles available for buttons. The parent class of any button is the `btn` class. In essence, the `btn` class adjusts the padding, margin, text alignment, font size, and weight as well as the border radius of any element that it is applied to. The seven context classes offered by Bootstrap are `btn-primary`, `btn-secondary`, `btn-success`, `btn-link`, `btn-info`, `btn-danger`, and `btn-warning` (see *Figure 4.16*). Take a look at the following screenshot:



Figure 4.16: The five Bootstrap button context styles. From left to right: Default, success, danger, info, and warning.

Let's go ahead and complete our form by inserting a `button` element and apply the `success` context class. Create a new `form group` below our text area. Inside the `form group`, create a `button` element and set its `class` to `btn btn-success`. Insert a nice icon and add some text to the button:

```
<div class="container-fluid myphoto-section bg-myphoto-light"
id="contact">
    <div class="container">
        <div class="row">
            <h3>Contact Us</h3>
            <form>
```

```
<div class="form-group input-group">
    <span class="input-group-addon"><i class="fa fa-user">
    </i></span>
    <input type="text" class="form-control"
        placeholder="Your name">
</div>
<div class="form-group input-group">
    <span class="input-group-addon"><i class="fa fa-at">
    </i></span>
    <input type="text" class="form-control"
        placeholder="Your email address">
</div>
<div class="form-group">
    <label for="name">Your message</label>
    <textarea class="form-control" rows="5"
        id="message">
    </textarea>
</div>
<div class="form-group">
    <button class="btn btn-success">
        <i class="fa fa-send-o"></i>
        Send
    </button>
</div>
</form>
</div>
</div>
```

Last but not least, we can add some descriptive text to our **Contact Us** section. In this case, we will use placeholder text generated using <http://generator.lorem-ipsum.info/> (a useful tool if you ever require placeholder text for demo purposes):

```
<h3>Contact Us</h3>
<p>
    Lorem ipsum dolor sit amet, modo integre ad est, omittam temporibus
    ex sit,
    dicam molestie eum ne. His ad nonumy mentitum offendit, ea tempor
    timeam nec,
    doming latine liberavisse his ne. An vix movet dolor. Ut pri
    qualisque reprehendunt,
    altera insolens torquatos in per. Mei veri omnium omittam at,
    ea qui discere ceteros.
</p>
```

Take a look at the following screenshot:

The screenshot shows a "Contact Us" form within a bordered container. At the top, there is some Latin placeholder text. Below it, there are two input fields: one for "Your name" with a person icon and another for "Your email address" with an '@' icon. A large text area labeled "Your message" follows. At the bottom is a green "Send" button with a paper airplane icon.

Figure 4.17: The Contact Us section along with the Send button.

While in the case of the preceding example the inputs are pretty explanatory, there may be inputs that require an explanation. As such, you can add descriptive text using the `form-text` class:

```
<p class="form-text">Some text describing my input.</p>
```

All that the `form-text` class does is set the element's display to block, and add a top margin of 0.25 rem to the element.

Furthermore, it should be noted that we wrapped both the **Contact Us** and **Send us a message** text inside `h3` and `h4` header tags. This is appropriate in this case, as the titles blend nicely with the content. However, for cases in which the title should stand out more, or should not blend with the content, one should use Bootstrap's display heading class `display-*`, where * denotes a number between 1 and 4. As with the header tags, the smaller the number, the bigger the font. The largest display class applies a font size of 6 rem to the target element, and decreases this by 0.5 rem for each successive display class (so, for example, `display-2` sets the font size to 5.5 rem). The display style definition also sets the font weight to 300. The *Figure 4.18* contrasts the use of `display-*` against the use of the HTML header tags. Take a look at the following screenshot:



Figure 4.18: Contrasting Bootstrap's display classes against the HTML header tags.

Increasing the font size and weight of paragraphs



In order to emphasize the text contained within paragraphs (as opposed to their headers), the `lead` class should be used. This class changes the font size to 1.25 rem and the font weight to 300.

Form validation

Although this book does not cover the server-side logic required to make this happen, we will need to implement one last essential stepping stone, client-side form validation. In other words, what is currently missing from our form, aside from the server-side code to send or store the actual message, is a way for us to validate the user's form input and to notify the user should they fill out the form incorrectly. One obvious solution is to identify what type of input the form fields require, and then to use either plain JavaScript or jQuery to check the contents of each field using regular expressions. Once an expression fails to match, we could then set the contents of a specific HTML element to contain an error message. However, do we really need to implement all this from scratch? The short answer is *no*. Form validation is a well explored area within web development. Consequently, there exist plenty of third-party JavaScript libraries that allow us to implement form validation quickly and with relatively little effort. As such, we will not be concerned with implementing the client-side validation logic as part of this book. Instead, a more noteworthy topic is Bootstrap's new validation styles. Unlike its predecessor, Bootstrap 4 comes with form validation styles, which greatly simplifies form development. Specifically, Bootstrap provides the `has-success`, `has-warning`, and `has-danger` validation classes, which are to be added to the parent element of `input` in order to indicate a specific context (Figure 4.19):

```
<div class="form-group input-group has-danger">
    <span class="input-group-addon"><i class="fa fa-at"></i></span>
    <input type="text" class="form-control" placeholder="Your email
address">
</div>
```

Take a look at the following screenshot:

The form consists of a single column. At the top is a text input field with a placeholder 'Send us a message' and a user icon. Below it is another text input field with a placeholder 'Your name'. A third text input field follows, with a placeholder 'Your email address' and a red border indicating an error or warning state. At the bottom is a large text area labeled 'Your message'. At the very bottom is a green button labeled 'Send' with a paper airplane icon.

Figure 4.19: The has-danger context class applied to the form-group element containing our e-mail address input.

Adding the `form-control-*` to the input will cause a contextual icon to appear to the right-hand side of the input. The * denotes one of the three contexts—danger, success, or warning. So adding both `has-danger` to the `form-group` element and `form-control-danger` to the `input` element will cause the input to be highlighted using the danger context. A small icon, indicating failure, will appear to the right of the input:

```
<div class="form-group input-group has-danger">
  <span class="input-group-addon"><i class="fa fa-at"></i></span>
  <input type="text" class="form-control form-control-danger"
    placeholder="Your email address">
</div>
```

Progress indicators

Although unfitting of the context in which we are developing MyPhoto, progress indicators form an important part of many user interfaces. As such, it is worth pointing out that Bootstrap comes with some very nice styles for the `progress` element present in HTML5. Till date, the following classes are available:

- `progress`: This is for applying a default progress bar style.
- `progress-`: This is for applying context styles. Specifically, `progress-success`, `progress-info`, `progress-warning`, and `progress-danger`.
- `progress-striped`: This is for adding stripes to the progress bar, and `progress-animated` for animating the added stripes (note that currently animations are not supported by all browsers).

Since the `progress` element is not supported by Internet Explorer 9, Bootstrap also supplies

the progress-bar class, which allows an element to be turned into a progress bar. The progress-bar requires the parent element to have the progress class applied to it.

Adding content using media objects

As you add more contents to your website or web application, you will notice that a large proportion of this content revolves around the text aligned next to an image. Indeed, text and image alignment form the basis from which most modern websites are built, and combining these two elements into a reusable component results in what are called **media objects** (Figure 4.20). Take a look at the following screenshot:

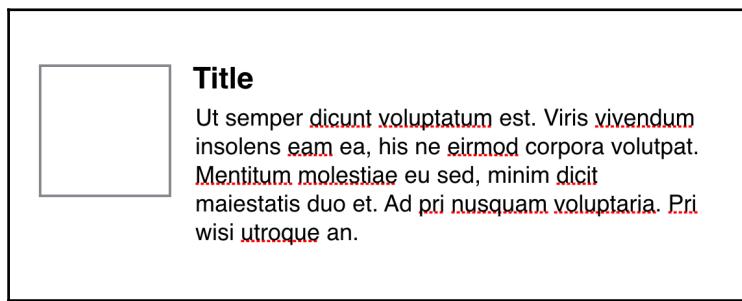


Figure 4.20: A media object refers to the combination of a title, text, and image element in such a way that they form a reusable entity.

Given how fundamental these media objects are to a website's content, it is of no surprise that Bootstrap 4 offers out-of-the-box support for them. Let's see how we can make use of Bootstrap's media-object to improve the appearance of our **About Us** section. One suggestion for improvement could be to add some profile information about one of the photographers at MyPhoto. To do so, go ahead and create a new div element, and assign it the media class. This identifies the element as a media object and gives the element a top margin of 15 px. To add a left aligned image of one of the MyPhoto photographers, create a new div element, set its class to media-left and create a nested image element to which the class media-object is applied. This ensures that the image is aligned to the left of our content, with appropriate padding (specifically, a padding-right of 10 px and a top vertical alignment). To add some biographical information about our photographer, we must use the media-body class. Create a second div inside the media element, assign it the media-body class, and add some text. Use a header element in conjunction with the media-heading class to create a title:

```
<div class="media">
  <div class="media-left">
    
```

```
</div>
<div class="media-body">
    <h4 class="media-heading">Jason</h4>
    Some text about Jason, our photographer. Aeterno meliore
    has ut, sed ad.
    Tollit volumnus mea id, sed dicunt aliquando cu. Ea reque
    dmedsimilique deseruisse duo
    Est te essent argumentum, mea et error tritani eleifend.
    Eum appellantur intellegebat at, ne graece repudiandae
    vituperatoribus duo.
</div>
</div>
```

Take a look at the following screenshot:

The screenshot shows the 'About' section of a website. At the top, there is a dark header bar with the word 'About'. Below it is a white content area. On the left side of this area, there is a small thumbnail image of a man and a woman, followed by the name 'Jason'. To the right of the thumbnail, there is a block of descriptive text. Above the thumbnail, there is a heading and some introductory text. Below the thumbnail, there is more detailed information about the photographer.

```
The style of photography will be customised to your personal preference, as if being shot from your own eyes. You will be part of every step of the photography process, to ensure these photos feel like your own.

Our excellent photographers have many years of experience, capturing moments from weddings to sporting events of absolute quality.

MyPhoto also provides superb printing options of all shapes and sizes, on any canvas, in any form. From large canvas prints to personalised photo albums, MyPhoto provides it all.

MyPhoto provides a full, holistic solution for all your photography needs.

 Jason
Some text about Jason, our photographer. Aeterno meliore has ut, sed ad nibh sadipscing accommodare. Tollit volumnus mea id, sed dicunt aliquando cu. Ea reque similique deseruisse duo. Est te essent argumentum, mea et error tritani eleifend. Eum appellantur intellegebat at, ne graece repudiandae vituperatoribus duo.
```

Figure 4.21: Using Bootstrap's media objects to add a photographer profile to MyPhoto's About Us section.

Instead of top aligning our `media` element, we could also:

- Middle align the `media` element using `media-left media-middle`.
- Bottom align the `media` element using `media-left media-bottom`.



Note that, currently, Bootstrap 4 only supports left alignment (right alignment is not supported).

Of course, media elements do not need to appear on their own. Instead, they can be both nested and/or combined into lists. To nest a media element, simply place the child element inside the parent's body element (recall from our example above that body elements are denoted using the `media-heading` class).

Lists of media elements are created using the `media-list` class. Simply apply it to a list element (`ul` or `ol` tag), and use the `media` class in conjunction with the individual list items (`li` tags).

Figures

If requiring just a figure, and no media context, then Bootstrap's figure classes should be used. Although they do not fit into the current context of MyPhoto, Bootstrap's figure styling is a commonly used and important feature. As such, it is worth explaining the three classes that are to be used when creating a figure. The `figure` class sets the element's display to `inline-block`. This forces the element to behave just like inline elements, but also allows it to have a set width and height. The `figure-img` class should be applied to `img` elements within the `figure` element, adjusting their bottom margin and line height. Lastly, the `figure-caption` class is used to denote captions, and adjusts the font size (setting it to 90%) and the font color (setting it to #818a91). Observe the following code:

```
<figure class="figure">
    
    <figcaption class="figure-caption">Sample text.</figcaption>
</figure>
```

Quotes

There will be occasions in which you may want to display a famous quote or citation on your site. While an appropriate style for this would not be too difficult or time consuming to implement on your own, block quotes are a common enough scenario for the Bootstrap developers to have decided to take this off your hands. Bootstrap 4, therefore, offers the `blockquote` for displaying quotes. The style rule for this class is not very complex. It merely adjusts the font size, bottom margin, and padding of the element to which it applies. It also adds a grey left-hand border to emphasize the element's contents. Let's go ahead and apply this class to an important motivational quote by one of the founders of MyPhoto that underpins the very foundations of the company (see *Figure 4.22*):

```
<blockquote class="blockquote">
    <p>I am very motivated today.</p>
```

```
<footer class="blockquote-footer">The Founder,  
    <cite>Times Magazine</cite>  
  </footer>  
</blockquote>
```

Note how we not only display the quote itself, but also provide a source using the optional `blockquote-footer` class.

Take a look at the following screenshot:

The screenshot shows a section titled "About" with a dark header bar. Below it is a white content area containing several paragraphs of text. On the left side of the content area, there is a vertical grey border. At the top of this border, the text "I am very motivated today." is displayed. Below this, a small horizontal line separates the quote from the source citation: "— The Founder, *Times Magazine*". To the left of the source citation is a small thumbnail image of a man, identified as Jason. Next to the thumbnail is the name "Jason". Below the name is a block of Latin text: "Some text about Jason, our photographer. Aeterno meliore has ut, sed ad nibh sadipscing accommodare. Tollit volumnus mea id, sed dicunt aliquando cu. Ea reque similique deseruisse duo. Est te essent argumentum, mea et error tritani eleifend. Eum appellantur intellegebat at, ne graece repudiandae vituperatoribus duo ABBR."

Figure 4.22: Using Bootstrap block quotes to cite the MyPhoto founder.

Instead of a left-hand border and left alignment, we can apply a right-hand border and align the source to the right using the reverse block quote class, `blockquote-reverse`:

```
<blockquote class="blockquote-blockquote-reverse">  
  <p>I am very motivated today.</p>  
  <footer class="blockquote-footer">The Founder,  
    <cite>Times Magazine</cite>  
  </footer>  
</blockquote>
```

All that `blockquote-reverse` does is set the CSS `text-align` rule to right and adjust the padding and border properties accordingly.

Abbreviations

Just as with Bootstrap 3, Bootstrap 4 styles HTML's built-in `abbr` tag, which gives developers the ability to denote abbreviations. Denoting a piece of text as being an abbreviation will result in a small, dotted line being drawn underneath the text. As the user hovers over the text, the mouse pointer will change into a question mark. By setting the `title` attribute, a tooltip will appear with the full `title` attribute's text content:

```
<abbr>ABBR</abbr>
```

And, just as with Bootstrap 3, the `initialise` class can be used in conjunction with the `abbr` tag in order to reduce the font size.

Summary

In this chapter we continued fleshing out the landing page of our demo project. We learned how to improve our website's navbar, and how to automate and customize our scroll speed. We used Bootstrap's Scrollspy plugin, and worked with and customized icons. This was followed by a discussion and practical examples of how to use and style custom Bootstrap alerts. Last but not least, we improved the footer for `MyPhoto`, learned how to create forms, discovered Bootstrap's media objects, and used Bootstrap's various content styles.

In the next chapter, *Chapter 5, Speeding Up Development Using jQuery Plugins*, we will bring this chapter's various new components to life. By learning how to apply third-party jQuery plugins, we will make `MyPhoto` interactive, and tie this chapter's various additions together to provide our site's visitors with a rich user experience.

5

Speeding Up Development Using jQuery Plugins

The previous chapter showed us how to style the content for MyPhoto and how to improve a website's overall appearance by using and customizing different Bootstrap components. We learned how to use Bootstrap's navbar, how to use icons, and how to customize a website's scrolling behavior using the Scrollspy plugin. In this chapter, we will emphasize the power of third-party plugins, introducing you to some essential third-party (and hence non-Bootstrap) plugins that will help speed up the development of the most common and mundane features. Building on the features implemented throughout the previous chapters, we will first teach you how to quickly and efficiently implement client-side browser detection using the jQuery browser plugin (`jquery.browser`). We will then improve the display of our tabular **Events** section by using pagination, first covering Bootstrap's pagination, and then showing you how to rapidly improve the default pagination feature using **bootpag**, Bootstrap's pagination plugin. Further improvements to our **Events** section will be made by adding images using Bootstrap Lightbox. Finally, staying within our tabular data display, we will improve the display of MyPhoto's price list using jQuery DataTables.

To summarize, this chapter will cover the following:

- Browser detection using the jQuery browser plugin
- Improved pagination using bootpag
- Using Bootstrap Lightbox to display images
- Enhancing the display of tabular data using DataTables

Browser detection

Recall the hypothetical example from [Chapter 4, On Navigation, Footers, Alerts, and Content](#), where MyPhoto only supports browsers above certain versions. To this end, we added a Bootstrap alert to our page, which notified visitors that their browser is not supported. Up until now, however, we had no way to actually identify which browser or browser version a MyPhoto visitor was using. Lacking any logic to hide and display the alert, the alert was visible regardless of whether or not the user's browser was actually supported by our website. Now the time has come for us to implement this missing logic.

Web browsers identify themselves by specifying their name and version information using a special field called **User-Agent**, which is part of the **HTTP Request Header** (see [Figure 5.1](#)). JavaScript allows users to access this field using the `window.navigator` property. This property contains the exact same string that is present in the **User-Agent** field of the **HTTP Request Header**. Therefore, to determine whether our visitor's browser is indeed supported, all that one needs to do is match the supported browser against the string presented by `window.navigator`. However, as can be seen from [Figure 5.1](#), these strings are typically quite long and complex. As a result, matching different browsers can be tedious and prone to programming errors. Therefore, it is much better to use external resources that do this matching for us, and have been well tested, well documented, and are kept up to date. The popular, open source jQuery browser plugin (<https://github.com/gabceb/jquery-browser-plugin>) is one such resource that we can use. Let's go ahead and do just that! As usual, go ahead and fire up your console and install the plugin using Bower:

```
bower install jquery.browser
```

Once the installation is complete, you should see a new directory under `bower_components`:

```
bower_components/jquery.browser
```

Inside the `dist` folder, you should see two files, namely, `jquery.browser.js` and `jquery.browser.min.js`.

Take a look at the following screenshot:



Figure 5.1: An example of a User-Agent string transmitted as part of the HTTP header when making a request.

If you can see the aforementioned files, then this means that `jquery.browser` has been successfully installed. Now, before we start using `jquery.browser`, we first need to work some more on our alert. The first thing that we will need to do is find a way of uniquely identifying it. Let's use the HTML `id` attribute for this purpose. Add the `id` attribute to our alert: `id="unsupported-browser-alert"`.

Next, we should tidy up our markup. Go ahead and open `css/myphoto.css` and move the inline styles for our alert into our style sheet.



As previously stated, we must eliminate inline styles at all costs. While we may, at times, apply inline styles to keep sample code snippets short, you should always try to avoid doing so outside the classroom.

This won't affect browser detection, but will simply allow us to keep our markup nice and tidy. Observe the following code:

```
#unsupported-browser-alert {  
    position: fixed;  
    margin-top: 4em;  
    width: 90%;  
    margin-left: 4em;  
}
```



Generally, it is not a good idea to couple CSS rules with an `id`, as this is bad for reusability. But hold tight; we will talk more about this in Chapter 6, *Customizing Your Plugins*.

Since we only want the alert to display under certain conditions (namely, if the user is using a specific version of Internet Explorer), we should hide the `alert` div by default. Go ahead and add the following CSS to our alert styles:

```
display: none;
```

Save and refresh. The alert at the top of the page should now no longer be visible. Furthermore, our outermost `alert` div should now only contain `id` and `class` attributes:

```
<div class="alert alert-danger" id="unsupported-browser-alert">
    <a href="#" class="close" data-dismiss="alert" aria-label="close">&times;</a>
    <strong class="alert-heading"><i class="fa fa-exclamation"></i>
        Unsupported browser</strong> Internet Explorer 8 and lower are
        not supported by this website.
</div>
```

Now it is finally time to start using our freshly installed jQuery plugin. Open the `MyPhoto` `index.html` file and include the minified `jquery.browser` JavaScript file within the head of the document:

```
<script
src="bower_components/jquery.browser/dist/jquery.browser.min.js">

</script>
Great. The head of our index.html should now look as follows:
<head>
    <meta charset="UTF-8">
    <title>ch05</title>
    <link rel="stylesheet" href="bower_components/bootstrap/dist/css
/bootstrap.min.css" />
    <link rel="stylesheet" href="styles/myphoto.css" />
    <link rel="stylesheet" href="bower_components/components-
font-awesome/css/font-awesome.min.css" />
    <script src="bower_components/jquery/dist/jquery.min.js"></script>
    <script src="bower_components/bootstrap/dist/js/bootstrap.min.js">
    </script>
    <script src=
    "bower_components/jquery.browser/dist/jquery.browser.min.js">
    </script>
    <script type="text/javascript">
        $( document ).ready(function() {
            $("nav ul li a").on('click', function(evt) {
                evt.preventDefault();
                var offset = $(this.hash).offset();
                if (offset) {
                    $('body').animate({
```

```
        scrollTop: offset.top
    });
}
});
});
</script>
</head>
```

We are now ready to use `jquery.browser` to detect which browser the visitor is using. To this end, and as noted in the `jquery.browser` documentation, the following variables are available:

- `$.browser.msie`: This is true if the website visitor is using Microsoft Internet Explorer.
- `$.browser.webkit`: This is true if the website visitor is using either Chrome, Safari, or Opera.
- `$.browser.version`: This shows the browser version (not type!) that the website visitor is using.

To add the logic that makes the alert visible, let us start with a general condition of whether or not the user is using Internet Explorer. We will move to a more specific condition later (whether the user is using a specific version of Internet Explorer). In other words, let us begin by making the browser alert visible only if the visitor's browser is identifying itself as Internet Explorer. The code for this is fairly straightforward. We simply check the `$.browser.msie` variable. If this variable evaluates to true, then we use jQuery to make our alert visible:

```
if ($.browser.msie) {
    $('#unsupported-browser-alert').show();
}
```

Now let us make our browser test more specific. We now want to test whether the visitor is using both Internet Explorer and whether the version of Internet Explorer (let's say version 8 and below) is unsupported by MyPhoto. To do so, we simply perform a second check using the `$.browser.version` variable. If the conditional evaluates to true, then the `show()` function is executed. The `show()` function modifies the display rule of an element to make it visible:

```
if ($.browser.msie && $.browser.version <= 8) {
    $('#unsupported-browser-alert').show();
}
```

Go ahead and insert this snippet into the head of our HTML document:

```
<script type="text/javascript">
    $( document ).ready(function() {
        $("nav ul li a").on('click', function(evt) {
            evt.preventDefault();
            var offset = $(this.hash).offset();
            if (offset) {
                $('body').animate({
                    scrollTop: offset.top
                });
            }
        });
        if ($.browser.msie && $.browser.version <= 7) {
            $('#unsupported-browser-alert').show();
        }
    });
</script>
```

Conditional comments with Internet Explorer



If you wished to target Internet Explorer only for specific portions of your markup, then you can use Microsoft's conditional comments with Internet Explorer. Other browsers that are not Internet Explorer will simply ignore these proprietary comments:

```
<!--[if IE 8]-->
    <insert IE specific markup here>
[endif]-->
```

Enhanced pagination using bootpag

In this section, we will learn both how to use Bootstrap's default pagination, and how to overcome its limitations quickly and with minimal effort. We will first populate the section with a set of sample events, and then group these events into pages in an effort to reduce the overall length of the section. In order to add a set of events to the **Events** section, replace the `<p>Lorem Ipsum</p>` markup in the `services-events` div with the following event placeholder text:

```
<h3>My Sample Event #1</h3>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Curabitur leo dolor,
fringilla vel lacus at, auctor finibus ipsum. Lorem ipsum dolor sit
amet,
```

```
consectetur adipiscing elit. Morbi quis arcu lorem. Vivamus elementum  
convallis  
enim sagittis tincidunt. Nunc feugiat mollis risus non dictum.  
Nam commodo nec  
sapien a vestibulum. Duis et tellus cursus, laoreet ante non,  
mollis sem.  
Nullam vulputate justo nisi, sit amet bibendum ligula varius id.</p>
```

Repeat this text three times so that we now have three sample events displaying on our page under the **Events** tab (see *Figure 5.2*). Add top padding and a left-hand margin of 2rem to the parent container to offset the events in an effort to make the section look less crowded:

```
#services-events .container {  
    margin-left: 2rem;  
    padding-top: 1rem;  
}
```

Take a look at the following screenshot:

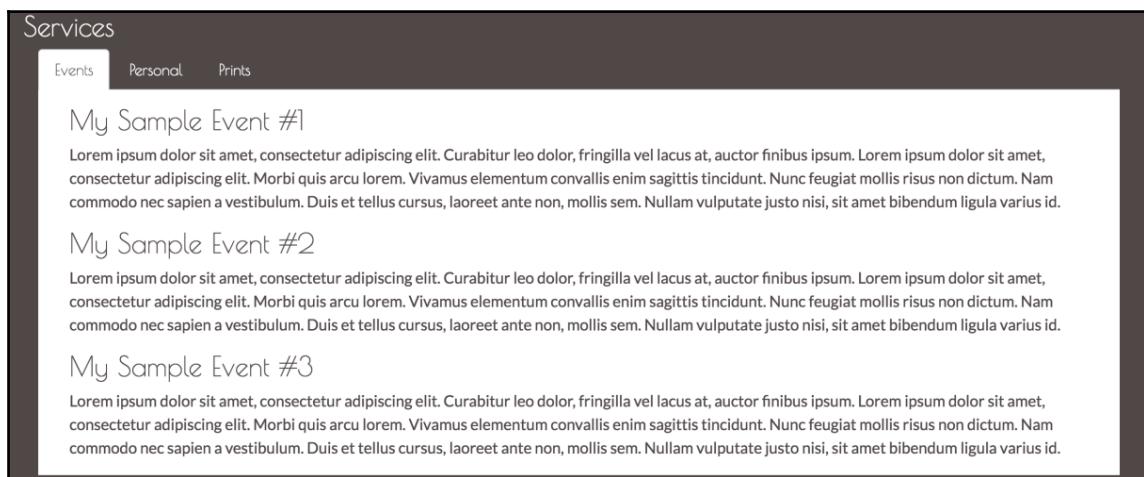


Figure 5.2: Three sample events displayed one below the other within our Events tab.

Hit save and refresh. Voila! This looks pretty good already, so why exactly would we want to display the events on separate pages? Well, as we begin adding more and more events, the events will appear below one another. As such, the **Events** section will grow indefinitely. Pagination is a clever way of avoiding this while allowing us to maintain the ability to list all MyPhoto events. Bootstrap offers a visually appealing pagination style (see *Figure 5.3*) that can be added to any section of your page by applying the `pagination` class to an unordered list element. The individual list items within this unordered list should

have the `page-item` class applied to them. Applying this class simply sets the element's `display` property to `inline`. Applying the `pagination` class sets the `display` of the unordered list to `inline-block` and adjusts its margins. As such, in order to display pagination with 10 pages (for example's sake, we will carry on using 10 pages from now on), add the following markup after the `p` element of our third event (note how the `active` class is used on a list item to denote the currently selected page. The `pagination-lg` and `pagination-sm` classes can be used to increase or decrease the size of the pagination control):

```
<ul class="pagination">
  <li class="page-item"><a class="page-link active" href="#">1</a></li>
  <li class="page-item"><a class="page-link" href="#">2</a></li>
  <li class="page-item"><a class="page-link" href="#">3</a></li>
  <li class="page-item"><a class="page-link" href="#">4</a></li>
  <li class="page-item"><a class="page-link" href="#">5</a></li>
  <li class="page-item"><a class="page-link" href="#">6</a></li>
  <li class="page-item"><a class="page-link" href="#">7</a></li>
  <li class="page-item"><a class="page-link" href="#">8</a></li>
  <li class="page-item"><a class="page-link" href="#">9</a></li>
  <li class="page-item"><a class="page-link" href="#">10</a></li>
</ul>
```

Pagination in Bootstrap 3

When it comes to pagination, the changes from Bootstrap 3 to Bootstrap 4 are not that drastic. The `pagination` class remains between the two versions. However, in Bootstrap 3, we did not explicitly need to specify which elements were pagination items and which elements were pagination links. As the `page-item` and `page-link` classes have only been introduced with Bootstrap 4, one could previously specify the pagination by simply creating an unordered list and applying the `pagination` class to it:



```
<ul class="pagination">
  <li><a class="active" href="#">1</a></li>
  <li><a href="#">2</a></li>
  <li><a href="#">3</a></li>
</ul>
```

With the addition of the preceding markup, we have already reached the end of Bootstrap's default pagination capabilities. The implementation of the actual pagination is up to us. Specifically, this would involve the following:

- Grouping our events into pages.
- Detecting the currently active page.
- Toggling the visibility of the various pages depending on the page that is currently selected.

As our event grows beyond 10 pages, we would then be required to manually add both a new page and a new list item to the paginator. While implementing the logic for all this is not quite rocket science, it would be nice if we did not have to be concerned with reinventing a solution to such a well known user interface problem. Indeed, there exist plenty of third-party libraries to help us speed up the development of our events pagination.

One of the most popular libraries is `jQuery.bootpag`, a jQuery plugin that allows you to paginate your data. Unfortunately, `bootpag` (versions 1.0.7 and below) currently does not support Bootstrap 4 out of the box, and as such will require a little bit of tweaking. As with all libraries presented in this chapter, `jQuery.bootpag` is free to use, and its source code, as well as licensing information, is available on GitHub at <https://github.com/botmonster/jquery-bootpag>. Take a look at the following screenshot:

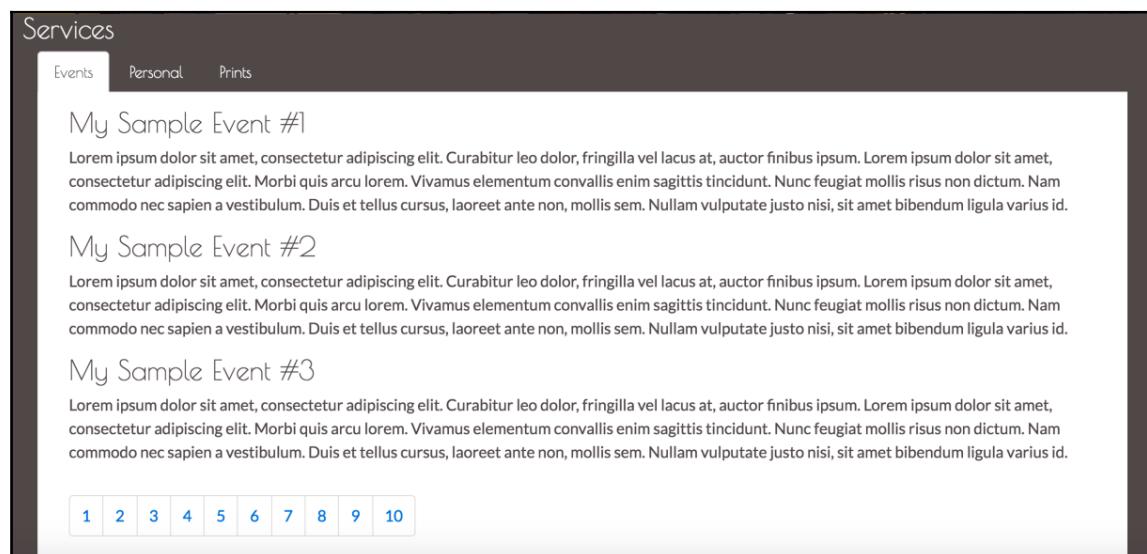


Figure 5.3: Bootstrap default pagination.

Unsurprisingly, the `bootpag` Bower package's name is also `bootpag`. Go ahead and install it:

```
bower install bootpag
```

Once installation is complete, you should see a directory named `bootpag` under your `bower_components` directory. Inside `bootpag/lib`, you should see the following files:

- `jquery.bootpag.js`
- `jquery.bootpag.min.js`

As always, we want to work with the minified version of our plugin, so go ahead and include `jquery.bootpag.min.js` within the head of our document:

```
<script  
src="bower_components/bootpag/lib/jquery.bootpag.min.js"></script>
```

Before we can start using `bootpag`, we must understand that the plugin needs containers: one container in which to display the pagination control, and one container for displaying the content that is to be paginated. In other words, it requires one to divide the area of the **Events** section between the data that is to be displayed, and one to separate the controls with which the user navigates the data. As a user navigates the data using the pagination control, the content area will be updated with the new content, or, alternatively, the visibility of multiple containers will be toggled.

We will be using the latter approach. That is, we will first divide our events into pages, and then use an event listener on the pagination control to toggle the visibility of these various pages. To this end, we must now go ahead and modify our events in the **Services** section so that each of our events is contained within its own distinct page (that is, by using `div`).

Since our example consists of only three sample events, we will divide the events into two pages. The first page will contain **My Sample Event #1** and **My Sample Event #2**, while the second page will contain **My Sample Event #3**. We will use a `div` element to represent an individual page. Each page's `div` will consist of a unique `id`, the word `page`, followed by the page number. The pagination control will be added in after our last event. To do this, add an empty `div` for holding the pagination below the last of our pages. It should also be assigned a unique `id`:

```
<div class="row">  
    <div id="page-1">  
        <h3>My Sample Event #1</h3>  
        <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.  
        Curabitur leo dolor,  
        fringilla vel lacus at, auctor finibus ipsum. Lorem ipsum
```

```
dolor sit amet,  
consectetur adipiscing elit. Morbi quis arcu lorem. Vivamus  
elementum convallis  
enim sagittis tincidunt. Nunc feugiat mollis risus non dictum.  
Nam commodo nec  
sapien a vestibulum. Duis et tellus cursus, laoreet ante non,  
mollis sem.  
Nullam vulputate justo nisi, sit amet bibendum ligula varius  
id.  
</p>  
<h3>My Sample Event #2</h3>  
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.  
Curabitur leo dolor,  
fringilla vel lacus at, auctor finibus ipsum. Lorem ipsum  
dolor sit amet,  
consectetur adipiscing elit. Morbi quis arcu lorem. Vivamus  
elementum convallis  
enim sagittis tincidunt. Nunc feugiat mollis risus non dictum.  
Nam commodo nec  
sapien a vestibulum. Duis et tellus cursus, laoreet ante non,  
mollis sem.  
Nullam vulputate justo nisi, sit amet bibendum ligula varius  
id.  
</p>  
</div>  
<div id="page-2">  
<h3>My Sample Event #3</h3>  
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.  
Curabitur leo dolor,  
fringilla vel lacus at, auctor finibus ipsum. Lorem ipsum  
dolor sit amet,  
consectetur adipiscing elit. Morbi quis arcu lorem. Vivamus  
elementum convallis  
enim sagittis tincidunt. Nunc feugiat mollis risus non dictum.  
Nam commodo nec  
sapien a vestibulum. Duis et tellus cursus, laoreet ante non,  
mollis sem.  
Nullam vulputate justo nisi, sit amet bibendum ligula varius  
id.  
</p>  
</div>  
<div id="services-events-pagination"></div>  
</div>
```

Before we will be able to actually use our pagination control, we must inform `bootpag` of its container. We do so by calling the `bootpag` function on our element, passing a configuration object as a parameter that contains our desired page count (10 in our case). Insert the following code into the head of our HTML document:

```
$('#services-events-pagination').bootpag({
    total: 10
}).on("page", function(event, num){});
```

The `bootpag` function will render the control to the element with an `id` equal to `services-events-pagination`, but notice the `on` event listener with the `page` parameter. This is our event listener, which will invoke the code contained within the (currently empty) callback as the user uses the pagination control to change pages. However, before we can implement the page change logic that will toggle the visibility of our individual pages, we must first hide our pages. To this end, we must update our `myphoto.css` file.

Now, one obvious approach would be to add a style for each one of our individual pages, identifying them by their `id`. As our number of events grows, this will seriously bloat our style sheet, as you will be required to add a new CSS rule for each new page. A much neater approach would be to wrap our pages within their own container and then use CSS selectors to hide all pages (that is, `div` elements) within this content area. To achieve this, first wrap the pages inside a new container `div` and assign this container a unique `id`:

```
<div id="services-events-content">
    <div id="page-1">
        <h3>My Sample Event #1</h3>
        <p>...</p>
        <h3>My Sample Event #2</h3>
        <p>...</p>
    </div>
    <div id="page-2">
        <h3>My Sample Event #3</h3>
        <p>...</p>
    </div>
</div>
```

Did you know?



There is an easier way to implement the aforementioned code. You could use Bootstrap's `hide` class, and toggle it in the callback. Try solving this yourself!

Then we update our style sheet so that the individual page `div` elements held within this new container are hidden by default:

```
#services-events-content div
{
    display: none;
}
```

Save and hit refresh. All of our events should now be hidden.

Now all we need to do is implement the logic that makes our individual pages visible as the user navigates. To this end, we complete the currently empty callback function, so that it first hides all pages and only then displays the currently selected page. Hiding all the pages, instead of the previous page, makes our code much cleaner, as we require no logic to determine the previously selected page; instead, we just use a CSS selector to hide all `div` elements contained within our `services-events-content` container. The `bootpag` plugin informs us of the currently selected page number through the second parameter (here named `num`) passed to our callback function. As such, we can use this page number to construct the `id` of the `div` (page) that we wish to make visible:

```
$('#services-events-pagination').bootpag({
    total: 10
}).on("page", function(event, num){
    $('#services-events-content div').hide();
    var current_page = '#page-' + num;
    $(current_page).show();
});
```

Seeing how our style sheet hides all the pages, we should include a statement that makes the first page visible as the user first visits our page. To do this, simply add `$('#page-1').show();` to the head of our document, so that our code takes the following structure:

```
$('#page-1').show();
$('#services-events-pagination').bootpag({
    total: 10
}).on("page", function(event, num){
    // Pagination logic
});
```

Take a look at the following screenshot:

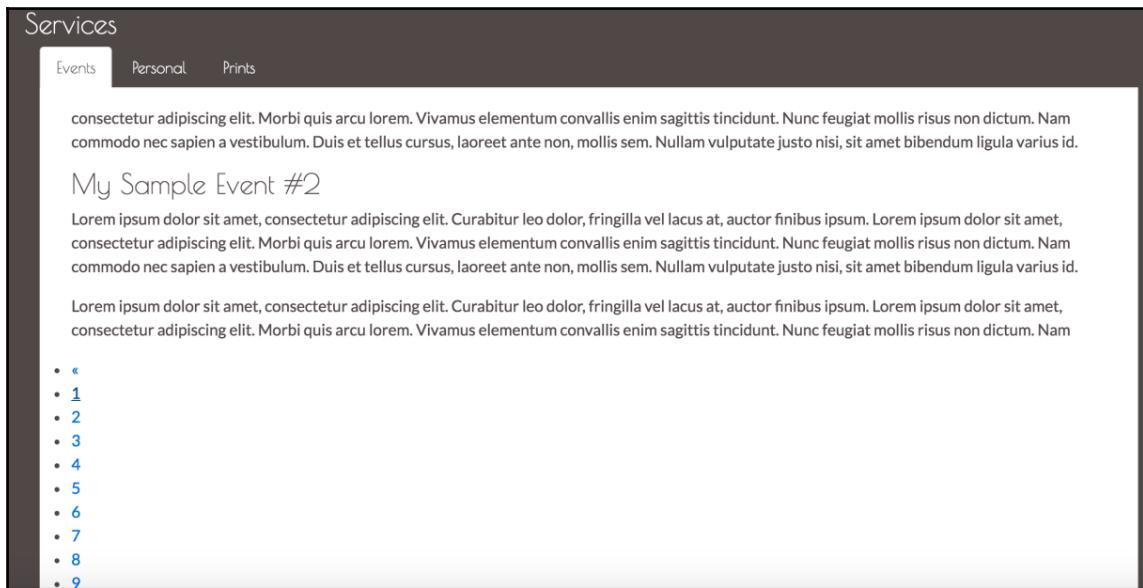


Figure 5.4: The display of the bootpag pagination controls is broken for Bootstrap 4. This is due to the changes to the pagination controls introduced by Bootstrap 4.

Hit save and refresh. While the pagination controls themselves are working, their display is broken (see *Figure 5.4*). This is due to the previously discussed changes to the pagination controls introduced by Bootstrap 4. Examining `jquery.bootpag.js`, we can see that the issue lies in constructing the pagination list items on line 130+. Observe the following code:

```
return this.each(function() {
    var $bootpag, lp, me = $(this),
        p = ['<ul class="', settings.wrapClass, ' bootpag">'];

    if(settings.firstLastUse) {
        p = p.concat(['<li data-lp="1" class="', settings.firstClass,
                      '"><a href="', href(1), '">', settings.first, '</a></li>']);
    }
    // ...
});
```

As the pagination items are being created for Bootstrap 3, the problem here lies with the fact that the code generating the items fails to apply the `page-item` and `page-link` classes. We can fix this easily enough. First, create a new folder, `js`, in our project root. Copy the `jquery.bootpag.js` file into this folder. Update the pagination markup generation logic so that the `page-item` and `page-link` classes are being applied to the list item and anchor elements:

```
return this.each(function(){
    var $bootpag, lp, me = $(this),
        p = ['<ul class="', settings.wrapClass, ' bootpag">'];

    if(settings.firstLastUse){
        p = p.concat(['<li data-lp="1" class="page-item ',
                      settings.firstClass, '"><a class="page-link" href="', href(1),
                      '">',
                      settings.first, '</a></li>']);
    }
    if(settings.prev){
        p = p.concat(['<li data-lp="1" class="page-item ',
                      settings.prevClass, '"><a class="page-link" href="', href(1),
                      '">',
                      settings.prev, '</a></li>']);
    }
    for(var c = 1; c <= Math.min(settings.total, settings.maxVisible);
        c++){
        p = p.concat(['<li class="page-item" data-lp="', c, '"><a
                     class="page-link" href="', href(c), '">', c, '</a></li>']);
    }
    if(settings.next){
        lp = settings.leaps && settings.total > settings.maxVisible
            ? Math.min(settings.maxVisible + 1, settings.total) : 2;
        p = p.concat(['<li data-lp="', lp, '" class="page-item ',
                      settings.nextClass, '"><a class="page-link" href="', href(lp),
                      '">', settings.next, '</a></li>']);
    }
    if(settings.firstLastUse){
        p = p.concat(['<li data-lp="', settings.total, '" class=
                     "page-item
                     last"><a class="page-link" href="', href(settings.total), '">',
                     settings.last, '</a></li>']);
    }
});
```

Finally, update the reference in the document head to point to our modified version of bootpag:

```
<script src="js/jquery.bootpag.js"></script>
```

Take a look at the following screenshot:

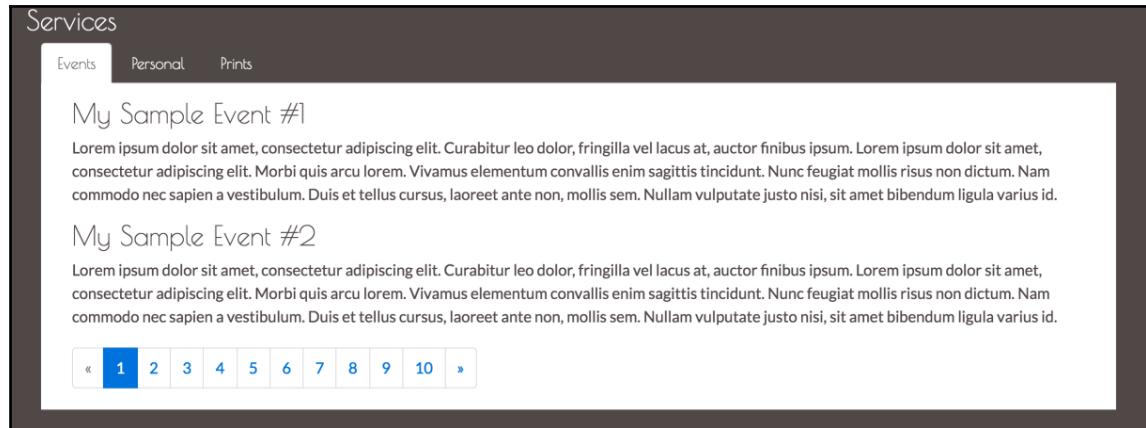


Figure 5.5: Pagination using our modified version of the bootpag plugin.

As you paginate to our second page, you will spot one issue: the last page may, at times, contain only one event (as is the case with **My Sample Event #3**), and the event descriptions differ in their lengths. Hence, there will be a height difference, which becomes apparent as the user switches pages. As a consequence, the pagination control `div` will move up and down (see *Figure 5.6*). Luckily, the fix for this is straightforward, and involves assigning our `event-services-content` `div` a fixed height of `15em`. Open the `myphoto.css` and add the following:

```
#services-events-content {  
    height: 15em;  
}
```

Take a look at the following screenshot:

The image contains two screenshots of a web application interface. Both screenshots show a header with tabs: 'Events' (which is active), 'Personal', and 'Prints'. Below the header, there are two sections, each containing an event thumbnail, the event title, and a long text description.

Screenshot 1 (Top):

- Event #1:** My Sample Event #1
- Description:** Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur leo dolor, fringilla vel lacus at, auctor finibus ipsum. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi quis arcu lorem. Vivamus elementum convallis enim sagittis tincidunt. Nunc feugiat mollis risus non dictum. Nam commodo nec sapien a vestibulum. Duis et tellus cursus, laoreet ante non, mollis sem. Nullam vulputate justo nisi, sit amet bibendum ligula varius id.

Screenshot 2 (Bottom):

- Event #2:** My Sample Event #2
- Description:** Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur leo dolor, fringilla vel lacus at, auctor finibus ipsum. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi quis arcu lorem. Vivamus elementum convallis enim sagittis tincidunt. Nunc feugiat mollis risus non dictum. Nam commodo nec sapien a vestibulum. Duis et tellus cursus, laoreet ante non, mollis sem. Nullam vulputate justo nisi, sit amet bibendum ligula varius id.

Both screenshots feature a pagination control at the bottom of the event list, consisting of a series of numbered buttons (1 through 10) and navigation arrows ('«' and '»').

Figure 5.6: Notice the height difference between the two pages. Having a different number of events per page or listing events with differing descriptions will result in the container growing and shrinking.

Now that our `events` container is of a fixed height, we can be certain that the container will not shrink based on its content. As a result, the pagination control will remain fixed in its position. However, this raises one final issue, of long event descriptions. How can we deal with events that contain more text than is permissible by our `events` container? For example, consider the additional paragraph added to **My Sample Event #2** in *Figure 5.7*. As you can see, the pagination control is now rendered above the event description.

Any text exceeding our container's height is simply cut:

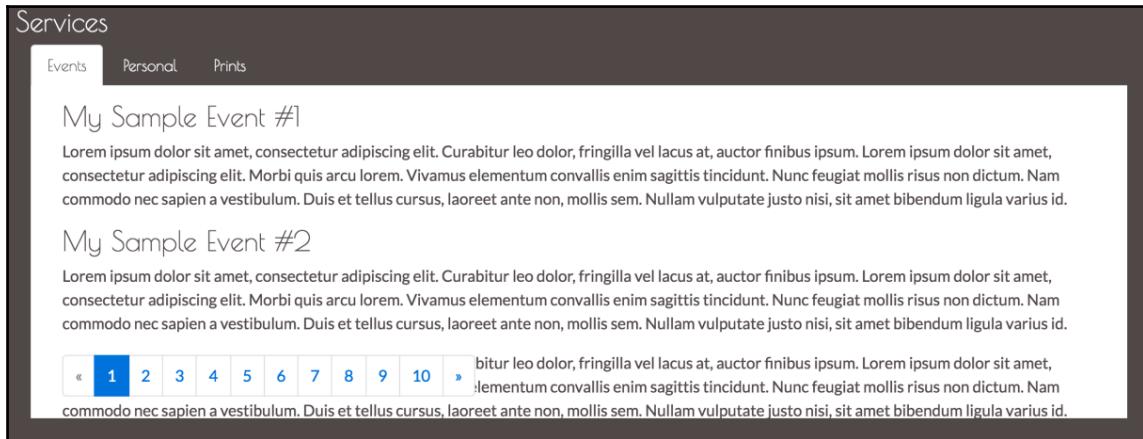


Figure 5.7: Display bug: Long event descriptions result in the pagination control being rendered above the event description. Any text exceeding our container's height is cut.

Once again, our fix is a simple one-liner, and involves setting the container's Y-axis overflow so that any content within the container is scrollable. Open `myphoto.css` and update the styling for our `services-events-content` container so that its `overflow-y` property is set to `scroll`:

```
#services-events-content {  
    height: 15em;  
    overflow-y: scroll;  
}
```

Save and refresh. Voila!

Displaying images using Bootstrap Lightbox

One important feature missing from our **Events** section is the ability to include images that illustrate an event (or provide additional information). Sure, you can add images using the `img` tag, but that may not be very practical, as the image size will be limited by the container's dimensions.

In this section, we will demonstrate how we can overcome this limitation by allowing users to enlarge images as they click on them, without redirecting them away from our page. To this end, go ahead and embed one image with each event (see *Figure 5.8*). Each image should be aligned to the left of the event description, have a width of 80, and a height of 45:

```
<div id="page-1">
    <h3>My Sample Event #1</h3>
    <p>
        
        Lorem ipsum dolor sit amet, consectetur adipiscing elit.
        Curabitur leo dolor,
        fringilla vel lacus at, auctor finibus ipsum. Lorem ipsum dolor
        sit amet,
        consectetur adipiscing elit. Morbi quis arcu lorem. Vivamus
        elementum convallis
        enim sagittis tincidunt. Nunc feugiat mollis risus non dictum.
        Nam commodo nec
        sapien a vestibulum. Duis et tellus cursus, laoreet ante non,
        mollis sem.
        Nullam vulputate justo nisi, sit amet bibendum ligula varius
        id.
    </p>
    <h3>My Sample Event #2</h3>
    <p>
        
        Lorem ipsum dolor sit amet, consectetur adipiscing elit.
        Curabitur leo dolor,
        fringilla vel lacus at, auctor finibus ipsum. Lorem ipsum
        dolor sit amet,
        consectetur adipiscing elit. Morbi quis arcu lorem. Vivamus
        elementum convallis
        enim sagittis tincidunt. Nunc feugiat mollis risus non dictum.
        Nam commodo nec
        sapien a vestibulum. Duis et tellus cursus, laoreet ante non,
        mollis sem.
        Nullam vulputate justo nisi, sit amet bibendum ligula varius
        id.
    </p>
</div>
```

As you save the preceding markup and refresh the page, you will notice that the images are not very nicely aligned with the text. They appear somewhat squashed. We can improve their appearance by adding some spacing between the images, the text, and the top of the container. To do this, go ahead and add a top margin of `0.5em` and a right margin of `1em` to each image within our `services-events-content` container:

```
#services-events-content div img {  
    margin-top: 0.5em;  
    margin-right: 1em;  
}
```



Did you know?

You can solve the aforementioned problem using a specific class (which may in fact be a neater solution). Try and improve these rules by yourself!

Take a look at the following screenshot:

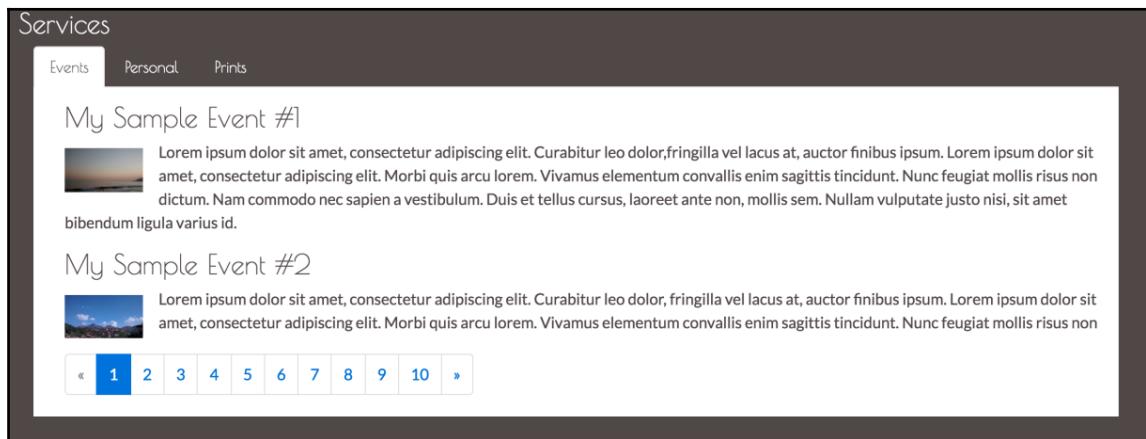


Figure 5.8: Sample images accompanying each event description. The images are left-aligned and have a dimension of 80 x 45.

One popular third-party library that allows users to enlarge the images embedded within the event description is Bootstrap Lightbox, available via GitHub at <https://github.com/jbutz/bootstrap-lightbox>. Unfortunately, the plugin is no longer maintained and ships with several unfixed bugs and usability issues. Upon downloading it, you will find that it does not immediately work out of the box. Luckily, DJ Interactive (<http://djinteractive.co.uk>) extended the original Bootstrap Lightbox through Lightbox for Bootstrap. Also available via GitHub (<https://github.com/djinteractive/Lightbox-for-Bootstrap>), the plugin is published under the Creative Commons Attribution 2.5 license. This means that the plugin is free for use under the condition that the author of the plugin is properly attributed.

Go ahead and download Lightbox for Bootstrap, and include both its JavaScript and CSS files within the head of our HTML document:

```
<script src="bower_components/lightbox-for-bootstrap
```

```
/js/bootstrap.lightbox.js"></script>
<link rel="stylesheet" href="bower_components/bootstrap-lightbox
/css/bootstrap.lightbox.css" />
```

Using the plugin to display our images within a lightbox fortunately requires hardly any modification to our existing markup. The only two steps to undertake are as follows:

1. Place our existing `img` element inside a container element that has a `thumbnail` class and `data-toggle` attribute.
2. Apply the `thumbnail` class and `data-target` attribute to our `img` element:

```
<p>
  <span class="thumbnails" data-toggle="lightbox">
    
  </span>
  Lorem ipsum dolor sit amet, consectetur adipiscing elit.
  Curabitur leo
  dolor, fringilla vel lacus at, auctor finibus ipsum. Lorem
  ipsum dolor sit amet,
  consectetur adipiscing elit. Morbi quis arcu lorem. Vivamus
  elementum convallis
  enim sagittis tincidunt. Nunc feugiat mollis risus non dictum.
  Nam commodo nec
  sapien a vestibulum. Duis et tellus cursus, laoreet ante non,
  mollis sem.
  Nullam vulputate justo nisi, sit amet bibendum ligula varius
  id.
</p>
```

The `data-target` attribute tells Lightbox for Bootstrap where our larger image is located. It is important to note that the `src` attribute of our image element has no effect on the image that will be displayed within the lightbox; only the `data-target` attribute determines this. As such, we could display a thumbnail, which then actually links to a different lightbox image (although this would make little sense and be grossly misleading). Unsurprisingly, the `data-toggle` attribute is used to identify the element that serves as the lightbox toggle, in our case, the toggle is our 80 x 45 image. However, it is important to note that toggles do not need to be image elements. Any element can become a lightbox toggle.

Last but not least, the `thumbnails` class serves as selectors for the `lightbox` plugin. Without them, the desired functionality would not work. Take a look at the following screenshot:



Figure 5.9: An enlarged image displayed using Lightbox for Bootstrap.

To summarize, our **Events** section should now look as follows:

```
<div class="container">
    <div class="row" style="margin: 1em;">
        <div id="services-events-content">
            <div id="page-1">
                <h3>My Sample Event #1</h3>
                <p>
                    <span class="thumbnails" data-toggle="lightbox">
                        
                    </span>
                    Lorem ipsum...
                </p>
                <h3>My Sample Event #2</h3>
                <p>
                    <span class="thumbnails" data-toggle="lightbox">
                        
                    </span>
                    ...
                </p>
            </div>
        </div>
    </div>
</div>
```

```
        data-target="images/event2.jpg"/>
    </span>  Lorem ipsum...
</p>
</div>
<div id="page-2">
    <h3>My Sample Event #3</h3>
    <p>
        <span class="thumbnails" data-toggle="lightbox">
            
        </span>
        Lorem ipsum...
    </p>
</div>
</div>
<div id="services-events-pagination"></div>
</div>
</div>
```

Improving our price list with DataTables

With the **Events** section in place, it is time to move onto our price list that we built in Chapter 2, *Making a Style Statement* and Chapter 3, *Building the Layout*. For the data that is currently displayed, the existing table structure works perfectly fine. The prices are nicely presented, and the table is not too crowded. However, what if MyPhoto were required to display hundreds of prices (yes, this case may seem far fetched, but bear with it for demonstration purposes)? Our existing table structure would far exceed its display capacity; the columns would be too crowded and we would need to implement some form of pagination to help keep the table organized. Of course, if you read the previous sections, you will know how easy it is to implement pagination using a third-party plugin. However, with hundreds or thousands of items, pagination will not be enough to make the website usable. Users may require more advanced features, such as the ability to filter tabular data, or the ability to search for a specific table item. Users may also desire the ability to adjust the number of table items displayed per page. All these requirements are bound to make our table implementation quite complex and challenging. However, once again, these user requirements are common, well understood, and well studied. Because of this, there is an excellent third-party library that we can use to enhance our MyPhoto price list. Meet DataTables (<https://www.datatables.net>). DataTables is a jQuery plugin that includes Bootstrap styles, and provides us with all of the previously mentioned features.

To use DataTables, you can either customize your own build via the DataTables website (they offer a neat download builder), or you can use Bower:

```
bower install DataTables
```

Once installed, you should find the following directory: `bower_components/DataTables`

Inside this directory, `media/` will contain both minified and un-minified JavaScript and CSS files, which we can include within the head of our document. Specifically, the directory will contain the normal jQuery plugin and styles, as well as the Bootstrap-specific styling:

- `dataTables.bootstrap.min.js`
- `jquery.dataTables.min.js`
- `dataTables.bootstrap.min.css`

Let's go ahead and incorporate the files:

```
<script src="bower_components/DataTables/media  
/js/jquery.dataTables.min.js"></script>  
<script src="bower_components/DataTables/media  
/js/dataTables.bootstrap.min.js"></script>  
<link rel="stylesheet" href="bower_components/DataTables/media  
/css/dataTables.bootstrap.min.css" />
```

Before we can dive into our freshly included `dataTables`, we must reorganize our print sizes and prices. Go ahead and create a table, using the same dataset as before (however, for simplicity's sake, we can just display one price set):

```
<table id="services-prints-table">  
    <thead>  
        <tr>  
            <th>Extra Large</th>  
            <th>Large</th>  
            <th>Medium</th>  
            <th>Small</th>  
        </tr>  
    </thead>  
    <tbody>  
        <tr>  
            <td>24x36 (€60)</td>  
            <td>19x27 (€45)</td>  
            <td>12x18 (€28)</td>  
            <td>6x5 (€15)</td>  
        </tr>  
        <tr>  
            <td>27x39 (€75)</td>
```

```
<td>20x30 (€48)</td>
<td>16x20 (€35)</td>
<td>8x10 (€18)</td>
</tr>
<tr>
    <td>27x40 (€75)</td>
    <td>22x28 (€55)</td>
    <td>18x24 (€40)</td>
    <td>11x17 (€55)</td>
</tr>
</tbody>
</table>
```

The preceding table is a standard HTML table with a head and a body. Save and refresh. Good, we now have a plain and simple table. Next, let's go ahead and style it. First, set the table to use the entire available space by setting its width to 100%. Next, apply the following two Bootstrap classes: `table` and `table-striped`. The former class, `table`, applies a basic table styling to our table by adjusting the padding, line height, and vertical alignment. The latter class, `table-striped`, alternates the colors of our individual rows:

```
<table id="services-prints-table" class="table table-striped"
width="100%">
    <thead>
        <!-- Content here-->
    </thead>
    <tbody>
        <!--Content here-->
    </tbody>
</table>
```

To initialize the data table, we just need one line of code:

```
$('#services-prints-table').DataTable();
```

Save and refresh. Immediately, you will see that the table is flowing outside of our container. To solve this, wrap the table element into a `div`, and give this `div` a maximum width of 90% (note that we are using inline styles for demonstration purposes only, and you should always try to avoid inline styles):

```
<div style="max-width: 90%;">
    <table id="services-prints-table" class="display">...</table>
</div>
```

Once again, save and hit refresh. Voila! Our table is displaying nicely (see *Figure 5.10*). Go ahead and play with it for a bit. Use the search box to filter specific table rows, or add more table rows and see how the table becomes magically pageable. You can even control the number of entries to display per page without any additional effort. Take a look at the following screenshot:

The screenshot shows a web application interface titled "Services". At the top, there are three tabs: "Events", "Personal", and "Prints", with "Prints" being the active tab. Below the tabs, a blue header bar contains the text "Special Offers". The main content area is titled "Our Print Sizes". It features a table with four columns: "Extra Large", "Large", "Medium", and "Small". Each column has a corresponding checkbox above it. Below the table, there is a search bar labeled "Search:" and a pagination section showing "Showing 1 to 3 of 3 entries". The pagination includes links for "Previous", "1", and "Next".

Extra Large	<input type="checkbox"/> Large	<input type="checkbox"/> Medium	<input type="checkbox"/> Small
27x39 (€75)	20x30 (€48)	16x20 (€35)	8x10 (€18)
24x36 (€60)	19x27 (€45)	12x18 (€28)	6x5 (€15)
27x40 (€75)	22x28 (€55)	18x24 (€40)	11x17 (€55)

Figure 5.10: The non-Bootstrap variant of our DataTables price list. Note the broken pagination display.

Note how, just as with `bootpag`, the `dataTable` plugin applies Bootstrap 3 pagination styling. To make the plugin Bootstrap 4-compatible, first copy `dataTables.bootstrap.js` into the `js` folder (which we created when fixing `bootpag`) and navigate to line 63. The `switch` statement is what determines the list item classes to use. Immediately after the `switch` statement, add the following line:

```
btnClass += 'page-item table-page-item';
```

Furthermore, update line 109 (which generates the anchor element) to include the `page-link` class:

```
.append( $('<a>', {
    'class': 'page-link',
    'href': '#',
    'aria-controls': settings.sTableId,
    'data-dt-idx': counter,
    'tabindex': settings.iTabIndex
  })
  .html( btnDisplay )
```

)

Note how, along with the `page-item` class, we also added the `table-page-item` class, which we now define as follows:

```
.table-page-item {  
    margin-left: 0.5rem;  
}
```

Take a look at the following screenshot:

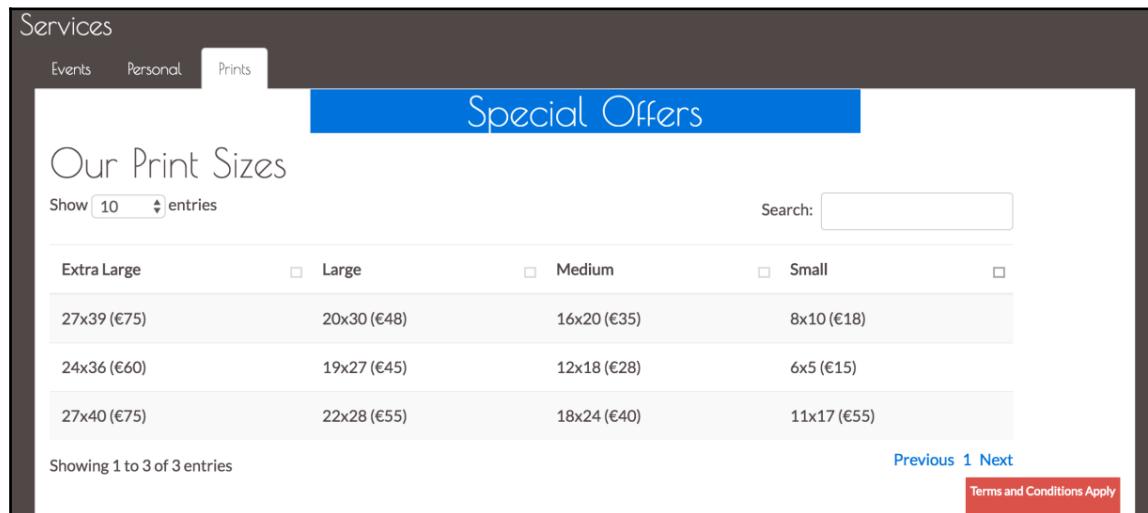


Figure 5.11: The fixed DataTable, complete with pagination controls and a Terms and Conditions Apply label.

To conclude this section, our complete price section should now look as follows:

```
<div class="container">  
    <h1 class="hidden-md">Our Print Sizes</h1>  
    <div style="max-width: 90%; ">  
        <table id="services-prints-table" class="table table-striped"  
            width="100%">  
            <thead>  
                <tr>  
                    <th>Extra Large</th>  
                    <th>Large</th>  
                    <th>Medium</th>  
                    <th>Small</th>  
                </tr>  
            </thead>  
            <tbody>
```

```
<tr>
    <td>24x36 (€60)</td>
    <td>19x27 (€45)</td>
    <td>12x18 (€28)</td>
    <td>6x5 (€15)</td>
</tr>
<tr>
    <td>27x39 (€75)</td>
    <td>20x30 (€48)</td>
    <td>16x20 (€35)</td>
    <td>8x10 (€18)</td>
</tr>
<tr>
    <td>27x40 (€75)</td>
    <td>22x28 (€55)</td>
    <td>18x24 (€40)</td>
    <td>11x17 (€55)</td>
</tr>
</tbody>
</table>
</div>
</div>
```

Summary

In this chapter, we learned how to solve common user interface development requirements using third-party libraries. By improving various functionalities built out during the course of the previous chapters, we demonstrated how to detect a user's browser and its version using the `jQuery browser` plugin. We used the `bootpag` plugin to allow users to navigate data. Along with it, we enhanced the look and feel of our **Events** section using images and Lightbox for Bootstrap. We also looked at improving the usability of our price list using `DataTables`.

Armed with the knowledge of how to use popular and widely adopted third-party libraries, we are now ready to customize Bootstrap's `jQuery` plugins to suit our exact needs. Lets go to the next chapter!

6

Customizing Your Plugins

So far, we have built the `MyPhoto` demo page leveraging all that Bootstrap has to offer, customizing Bootstrap's themes and components, and using jQuery plugins along the way. In this chapter, we will be delving deep into Bootstrap's jQuery plugins with extensive customization via JavaScript and CSS.

We will take some of the plugins we have introduced into `MyPhoto`, take a look under the hood, and, step by step, we will customize them to meet the needs of our page. Plugins will be examined and extended throughout this chapter in an effort to not only make our page better, but to also build our knowledge of how jQuery plugins are built and behave within Bootstrap's ecosystem.

When we are comfortable with customizing Bootstrap's jQuery plugins, we will create a fully customized jQuery plugin of our own for `MyPhoto`.

Summarizing all of this, in this chapter we will globally do the following:

- Learn about the anatomy of a Bootstrap jQuery plugin
- Learn how to extensively customize the behavior and features of Bootstrap's jQuery plugins via JavaScript
- Learn how to extensively customize the styling of Bootstrap's jQuery plugins via CSS
- Learn how to create a custom Bootstrap jQuery plugin from scratch

Anatomy of a plugin

Bootstrap jQuery plugins all follow the same convention in how they are constructed. At the top level, a plugin is generally split across two files, a JavaScript file and a Sass file. For example, the Alert component is made up of `bootstrap/js/alert.js` and `bootstrap/scss/_alert.scss`. These files are compiled and concatenated as part of Bootstrap's distributable JavaScript and CSS files. Let us look at these two files in isolation to learn about the anatomy of a plugin.

JavaScript

Open up any JavaScript file in `bootstrap/js/src`, and you will see that they all follow the same pattern: an initial setup, a class definition, data API implementation, and jQuery extension. Let's take a detailed look at `alert.js`.

Setup

The `alert.js` file, written in ECMAScript 2015 syntax (also known as ES6, the latest (at the time of writing) standardized specification of JavaScript), first imports a utilities module:

```
import Util from './util'
```

A constant is then created, named `Alert`, which is assigned the result of an **Immediately Invoked Function Expression (IIFE)**:

```
const Alert = ($ => {
  ...
}) (jQuery)
```

A `jQuery` object is being passed into a function for execution, the result of which will be assigned to the immutable `Alert` constant.

Within the function itself, a number of constants are also declared for use throughout the rest of the code. Declaring immutables at the beginning of the file is generally seen as best practice. Observe the following code:

```
const NAME          = 'alert'
const VERSION      = '4.0.0-alpha'
const DATA_KEY     = 'bs.alert'
const EVENT_KEY    = '.${DATA_KEY}'
const DATA_API_KEY = '.data-api'
const JQUERY_NO_CONFLICT = $.fn[NAME]
```

```
const TRANSITION_DURATION = 150

const Selector = {
  DISMISS : '[data-dismiss="alert"]'
}
const Event = {
  CLOSE : 'close${EVENT_KEY}',
  CLOSED : 'closed${EVENT_KEY}',
  CLICK_DATA_API : 'click${EVENT_KEY}${DATA_API_KEY}'
}

const ClassName = {
  ALERT : 'alert',
  FADE : 'fade',
  IN : 'in'
}
```

The NAME property is the name of the plugin, and VERSION defines the version of the plugin, which generally correlates to the version of Bootstrap. DATA_KEY, EVENT_KEY, and DATA_API_KEY relate to the data attributes that the plugin hooks into, while the rest are coherent, more readable, aliases for the various values used throughout the plugin code. Following that is the class definition.



Immediately Invoked Function Expression

An Immediately Invoked Function Expression (IIFE or iffy) is a function which is executed as soon as it has been declared, and is known as a self-executing function in other languages. A function is declared as an IIFE by either wrapping the function in parentheses or including a preceding unary operator, and including a trailing pair of parentheses. Examples:

```
(function(args){ }) (args)  
!function(args){ } (args)
```

Class definition

Near the top of any of the plugin JS files, you will see a comment declaring the beginning of the class definition for that particular plugin. In the case of alerts, it is:

```
/**  
* -----  
-----  
* Class Definition
```

```
* -----
-----
*/
```

The class definition is simply the constructor of the base object, in this case, the `Alert` object:

```
class Alert {
    constructor(element) {
        this._element = element
    }
    ...
}
```

The convention with plugins is to use Prototypal inheritance. The `Alert` base object is the object all other `Alert` type objects should extend and inherit from. Within the class definition, we have the public and private functions of the `Alert` class. Let's take a look at the public `close` function:

```
close(element) {
    element = element || this._element
    let rootElement = this._getRootElement(element)
    let customEvent = this._triggerCloseEvent(rootElement)
    if (customEvent.isDefaultPrevented()) {
        return
    }

    this._removeElement(rootElement)
}
```

The `close` function takes an `element` as an argument, which is the reference to the DOM element the `close` function is to act upon. The `close` function uses the private function `_getRootElement` to retrieve the specific DOM element, and `_triggerCloseEvent` to reference the specific event to be processed. Finally, `close` calls `_removeElement`. Let's take a look at these private functions:

```
_getRootElement(element) {
    let selector = Util.getSelectorFromElement(element)
    let parent   = false

    if (selector) {
        parent = $(selector)[0]
    }

    if (!parent) {
        parent = $(element).closest(`.${ClassName.ALERT}`)[0]
    }
}
```

```
        return parent
    }
```

The `_getRootElement` tries to find the parent element of the DOM element passed to the calling function, in this case, `close`. If a parent does not exist, `_getRootElement` returns the closest element with the class name defined by `ClassName.ALERT` in the plugin's initial setup. This in our case is `Alert`. Observe the following code:

```
_triggerCloseEvent(element) {
    let closeEvent = $.Event(Event.CLOSE)
    $(element).trigger(closeEvent)
    return closeEvent
}
```

The `_triggerCloseEvent` also takes an `element` as an argument and triggers the event referenced in the plugin's initial setup by `Event.CLOSE`:

```
_removeElement(element) {
    $(element).removeClass(ClassName.IN)
    if (!Util.supportsTransitionEnd() ||
        !$(element).hasClass(ClassName.FADE)) {
        this._destroyElement(element)
        return
    }
    $(element)
        .one(Util.TRANSITION_END, $.proxy(this._destroyElement,
            this, element))
        .emulateTransitionEnd(TRANSITION_DURATION)
}
```

The `_removeElement` then carries out the removal of the `rootElement` safely and in accordance with the configuration in the element itself, or as defined in the plugin's initial setup, for example, `TRANSITION_DURATION`.

All core behaviors and functions of the plugin should be defined in the same manner as the `close` function. The class definition represents the plugin's essence.

After the public and private functions come the static functions. These functions, which are also private, are similar to what would be described as the plugin definition in Bootstrap 3. Observe the following code:

```
static _jQueryInterface(config) {
    return this.each(function () {
        let $element = $(this)
        let data = $element.data(DATA_KEY)
        if (!data) {
```

```
        data = new Alert(this)
        $element.data(DATA_KEY, data)
    }
    if (config === 'close') {
        data[config](this)
    }
})
}
static _handleDismiss(alertInstance) {
    return function (event) {
        if (event) {
            event.preventDefault()
        }
        alertInstance.close(this)
    }
}
```

The `_jQueryInterface` is quite simple. First, it loops through an array of DOM elements. This array is represented here by the `this` object. It creates a jQuery wrapper around each element and then creates the `Alert` instance associated with this element, if it doesn't already exist. `_jQueryInterface` also takes in a `config` argument. As you can see, the only value of `config` that `_jQueryInterface` is concerned with is '`'close'`'. If `config` equals '`'close'`', then the `Alert` will be closed automatically.

`_handleDismiss` simply allows for a specific instance of `Alert` to be programmatically closed.

Following the class definition, we have the data API implementation.

Data API implementation

The role of the data API implementation is to create JavaScript hooks on the DOM, listening for actions on elements with a specific data attribute. In `alert.js`, there is only one hook:

```
$(document).on(
    Event.CLICK_DATA_API,
    Selector.DISMISS,
    Alert._handleDismiss(new Alert())
)
$(document).on('click.bs.alert.data-api', dismiss,
    Alert.prototype.close)
```

The hook is an on-click listener on any element that matches the dismiss selector.

When a click is registered, the `close` function of `Alert` is invoked. The dismiss selector here has actually been defined at the beginning of the file, in the plugin setup:

```
const Selector = {
    DISMISS : '[data-dismiss="alert"]'
}
```

Therefore, an element with the attribute `data-dismiss="alert"` will be hooked in, to listen for clicks. The `click` event reference is also defined in the setup:

```
const Event = {
    CLOSE          : 'close${EVENT_KEY}',
    CLOSED         : 'closed${EVENT_KEY}',
    CLICK_DATA_API: 'click${EVENT_KEY}${DATA_API_KEY}'
}
```

`EVENT_KEY` and `DATA_API_KEY`, if you remember, are also defined here:

```
const DATA_KEY      = 'bs.alert'
const EVENT_KEY     = '.${DATA_KEY}'
const DATA_API_KEY  = '.data-api'
```

We could actually rewrite the API definition to read as follows:

```
$(document).on('click.bs.alert.data-api', '[data-dismiss="alert"]',
Alert._handleDismiss(new Alert()))
```

The last piece of the puzzle is the jQuery section, which is a new feature in Bootstrap 4. It is a combination of Bootstrap 3's plugin definition and a conflict prevention pattern.

jQuery

The jQuery section is responsible for adding the plugin to the global jQuery object so that it is made available anywhere in an application where jQuery is available. Let's take a look at the code:

```
$.fn[NAME]           = Alert._jQueryInterface
$.fn[NAME].Constructor = Alert
$.fn[NAME].noConflict = function () {
    $.fn[NAME] = JQUERY_NO_CONFLICT
    return Alert._jQueryInterface
}
```

The first two assignments extend jQuery's prototype with the plugin function. As Alert is created within a closure, the constructor itself is actually private. Creating the Constructor property on `$.fn.alert` allows it to be accessible publicly.

Then, a property of `$.fn.alert` called `noConflict` is assigned the value of `Alert._jQueryInterface`. The `noConflict` property comes into use when trying to integrate Bootstrap with other frameworks to resolve issues with two jQuery objects with the same name. If in some framework the Bootstrap Alert got overridden, we could use `noConflict` to access the Bootstrap Alert and assign it to a new variable:

```
$.fn.bsAlert = $.fn.alert.noConflict()
```

`$.fn.alert` is the framework version of Alert, but we have transferred the Bootstrap Alert to `$.fn.bsAlert`.

All plugins tend to follow the pattern of initial setup, class definition, data API implementation, and jQuery extension. To accompany the JavaScript, a plugin also has its own specific Sass style sheet.

Sass

Sass files for plugins aren't as formulaic as the corresponding JavaScript. In general, JavaScript hooks into classes and attributes to carry out a generally simple functionality. In a lot of cases, much of the functionality is actually controlled by the style sheet; the JavaScript simply adds and removes classes or elements under certain conditions. The heavy lifting is generally carried out by the Sass, so it is understandable that the Sass itself may not fit into a uniform pattern.

Let's take a look at `scss/_alert.scss`. The `_alert.scss` opens up with a base style definition. Most, but not all, plugins will include a base definition (usually preceded by a base style or base class comment). Defining the base styles of a plugin at the beginning of the Sass file is best practice for maintainability and helps anyone who might want to extend the plugin to understand it.

Following the base styles, the styles associated with, or responsible for, the functionality of the plugin are defined. In the case of alerts, the dismissible alert styles are defined. The only piece of functionality an alert has, besides being rendered on the page, is to be dismissed. This is where Alerts defines what should happen when the `close` class is applied to an `Alerts` element.

The Sass will also generally include an alternate style definition. The alternate styles generally align with Bootstrap's contextual classes, which we explored in Chapter 2, *Making a Style Statement*. Observe the following code:

```
// Alternate styles
//
// Generate contextual modifier classes for colorizing the alert.
.alert-success {
    @include alert-variant($alert-success-bg, $alert-success-border,
    $alert-success-text);
}
.alert-info {
    @include alert-variant($alert-info-bg, $alert-info-border,
    $alert-info-text);
}
.alert-warning {
    @include alert-variant($alert-warning-bg, $alert-warning-border,
    $alert-warning-text);
}
.alert-danger {
    @include alert-variant($alert-danger-bg, $alert-danger-border,
    $alert-danger-text);
}
```

As you can see, `alert` provides styles to correspond with the `success`, `info`, `warning`, and `danger` contexts. The variables used in the rules, such as `$alert-danger-bg`, are declared in `_variables.scss`. Declaring variables in a `_variables.scss` file is best practice, as otherwise maintenance would be supremely difficult. For instance, open up `_variables.scss` and see the definition for `$alert-danger-bg`:

```
$alert-warning-bg: $state-warning-bg !default;
```

The `$state-warning-bg` is another variable, but this variable is used for all form feedback and alert warning background variables. If we wanted to change the color that the warning context corresponds to, we would just need to change the value in one place:

```
$state-warning-bg: #fcf8e3 !default;
```

Beyond the base styles and, to an extent, the alternate styles, there is no real template for plugging in the Sass files.

The JavaScript file and the Sass file are the two ingredients that make a plugin work. Looking at the example from Chapter 4, *On Navigation, Footers, Alerts, and Content*, we can see the alert plugin in action:

```
<div class="alert alert-danger">
  <a href="#" class="close" data-dismiss="alert"
    aria-label="close">&times;</a>
  <strong class="alert-heading"><i class="fa fa-
    exclamation"></i> Unsupported browser</strong>
    Internet Explorer 8 and lower are not supported by this website.
</div>
```

Let's start customizing plugins.

Customizing plugins

While there are many plugins to customize, we will choose two that we have already come across in the previous chapters:

- Bootstrap's jQuery alert plugin
- Bootstrap's jQuery carousel plugin

Customizing Bootstrap's jQuery alert plugin

The alert plugin, as we have seen, is exceedingly simple. The alert is rendered on the page, displaying a message, and the only functionality it has is the ability to close and disappear when a user clicks on a certain element.

To demonstrate how to customize or extend a plugin, in this case alert, we are going to keep it very simple. We are going to add an extra bit of functionality, when a user clicks on a certain element, the alert will minimize. We also, obviously, want to give the user the ability to expand the alert when it is in its minimized state. To do this, we need to extend both the JavaScript and the styling of alert.

Before we get to the coding of the plugin functionality and styling, let's put together the markup for an alert on the MyPhoto page.

The markup

As an example use case, let's display an alert informing the user of a special offer. We will add our alert above the unsupported browser alert:

```
<div class="alert alert-info" style="position: fixed; margin-top: 4em; width: 90%; margin-left: 4em;">
    <a href="#" class="close" data-dismiss="alert" aria-label="close">&times;</a>
    <strong><i class="fa fa-exclamation"></i> Special Offer - </strong>
    <span>2 FOR 1 PRINTS TODAY ONLY WITH PROMO CODE <span style="font-style: italic">BOOTSTRAP</span></span>
</div>
```

We're using Bootstrap's contextual information class, `alert-info`, to style the alert box, and we're following the same pattern as the unsupported browser alert. The special offer alert has inline styles applied, and the unsupported browser alert has styles linked to its id. Before we go any further, let's extract that out into a single class in `myphoto.css` for reusability and maintainability. Remove the `#unsupported-browser-alert` rules and add the following:

```
.alert-position {
    position: fixed;
    margin-top: 4em;
    width: 50%;
    margin-left: 25%;
    z-index: 10;
}
.alert-position #unsupported-browser-alert {
    display:none;
}
```

We've made some slight changes here. The alert will now have a hard-set width of 50% of the viewport and will be rendered 25% from the left. To make sure the alert is always rendered above any other content on the page, we set the `z-index` to 10. As long as no other elements have a higher `z-index`, then the alert will always be visible. Now, we remove the inline styles on the alert elements and add the `alert-position` class. We extend the class slightly for elements with the `unsupported-browser-alert` id to make sure it isn't displayed. Update the alert elements with the `alert-position` class:

```
<div class="alert alert-info alert-position">
```

Take a look at the following screenshot:

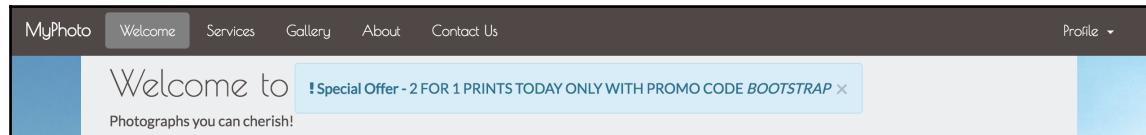


Figure 6.1: A modified alert used to display special promotions. Unlike the default alert, this alert has its width hard-set to 50% of the viewport and will be rendered 25% from the left.

Okay, great. We now have our special offers alert. Now let's add our minimize and expand elements. We want these elements to function and display similarly to the `close` element, so we can use the `close` element as a template. Observe the following code:

```
<a href="#" class="close" data-dismiss="alert"  
aria-label="close">&times;</a>  
<a href="#" class="close minimize" data-minimize="alert"  
aria-label="minimize">_</a>  
<a href="#" class="close expand" data-expand="alert"  
aria-label="expand">+</a>
```

We have replicated the `close` element twice. We have added a `minimize` and `expand` class, while retaining the `close` element as we want to inherit everything the `close` class includes. We have added new data attributes—instead of `data-dismiss`, we have `data-minimize` and `data-expand`. These are the data attributes that the plugin will listen to. We then updated the `aria-label` with the appropriate names, and applied appropriate content inside the element—an underscore (_) to indicate minimization and a plus (+) to indicate expansion (see *Figure 6.2*). Take a look at the following screenshot:

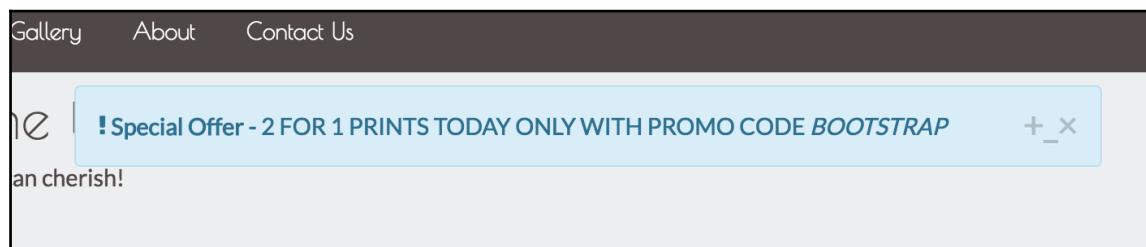


Figure 6.2: Our custom promotion alert with expand and minimize functionality

Besides the close button, we now have the expand and minimize buttons. We don't want to show the expand button when the alert is already expanded, and we don't want to show the minimize button when the alert is already minimized. As the alert is expanded by default, we'll add a `hidden-xs-up` class to the `expand` element. Recall that `hidden-xs-up` hides a

given element for viewports that are `xs` or larger. That is, the element will be hidden for all viewports (`hidden-xs-up` is the equivalent of `hide` in Bootstrap 3). Observe the following code:

```
<a href="#" class="close" data-dismiss="alert"  
aria-label="close">&times;</a>  
<a href="#" class="close minimize" data-minimize="alert"  
aria-label="minimize">_</a>  
<a href="#" class="close expand hide" data-expand="alert"  
aria-label="expand">+</a>
```

Take a look at the following screenshot:

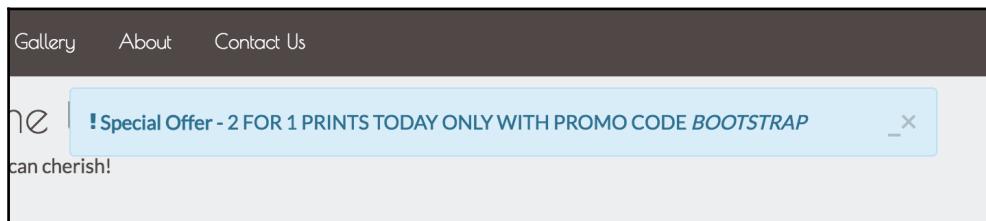


Figure 6.3: Our custom promotion alert with the `expand` element hidden

Nice. The alert is starting to look the way we want it. With that, we are ready to customize the styling of the alert plugin.

Extending alert's style sheets

As we mentioned before, it is bad practice to modify Bootstrap's Sass files directly, due to maintenance issues. Instead, we are going to create our own style sheet—`styles/alerts.css`.

Before we create any new classes, we should extract any alert-related CSS from `myphoto.css` into this new style sheet in order to improve code maintainability. The only classes we have so far are `alert-danger` and `alert-position`. Place them into our new alert specific style sheet, and include the style sheet in our HTML. Be sure to include it after `bootstrap.min.css` and `myphoto.css`, to make sure the style rules in `alert.css` take priority. Observe the following code:

```
<link rel="stylesheet" href="bower_components/bootstrap/dist/css/  
bootstrap.min.css" />  
<link rel="stylesheet" href="styles/myphoto.css" />  
<link rel="stylesheet" href="styles/alert.css" />
```

To create the ability to minimize and expand an alert, we actually do not need many style rules at all. In fact, we are going to use just one new class—`alert-minimize`. When a user clicks on the minimize button, the `alert-minimize` class will be applied to the root `alert` element. To expand it, the `alert-minimize` class will simply be removed.

Update `alert.css` with the following rules:

```
.alert-minimize {  
    width: 60px;  
}  
.alert-minimize * {  
    display: none;  
}  
.alert-minimize.close {  
    display: block;  
}
```

The `alert-minimize` class will force a `60px` element width. All descendants of the `alert-minimize` class will be given the `display` value of `none` so they do not appear on screen. To make sure the functional buttons are still visible, any element with the `close` class (remember we retained the `close` class for all our functional buttons in the alert) will be given the `display` value of `block`. Let's manually apply `alert-minimize` to our alert to see how it renders. Take a look at the following screenshot:

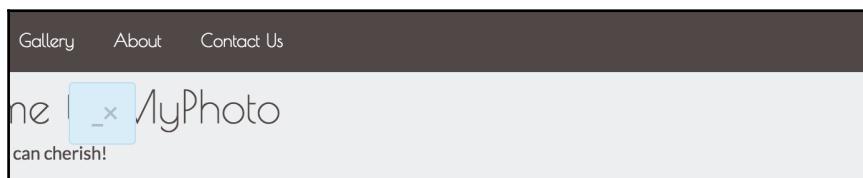


Figure 6.4: Our minimized custom promotion alert

Next up, let's remove the `alert-minimize` class and extend the alert plugin JavaScript to apply and remove the class dynamically.

Extending alert's functionality with JavaScript

As with extending the styles, to extend the JavaScript we could modify Bootstrap's `alert.js` directly, but, again, that is a bad idea in terms of maintainability. Instead, we are going to create a `js` directory in our project, and a file called `alert.js`. Include this file in your HTML, after `bootstrap.min.js`:

```
<script
```

```
src="bower_components/bootstrap/dist/js/bootstrap.min.js"></script>
<script src="js/alert.js"></script>
```

The first thing we are going to do is to create an immediately invoked function, and add the function to the `jQuery` object:

```
+function ($) {
  'use strict';
  var Alert = $.fn.alert.Constructor;
}(jQuery);
```

The function assigns a variable `Alert` to the `alert` plugin's prototype, which, as we saw earlier, is made available through the `Constructor` property.

With this reference to the `Alert` prototype, we can add our own functions to the prototype to handle minimizing and expanding an alert. Taking the `close` function we studied earlier, and with a few changes, let's create a function to minimize the alert:

```
Alert.prototype.minimize = function (e) {
  var $this = $(this)
  var selector = $this.attr('data-target')
  if (!selector) {
    selector = $this.attr('href')
    selector = selector && selector.replace(/.*(?:#[^\s]*$)/, '')
    // strip for ie7
  }
  $this.addClass('hidden-xs-up')
  $this.siblings('.expand').removeClass('hidden-xs-up')
  var $parent = $(selector)
  if (e) e.preventDefault()
  if (!$parent.length) {
    $parent = $this.closest('.alert')
  }
  $parent.trigger(e = $.Event('minimize.bs.alert'))
  if (e.isDefaultPrevented()) return
  $parent.addClass('alert-minimize')
}
```

The function is quite similar to the `close` function, so we will highlight the important differences. Line 15 and line 16 handle hiding the minimize button and showing the expand button, adding the `hide` class to the element that triggered the event, and removing the `hide` class from any sibling element with the `expand` class. Line 32 adds the `alert-minimize` class, which handles the shrinking of the `Alert` element, to the parent of the element that triggered the event. Essentially, the `minimize` function will shrink the alert, hide the minimize button, and show the expand button. Let's hook a listener up to this function.

We do this in the same way as the Bootstrap alert plugin links the data—dismiss the click event to the close function, adding the following to alert.js, below the minimize function definition:

```
$(document).on('click.bs.alert.data-api', '[data-minimize="alert"]',  
Alert.prototype.minimize)
```

Now, an element with the data-minimize attribute with an "alert" value will call the Alert.prototype.minimize function on a click event. The minimize element in our special offers alert has this attribute. Open up MyPhoto and click the minimize button. Take a look at the following screenshot:

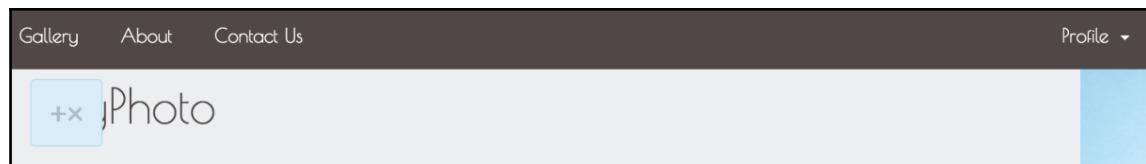


Figure 6.5: Our minimized custom promotion alert – note the expand and close buttons

Excellent. Our minimize button and functionality are wired up correctly to shrink our special offers alert and replace the minimize button with an expand button when clicked.

The last thing we need to do now is make sure the user can expand the alert when they click on the expand button. To do this, we follow the same steps as we did for the minimize functionality. Let's add an expand function to the Alert prototype:

```
Alert.prototype.expand = function (e) {  
    var $this = $(this)  
    var selector = $this.attr('data-target')  
    if (!selector) {  
        selector = $this.attr('href')  
        selector = selector && selector.replace(/.*(?:#[^\s]*$)/, '')  
        // strip for ie7  
    }  
    $this.addClass('hidden-xs-up')  
    $this.siblings('.minimize').removeClass('hide')  
    var $parent = $(selector)  
    if (e) e.preventDefault()  
    if (!$parent.length) {  
        $parent = $this.closest('.alert')  
    }  
    $parent.trigger(e = $.Event('expand.bs.alert'))  
    if (e.isDefaultPrevented()) return  
    $parent.removeClass('alert-minimize')
```

}

The differences between the `expand` and `minimize` functions are very small, so small that it probably makes sense for them to be encapsulated into one function. However, for the sake of simplicity, we will keep the two functions separate. Essentially, the actions of `minimize` are reversed. The `hidden-xs-up` class is again applied to the element triggering the event, the `hide` class is removed from any sibling with the `minimize` class, and the `alert-minimize` class is removed from the parent element. Simple and effective. Now, we just need to hook up a `click` event on an element with the `data-expand` attribute set to `alert` to the `expand` method. Observe the following code:

```
$ (document).on('click.bs.alert.data-api', '[data-expand="alert"]',
Alert.prototype.expand)
```

That's it. With our extension to the alert plugin, when a user clicks `expand` in the minimized state, the alert reverts back to its initial expanded state and the `expand` button is replaced by the `minimize` button. Our users now have the ability to reduce the screen real estate our alert covers, but are still able to retrieve the information from the alert at a later stage if needed.

While these alert customizations are relatively simple, they do provide a strong example of how to extend a plugin's functionality and teach principles that can be applied to more complex extensions.

Customizing Bootstrap's jQuery carousel plugin

`MyPhoto` uses Bootstrap's carousel as a gallery to display sample images. The carousel is a very neat component, allowing the user to cycle through images. We are going to add some new functionality to the carousel plugin. Specifically, we are going to implement the ability to surface a larger version of the image in a modal window when there is a `click` event on a carousel slide. We will be using Bootstrap's modal plugin to surface the modal, and we will dynamically pass the image source and carousel caption from the markup of the slide to the modal. First, let's write the markup.

The markup

The only thing we really need to do in the markup is create a modal element, and reference that modal in the carousel's slide elements, so as to link them together. First, let's create the modal. We only want a bare-bones modal here—an image, a close button, and a title. We've seen how to create modals before, so let's just add the markup we need to our HTML. We will add it just above the `carousel` element:

```
<div class="modal fade carousel-modal" id="carousel-modal"
    tabindex="-1" role="dialog">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <button type="button" class="close" data-dismiss="modal" aria-label="Close"><span aria-hidden="true">&times;</span></button>
                <h4 class="modal-title"></h4>
            </div>
            <div class="modal-body">
                <img>
            </div>
        </div>
    </div>
</div>
```

We have created a very simple modal here. We added a `carousel-modal` class to the parent element for any styles we may need to apply, and we attributed `carousel-modal` as the `id` for the modal. We have an empty `modal-title` element, which we will populate dynamically. Most interestingly, we have an empty `img` tag in the `modal-body` element. It isn't often that you see an `img` tag with no `src` attribute, but our extension will create this attribute dynamically. We could, of course, have created a different modal for each image, but that wouldn't scale, and it just wouldn't be interesting!

That's our simple modal window declared. Great. Now we just need to reference the modal in our slides. On each `img` element within the `carousel`, simply add a `data-modal-picture` attribute with the value `#carousel-modal`. Observe the following code:

```
<div class="carousel-inner" role="listbox">
    <div style="height: 400px" class="carousel-item active">
        
        <div class="carousel-caption">
            Brazil
        </div>
    </div>
    <div style="height: 400px" class="carousel-item">
```

```

<div class="carousel-caption">
    Datsun 260Z
</div>
</div>
<div style="height: 400px" class="carousel-item">
    
    <div class="carousel-caption">
        Skydive
    </div>
</div>
</div>
```

The `data-modal-picture` attribute is the `data-attribute` we are going to hook our `on-click` listener to, in the very same way that `alert` hooked into `data-dismiss`. Let's set up our carousel plugin extension and wire all this together.

Extending carousel's functionality with JavaScript

Just like with our alert extension, we will create a new JS file for the carousel extension. Create a `js/carousel.js` file and include the file on the `MyPhoto` page:

```
<script src="js/carousel.js"></script>
```

Again, we want to create an IIFE and assign the `carousel` constructor to a variable we can work with. Observe the following code:

```
+function ($) {
    'use strict';
    var Carousel = $.fn.carousel.Constructor;
} (jQuery);
```

From our markup, we know what `data-attribute` we want to listen to—`data-modal-picture`. Observe the following code:

```
+function ($) {
    'use strict';
    var Carousel = $.fn.carousel.Constructor;
    $(document).on('click.bs.carousel.data-api', '[data-modal-picture]', Carousel.prototype.zoom)
} (jQuery);
```

Notice that, unlike with alert, we are not referencing any particular value for the `data-modal-picture` attribute. We will be using the attribute value to identify which modal to use, so of course we want the plugin to be flexible enough to handle more than one modal id. We have also defined which function we want to call when the event is triggered—`Carousel.prototype.zoom`. Let's create that function:

```
Carousel.prototype.zoom = function () {
    var $this = $(this)
    var $src = $this.attr('src')
    var $title = $this.next('.carousel-caption').text()
    var $modal = $this.attr('data-modal-picture')
    var $modalElement = $.find($modal)
    $($modalElement).find('.modal-body').find('img').attr('src', $src)
    $($modalElement).find('.modal-title').text($title)
    $($modal).modal('show')
}
```

First, as before, we create a jQuery wrapper of the element that triggers the event. Next, we use the `attr` method to find the value of the element's `src` attribute. We then use the `next` method to find the next `carousel-caption` element, and assign the inner text of that element to `$title`. We need these to dynamically update the blank modal.

Next, we grab the value of the `data-modal-picture` element, which we then use as a reference to find the modal we want to use to render our picture. We use the `find` method to first find the `modal-body` of this element, then the nested image element. We then create an `src` attribute on this element, passing in a reference to the source of the slide's image element. Similarly, we inject the caption of the slide into the modal's `title` element.

Finally, we use the modal API to show the modal. Take a look at the following screenshot:

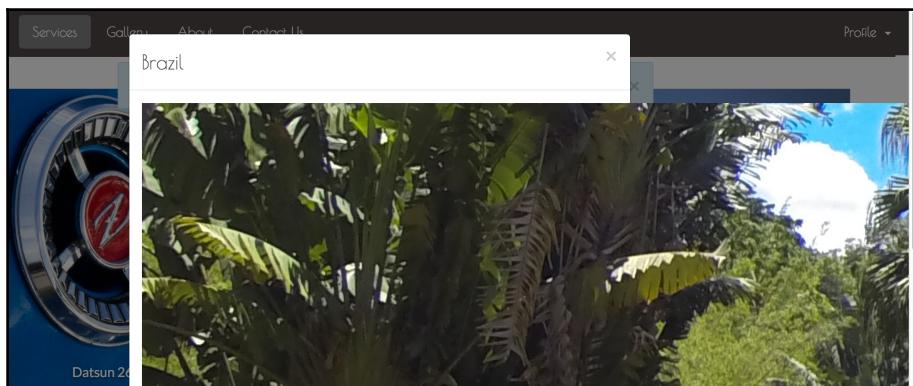


Figure 6.6: Our modal showing an enlarged version of a slide image

The modal is now surfacing. The dynamic title is working well, too. The dynamic image is getting applied and loaded. Perfect, except that the entire thing looks terrible. But that's nothing a bit of CSS can't fix.

Extending carousel's style sheets

Thanks to our forward thinking, we already have the `carousel-modal` class applied to the modal parent element. We just need to set some rules.

As this modal is directly related to our `carousel` plugin extension, we will create a CSS file explicitly for handling styling born out of our extension. Create `styles/carousel.css` and include the file in our page:

```
<link rel="stylesheet" href="bower_components/components-font-awesome/css/font-awesome.min.css" />
<link rel="stylesheet" href="styles/alert.css" />
<link rel="stylesheet" href="styles/carousel.css" />
```

There are two things wrong that we need to address. First, the modal is too narrow. We want it to be almost the full width of the page. Observe the following code:

```
.carousel-modal.modal-dialog {
    width: 95%;
}
```

Now, if an element has the `modal-dialog` class and its parent has the `carousel-modal` class, it will have have 95% of the available horizontal screen real estate. Take a look at the following screenshot:

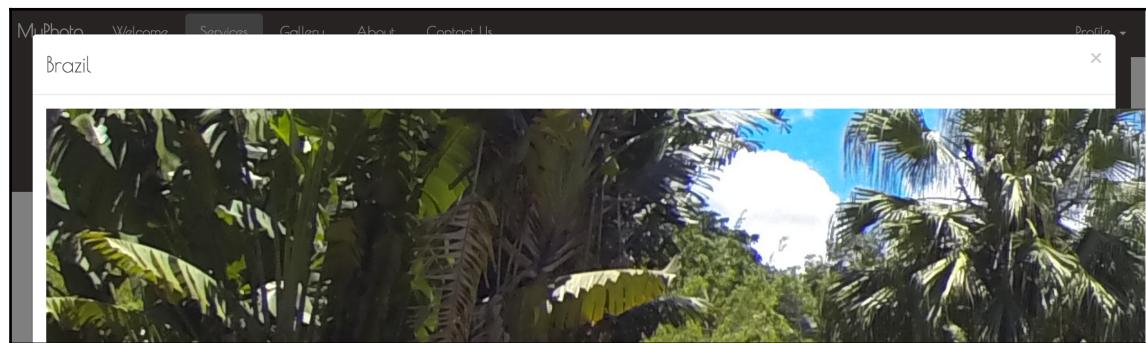


Figure 6.7: Our modal showing an enlarged version of a slide image with applied changes

Now, we just need to make sure the image doesn't breech the borders of the modal. The fix here is simple. We will just give any `img` element that is a descendent of a `carousel-modal` element a width of 100%, so that it will only take up the width explicitly available to it. Observe the following code:

```
.carousel-modal img {  
    width: 100%;  
}
```

Take a look at the following screenshot:

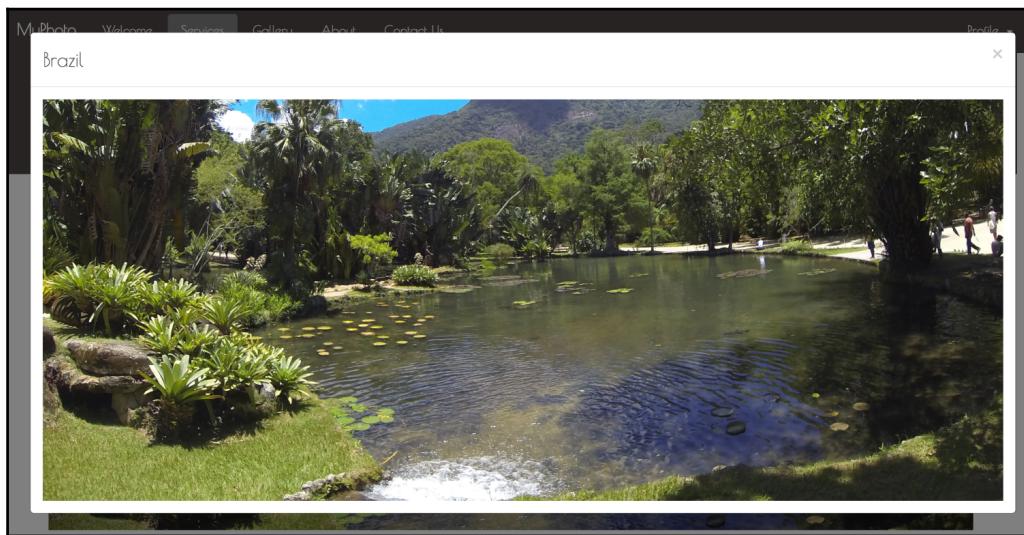


Figure 6.8: Our modal showing an enlarged version of a slide image, with the image fitting the modal

Much better. Our customization is complete. We have surfaced a modal on a `click` event from the carousel, passed data from the `carousel` component into the `modal` component, and rendered the image successfully. Very neat.

Now that we have successfully customized two of Bootstrap's jQuery plugins, let's build a plugin from scratch.

Writing a custom Bootstrap jQuery plugin

Following the patterns that we have seen in `alert.js` and `carousel.js`, we are going to build our own plugin. Of course, before we start coding, we need to understand what we want to build.

The idea – the A11yHCM plugin

The A11yHCM plugin, depending on your background and experience, may give you a clue about what we want to build. **A11y** is the accepted shorthand for **Accessibility**, or **Web Accessibility**. W3C defines web accessibility as follows:

“Web accessibility means that people with disabilities can use the web. More specifically, web accessibility means that people with disabilities can perceive, understand, navigate, and interact with the web, and that they can contribute to the web. Web accessibility also benefits others, including older people with changing abilities due to aging.”

— <https://www.w3.org/WAI/intro/accessibility.php>

HCM is an acronym for an accessibility-related term: **High Contrast Mode**. HCM, in its simplest form, modifies the colors on a display to help visually impaired users view content.

However, different tools for enabling HCM may render differently, and some web pages may not actually improve the experience of a visually impaired user in HCM. Let's take a look at a couple of examples of MyPhoto in HCM, using two different tools.

First, we will use Mac OS X El Capitan's built-in high contrast options, **Invert colors** and **Increase contrast**:

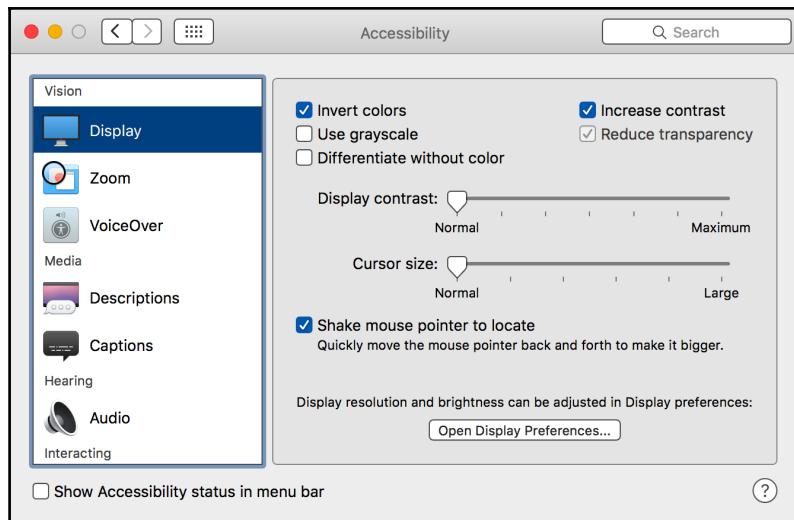


Figure 6.9: Mac OS X El Capitan's built-in Accessibility settings

With the **Invert colors** and **Increase contrast** settings enabled, MyPhoto is displayed quite differently:



Figure 6.10: Viewing MyPhoto with the Invert Colors and Increase Contrast Accessibility settings enabled on Mac OS X El Capitan, with the About navbar link in a focused state

Let's take a look at another tool—Google Accessibility's High Contrast Chrome plugin (<https://chrome.google.com/webstore/detail/high-contrast/djcfdncoelnlbldjfhinnj1hdjlikmph?hl=en>).

With the High Contrast Chrome plugin installed, a high contrast button is added to Google Chrome:



Figure 6.11: The High Contrast button in Chrome, which appears after installing Google Accessibility's High Contrast Chrome plugin

Let's see how MyPhoto displays with this High Contrast plugin enabled:

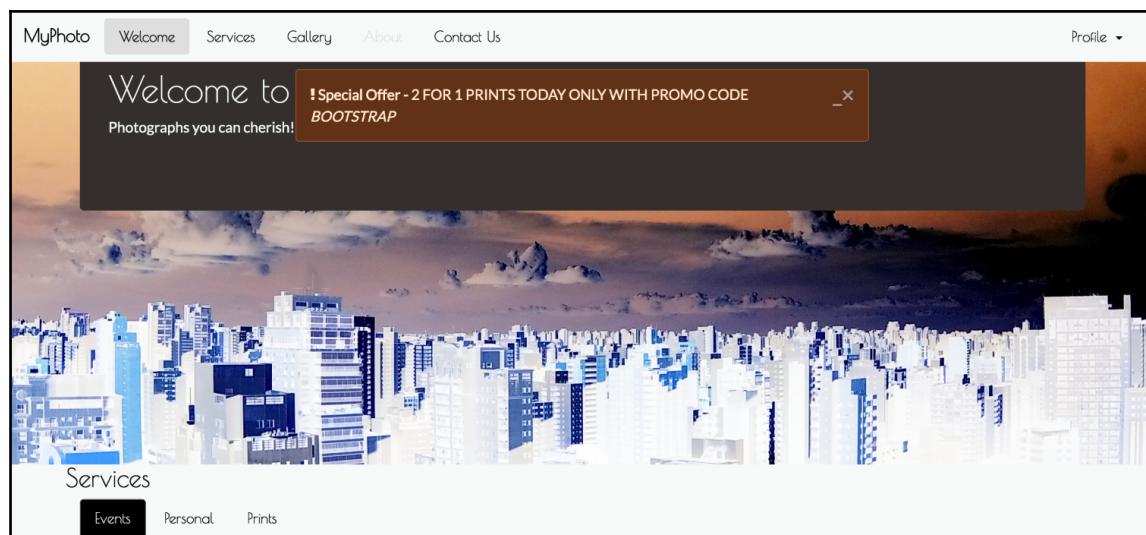


Figure 6.12: Viewing MyPhoto using Google's Accessibility High Contrast Plugin, with the 'About' navbar link in a focused state

The result of using the High Contrast Chrome plugin is similar to OS X's built-in high contrast options, but there are slight differences. For instance, the background color of our **Services** section and our special offers alert changes between the different tools.

At least one element of our page is clearly poor when displayed in HCM—the focused state of our navigation links. In HCM, the focused state of our navigation links, in this case the **Welcome** link, is practically illegible.

So what can we do here, without actually changing our default styles? The first thing that comes to mind is to simply check whether the display is in HCM and apply the appropriate styling, either through JavaScript or CSS. Unfortunately, high contrast may be applied in different ways by different tools, different browsers, different operating systems, and different devices. Programmatically, it may be very hard, or even impossible, to always recognize when your page is being displayed in HCM.

Time for plan B. Rather than figure out programmatically when a page is being viewed in high contrast, let's allow the user to simply tell us when they are viewing the page in HCM. Plan B is what our A11yHCM plugin is going to allow us to do.

The a11yHCM.js file

The first thing we are going to do is define the API. Before building a plugin, it is wise to figure out how you want developers to interact with it. By doing this, you will be able to understand exactly what you are trying to build before writing a single line of code.

The fundamental functionality of our plugin is to enable a style sheet to be dynamically loaded and removed from a page based on the occurrence of a specific event. So, we need two things:

- An event to trigger the JS
- The path to the style sheet that is to be loaded

We can use a `data-attribute` to act as the trigger, just as we did with the other plugins, and use it to pass the path of a CSS file. We want the `data-attribute` to be unique to our plugin, so we must use an appropriate `data-attribute` postfix. Let's try something like the following:

```
<div class="allyhcm" data-a11yhcm="path/to/css">High Contrast Mode</div>
```

Nice and succinct. There isn't anything else we need right now, so we will get to writing our JavaScript. Our plugin's JavaScript code is going to live in `js/a11yhcm.js`. Let's set up our skeleton; just like we have seen before, we want it immediately invoked and added to our page's `jQuery` object. We want to create an on-click listener for any element with the `data-a11yhcm` attribute, but we need to declare which function it triggers. As we want this plugin to dynamically load and remove a style sheet, we will call our function `toggle` as it toggles HCM on and off. We also want to add a `VERSION` constant:

```
+function ($) {  
    'use strict';
```

```
// A11YHCM CLASS DEFINITION
// =====
var A11yHCM = function (element, options) {
  $(element).on('click', '[data-a11yhcm]', this.toggle)
}
A11yHCM.VERSION = '1.0.0'
}(jQuery);
```

Next, we want to add the plugin definition. As explained earlier, the plugin definition creates an instance of the plugin for each DOM element with the `a11yhcm` class. Observe the following code:

```
// A11YHCM PLUGIN DEFINITION
// =====
function Plugin(option) {
  return this.each(function () {
    var $this = $(this)
    var data = $this.data('bs.a11yhcm')
    if (!data) $this.data('bs.a11yhcm', (data = new A11yHCM(this)))
    if (typeof option == 'string') data[option].call($this)
  })
}
var old = $.fn.a11yhcm
$.fn.a11yhcm = Plugin
$.fn.a11yhcm.Constructor = A11yHCM
```

We'd better not forget the `noConflict` function to help resolve namespace collisions.

```
// A11YHCM NO CONFLICT
// =====
$.fn.a11yhcm.noConflict = function () {
  $.fn.a11yhcm = old
  return this
}
```

Now we're getting to the fun part. Before we get into coding the functionality, we must declare our API. We know we want to use the `data-a11yhcm` attribute as our trigger (and to pass data to our plugin), and to use the `toggle` function that we declared in the constructor. Observe the following code:

```
// A11YHCM DATA-API
// =====
$(document).on('click.bs.allyhcm.data-api', '[data-a11yhcm]',
  A11yHCM.prototype.toggle)
```

We also want to make sure our plugin definition is called for all elements with the `data-a11yhc`m attributes. Add this to the data API:

```
$(window).on('load', function () {
    $('[data-a11yhc]').each(function () {
        var $a11yhc = $(this)
        Plugin.call($a11yhc, $a11yhc.data())
    })
})
```

Okay, now all we need to do is write the `toggle` function! Let's discuss our approach. The first thing we need to do is get the reference to the style sheet to be loaded from the `data-a11yhc`m attribute. Observe the following code:

```
A11yHCM.prototype.toggle = function (e) {
    var $this = $(this)
    var styleSheet = $this.attr('data-a11yhc')
}
```

Easy. Then, we need to figure out the current state. Are we in HCM or not? We could separate the functionality into on and off functions, hiding and showing the options in the UI as appropriate, much like our alert expand and minimize customizations. But let's try to keep the API and DOM manipulation to a minimum. Instead, we can simply check to see whether the `link` tag with the high contrast style sheet is present in the DOM. To do that, we need a way of being able to select the `link` tag. We will do this by adding the `link` tag with a unique id—`bs-a11yhc`m. Let's update `toggle` with a check to see if the element exists. If it does, use jQuery to remove it; if it doesn't, we will use jQuery to append it to the head of the DOM:

```
if(document.getElementById('bs-a11yhc'))
    $('#' + $this.styleSheetID).remove()
else {
    var styleSheetLink = '<link href="' + styleSheet + '"'
    rel="stylesheet" id="bs-a11yhc"/>'
    $('head').append(styleSheetLink)
}
```

That is pretty much it! Let's do one more thing. What if, by some chance, there is already another element on the page, unrelated to A11yHCM, with the `id` value of `bs-a11yhc`m? Rather than forcing a developer to change their page to suit the plugin, we will do the right thing and allow the developer to pass in a custom value for the `id`. The `toggle` function will check to see if an `a11yhc-id` attribute exists; if it does, A11yHCM will use that value as the `id` for the `link` tag. In that case, an element using A11yHCM could look like:

```
<div class="allyhc" data-a11yhc="path/to/css" a11yhc-
```

```
id="customId">High Contrast Mode</div>
```

Let's update the `toggle` function to reflect this. We will add the default value for the `id` as a property of A11yHCM:

```
var A11yHCM = function (element) {
    this.$element = $(element)
}
A11yHCM.VERSION = '1.0.0'
A11yHCM.DEFAULTS = {
    styleSheetID : 'bs-a11yhcm'
}
A11yHCM.prototype.toggle = function (e) {
    var $this = $(this)
    var styleSheet = $this.attr('data-a11yhcm')
    if ($this.attr('a11yhcm-id'))
        $this.styleSheetID = $this.attr('a11yhcm-id')
    else
        $this.styleSheetID = A11yHCM.DEFAULTS.styleSheetID
    if (document.getElementById($this.styleSheetID))
        $('#' + $this.styleSheetID).remove()
    else {
        var styleSheetLink = '<link href="' + styleSheet + '"'
        rel="stylesheet" id="' + $this.styleSheetID + '">'
        $('head').append(styleSheetLink)
    }
}
```

Okay, that's it. That looks like all the JavaScript we're going to need to make A11yHCM work the way we envisaged. Now, let's put it into practice by adding the markup.

The markup

The first thing we have to do is make sure the JavaScript for the A11yHCM plugin is loaded. Include the JS file in the head of the page, after `bootstrap.min.js` and `jquery.min.js`. Observe the following code:

```
<script src="bower_components/jquery/dist/jquery.min.js">
</script>
<script src="bower_components/bootstrap/dist/js/bootstrap.min.js">
</script>
<script src="js/alert.js"></script>
<script src="js/carousel.js"></script>
<script src="js/a11yhcm.js"></script>
```

Let's add the new **High Contrast Mode** option to the **Profile** drop-down. We will also include a contrast icon from Font Awesome, and we want to load `styles/myphoto-hcm.css`:

```
<li class="nav-item dropdown pull-xs-right">
    <a href="#" class="nav-link dropdown-toggle" data-toggle="dropdown" role="button"
        aria-haspopup="true" aria-expanded="false">
        Profile <span class="caret"></span>
    </a>
    <div class="dropdown-menu dropdown-menu-right">
        <a class="dropdown-item" href="#" data-toggle="modal" data-target="#profile-modal">
            Profile
        </a>
        <a class="dropdown-item" href="#" data-toggle="modal" data-target="#settings-modal">
            Settings
        </a>
        <div class="dropdown-divider"></div>
        <a class="dropdown-item" href="#">
            Logout
        </a>
        <a class="dropdown-item a11yhc" href="#" data-a11yhc="styles/myphoto-hcm.css">
            <i class="fa fa-adjust"></i> High Contrast Mode
        </a>
    </div>
</li>
```

Take a look at the following screenshot:

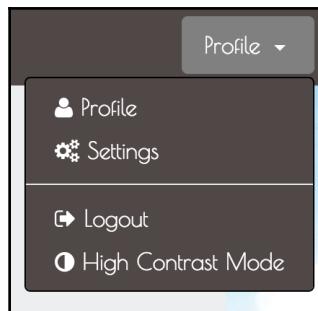


Figure 6.13: The new High Contrast Mode menu item

Great. Now, of course, when we click on **High Contrast Mode** we won't be able to see any visual changes as we haven't actually created `styles/myphoto-hcm.css`. But if we inspect the DOM, we should be able to see the CSS file referenced in the head. Take a look at the following screenshot:

```
<script src="js/carousel.js"></script>
<script src="js/a11y-hcm.js"></script>
▶<script type="text/javascript">...</script>
<link href="styles/myphoto-hcm.css" rel="stylesheet" id="bs-a11yhcm">
</head>
```

Figure 6.14: The High Contrast Mode style sheet dynamically added by the High Contrast Mode button

Click the **High Contrast Mode** button again, and the `link` element should be removed. Take a look at the following screenshot:

```
<script src="js/alert.js"></script>
<script src="js/carousel.js"></script>
<script src="js/a11y-hcm.js"></script>
▶<script type="text/javascript">...</script>
</head>
```

Figure 6.15: The High Contrast Mode style sheet dynamically removed by the High Contrast Mode button

Great. Our plugin is working. Let's pass in a non-default value to be used as the `id` for the `link` element, to make sure that is also working as expected.

```
<a href="#" class="a11yhcm" data-a11yhcm="styles/myphoto-hcm.css"
a11yhcm-id="myphoto-hcm">
```

Take a look at the following screenshot:

```
<script src="js/alert.js"></script>
<script src="js/carousel.js"></script>
<script src="js/a11y-hcm.js"></script>
▶<script type="text/javascript">...</script>
<link href="styles/myphoto-hcm.css" rel="stylesheet" id="myphoto-hcm">
</head>
```

Figure 6.16: The High Contrast Mode style sheet dynamically added by the High Contrast Mode button, with a custom id attribute

Perfect. All functionality is working just like we wanted. Now, let's get to the CSS!

Adding some style

First, we're going to write the style sheet for A11yHCM, and, for good measure, we are going to write `myphoto-hcm.css`, to make our navigation more useful in **High Contrast Mode**.

Create a `styles/a11yhcm.css` file and include it in the head of our page:

```
<link rel="stylesheet" href="bower_components/components-font-awesome/css/font-awesome.min.css" />
<link rel="stylesheet" href="styles/alert.css" />
<link rel="stylesheet" href="styles/carousel.css" />
<link rel="stylesheet" href="styles/a11yhcm.css" />
```

All we are going to do here is toggle the `a11yHCM` element to indicate whether it is disabled or enabled. We're simply going to add a checkmark when it is enabled. We will need a little bit of JavaScript too, to add and remove the `enabled` class to our element. First, let's write the CSS:

```
.a11yhcm.enabled::after {
    content: '✓'
}
```

This rule simply appends a checkmark to the end of any content within an element with both the `a11yhcm` and `enabled` class applied, using the `after` pseudo-class. Let's update `a11yHCM.js` to add and remove the `enabled` class:

```
if (document.getElementById($this.styleSheetID)) {
    $('#' + $this.styleSheetID).remove()
    $this.removeClass('enabled')
}
else {
    var styleSheetLink = '<link href="' + styleSheet + '"'
    rel="stylesheet" id="' + $this.styleSheetID + '"/>'
    $('head').append(styleSheetLink)
    $this.addClass('enabled')
}
```

Let's check it out. Click on the **High Contrast Mode** button:

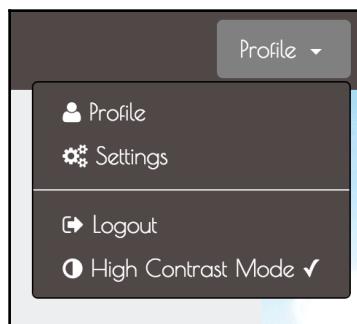


Figure 6.17: The checkmark is applied to the menu item to indicate when High Contrast Mode is enabled

Great. We now have a visual indicator for when **High Contrast Mode** is enabled. The checkmark might not be suitable for every design, but the rules can always be extended!

Now, let's get to fixing our navigation.

Create `styles/myphoto-hcm.css` and copy over the classes related to the hover, focus, and active states of the navigation from `myphoto.css`:

```
.navbar-myphoto .navbar-nav > li > a:hover {  
    background-color: #504747;  
    color: gray;  
}  
.navbar-myphoto .navbar-nav > li > a:focus {  
    background-color: #504747;  
    color: gray;  
}  
.navbar-myphoto .navbar-nav > li.active > a {  
    background-color: #504747;  
    color: gray;  
}  
.navbar-myphoto .dropdown-menu > a {  
    color: white;  
    background-color: #504747;  
}  
.navbar-myphoto .dropdown-menu > a:hover {  
    color: gray;  
    background-color: #504747;  
}  
.navbar-myphoto .dropdown-menu > a:focus {  
    color: gray;  
    background-color: #504747;  
}
```

```
.navbar-myphoto .dropdown-menu > .active > a:focus {  
    color: gray;  
    background-color: #504747;  
}
```

We want these states to be very clear in **High Contrast Mode**. For full effect, we will use the color blue. But hang on, before you go changing all the `color` properties to blue. The high contrast tools we're using are inverting the colors, so we need the inverse of blue, that is, yellow. Observe the following code:

```
.navbar-myphoto .navbar-nav > li > a:hover {  
    background-color: #504747;  
    color: yellow;  
}  
.navbar-myphoto .navbar-nav > li > a:focus {  
    background-color: #504747;  
    color: yellow;  
}  
.navbar-myphoto .navbar-nav > li.active > a {  
    background-color: #504747;  
    color: yellow;  
}  
.navbar-myphoto .dropdown-menu > li > a:hover {  
    background-color: #504747;  
    color: yellow;  
}  
.navbar-myphoto .dropdown-menu > li > a:focus {  
    background-color: #504747;  
    color: yellow;  
}  
.navbar-myphoto .dropdown-menu > li > .active > a:focus {  
    background-color: #504747;  
    color: yellow;  
}
```

Enable high contrast on your browser or OS, then click **High Contrast Mode** on the MyPhoto page to see the results. Take a look at the following screenshot:

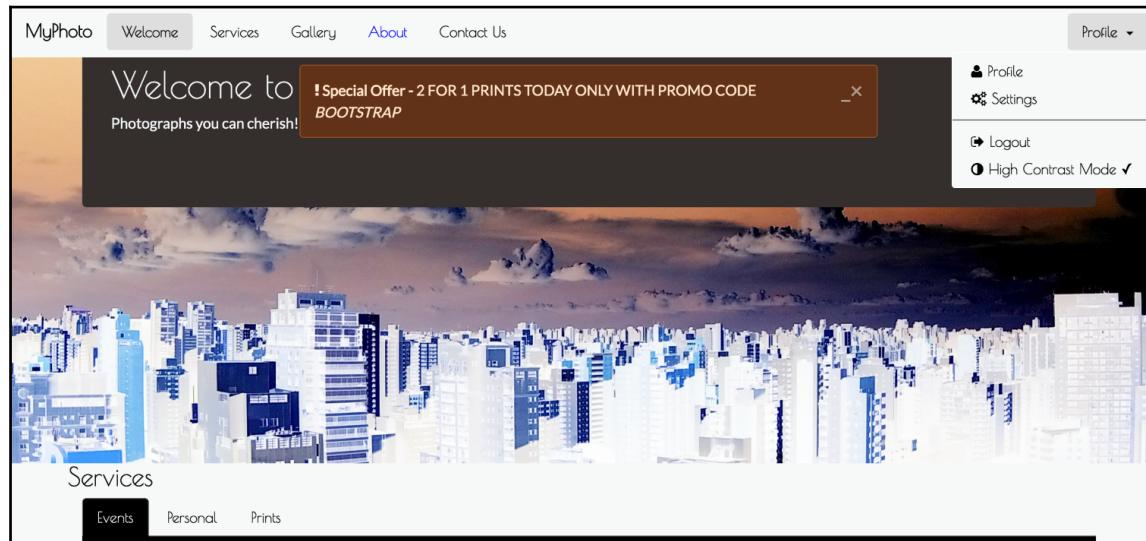


Figure 6.18: Viewing MyPhoto using Google's Accessibility High Contrast Chrome Plugin, with the 'About' navbar link in a focused state and with MyPhoto's High Contrast Mode enabled

The UI is now much clearer, and all thanks to our custom-built A11yHCM jQuery plugin. There is a lot more to do to make this page fully accessible, but it's a start.

Summary

In this chapter, we have discovered quite a bit about Bootstrap's jQuery plugins but, most importantly, we put what we learned into practice. We now have an understanding of the structure of a plugin, from its JavaScript to its CSS and HTML.

Using what we learned, we extended the alert plugin, adding the ability to minimize and expand it together with a simple API. We then leveraged the modal plugin to extend the carousel plugin to surface larger versions of carousel images. Again, we achieved this through a simple API .

We put everything that we learned from building the alert and carousel plugin together to build our very own custom plugin, A11yHCM. A11yHCM dynamically loads CSS at the user's command. Putting this use case in the context of a visually impaired user, we were able to understand how useful a plugin like this might be.

In the following chapter, we will discover how to integrate some popular third-party plugins and libraries into Bootstrap to enhance the overall user experience of MyPhoto.

7

Integrating Bootstrap with Third-Party Plugins

In this chapter, we are going to discover how to integrate some popular plugins and libraries into Bootstrap to enhance the user experience of `MyPhoto`.

We will identify new features or improvements we want to make to `MyPhoto`, introduce libraries to help us achieve those goals, and figure out how these can be gracefully integrated within our existing architecture.

This chapter will focus on the integration of three popular plugins and libraries, namely, `Salvattore`, `Animate.css`, and `Hover`. These libraries will allow us to improve the overall user experience of `MyPhoto`, creating a much more polished look and feel.

To summarize, this chapter will help us:

- Learn how to leverage `Salvattore` to add more flexibility to Bootstrap's grid system
- Learn how to leverage `Animate.css` to easily add CSS3 animations to a Bootstrap page
- Learn how to leverage `Hover` to easily add hover effects to a Bootstrap page

Building a testimonial component with Salvattore

MyPhoto does a good job of boasting about the services on offer and the quality of those services. However, a user may want to read some feedback from the previous customers. To do this, we're going to add a `testimonials` component to our page as a new tab in the `services` component. The **Testimonials** tab will display some positive feedback from the users. Let's add that new tab to our list of tabs:

```
<li class="nav-item">
    <a href="#services-testimonials" class="nav-link"
       data-toggle="tab">Testimonials</a>
</li>
```

Next, we want to add some content. Let's use Bootstrap's grid system to display the testimonials in four neat rows of three columns. Add the following code below the `services-prints` markup:

```
<div role="tabpanel" class="tab-pane" id="services-testimonials">
    <div class="container">
        <div class="row myphoto-testimonial-row">
            <div class="col-xs-3 myphoto-testimonial-column">
                <h6>Debbie</h6>
                <p>Great service! Would recommend to friends!</p>
            </div>
            <div class="col-xs-3 myphoto-testimonial-column">
                <h6>Joey</h6>
                <p>5 stars! Thanks for the great photos!</p>
            </div>
            <div class="col-xs-3 myphoto-testimonial-column">
                <h6>Jack & Jill</h6>
                <p>So happy with how the photos turned out!
                    Thanks for capturing the memories of our day!</p>
            </div>
            <div class="col-xs-3 myphoto-testimonial-column">
                <h6>Tony</h6>
                <p>Captured our Cup final win! Great stuff!</p>
            </div>
        </div>
        <div class="row myphoto-testimonial-row">
            <div class="col-xs-3 myphoto-testimonial-column">
                <h6>Anne</h6>
                <p>Really high quality prints!</p>
            </div>
            <div class="col-xs-3 myphoto-testimonial-column">
```

```
<h6>Mary</h6>
<p>Made a stressful event much easier!
Absolute professionals!</p>
</div>
</div>
<div class="row myphoto-testimonial-row">
    <div class="col-xs-3 myphoto-testimonial-column">
        <h6>Oscar</h6>
        <p>Declared their greatness, exhibited greatness.</p>
    </div>
    <div class="col-xs-3 myphoto-testimonial-column">
        <h6>Alice</h6>
        <p>Wonderful! Exactly as I imagined they would
turn out!
</p>
    </div>
    <div class="col-xs-3 myphoto-testimonial-column">
        <h6>Nick</h6>
        <p>Perfectly captured the mood of our gig.
Top notch.</p>
    </div>
</div>
</div>
</div>
```

We have added three rows with varying amounts of columns. Each column includes the name of a user and the associated testimonial. As we want a maximum of four columns in a row, we have given each column the `col-xs-3` class so that the column takes up three of the 12 columns in the Bootstrap grid system. We have also given each column an additional `myphoto-testimonial-column` class for specific styling, and each row a `myphoto-testimonial-row` class. Add the following rules to `myphoto.css`:

```
.myphoto-testimonial-row {
    margin-right : 5px;
}
.myphoto-testimonial-column
{
    border: 1px solid black; background-color: #FFFFFF;
    padding: 10px;
    margin: 5px;
    max-width: 23%;
}
```

We have given some extra spacing to our testimonials. To make up for the extra spacing, we set a `max-width` property of 23% as opposed to the 25% declared by `col-xs-3`, and overrode the default `margin-right` property of the `row` class of `-15px` to `5px`. We also included a black border and solid white background. Let's check out our results:

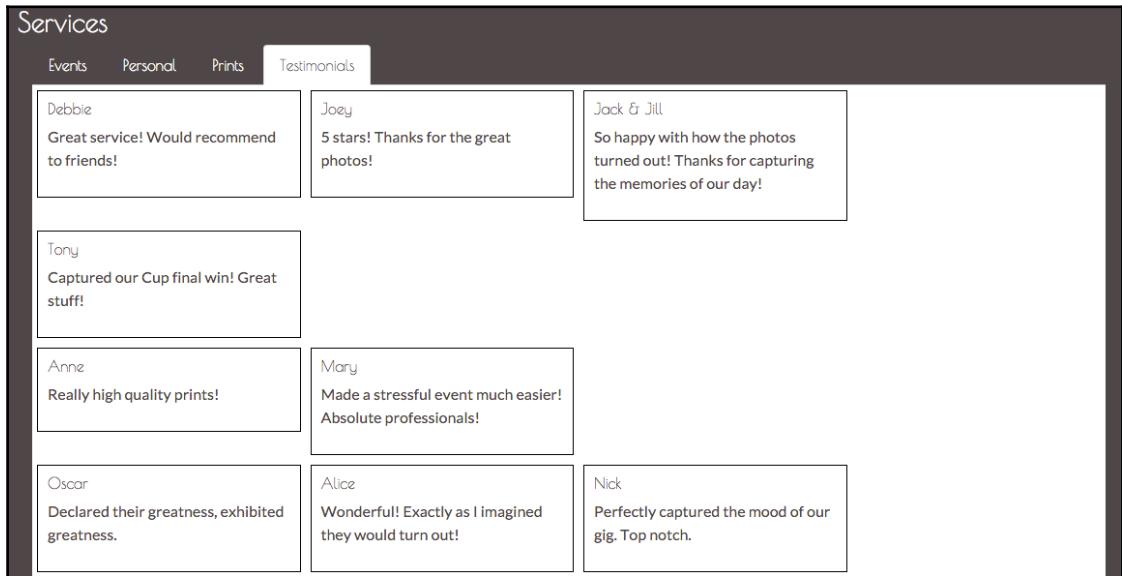


Figure 7.1: A Testimonials tab is created using Bootstrap's grid system

Okay, cool. We have our **Testimonials** tab and our positive testimonies from users. However, it looks kind of ugly. The contents of the columns are of varying lengths, which makes the spacing visually awkward. Take the distance between Joey's testimony on the first row and Mary's on the second. Wouldn't it be nice if we could decrease this distance? From column to column across individual rows, the spacing is uniform and looks good. Wouldn't it be nice if we could achieve this uniformity vertically, too? Unfortunately, Bootstrap's grid system is not flexible enough to make the vertical spacing uniform while allowing for varying heights of columns. However, we can achieve this with a popular library called Salvattore.

Introducing Salvattore

Salvattore (<http://salvattore.com/>) is a JavaScript library that allows for more flexible grid system definitions. For instance, with the help of Salvattore, we can define grid system rules that will allow three rows of four columns, but the position of each column is relative to the height of the corresponding column in the preceding row. Salvattore is a pure JavaScript library, meaning that it doesn't depend on any other library (such as jQuery) to work.

Let's add Salvattore to our project. Download Salvattore using Bower:

```
bower install salvattore
```

Then, make sure to include it in our markup, at the bottom of the page. Including the JavaScript file at the bottom of our page is the suggested practice by Salvattore as it needs the DOM to be loaded before it takes effect:

```
<footer class="footer text-center">
    <p class="text-muted">
        <small>&copy; MyPhoto Inc.</small>
    </p>
    <p class="text-muted text-center">
        <a href="#">
            <small>Terms & Conditions</small>
        </a>
    </p>
    <p class="text-muted">
        <a href="#">
            <small>About Us</small>
        </a>
    </p>
</footer>
<script src="bower_components/salvattore/dist/
salvattore.min.js"></script>
```

Next, we need to rewrite our `testimonials` grid to work with Salvattore. One big difference between how we constructed the grid initially and how we need to construct it for Salvattore, is that we won't be grouping the testimonials by row; Salvattore will do the grouping for us.

Currently, we have the following:

```
<div class="row myphoto-testimonial-row">
    <div class="col-xs-3 myphoto-testimonial-column">
        <h6>Anne</h6>
        <p>Really high quality prints!</p>
```

```
</div>
<div class="col-xs-3 myphoto-testimonial-column">
    <h6>Mary</h6>
    <p>Made a stressful event much easier! Absolute
        professionals!</p>
</div>
</div>
```

Here, we are creating a row, and then constructing the columns within that row. That is not how Salvattore works. Let's rewrite our grid. This time, we will simply be listing the testimonials:

```
<div role="tabpanel" class="tab-pane" id="services-testimonials">
    <div class="container">
        <div class="myphoto-testimonial-grid" data-columns>
            <div>
                <h6>Debbie</h6>
                <p>Great service! Would recommend to friends!</p>
            </div>
            <div>
                <h6>Anne</h6>
                <p>Really high quality prints!</p>
            </div>
            <div>
                <h6>Oscar</h6>
                <p>Declared their greatness, exhibited greatness.</p>
            </div>
            <div>
                <h6>Joey</h6>
                <p>5 stars! Thanks for the great photos!</p>
            </div>
            <div>
                <h6>Mary</h6>
                <p>Made a stressful event much easier! Absolute
                    professionals!</p>
            </div>
            <div>
                <h6>Alice</h6>
                <p>Wonderful! Exactly as I imagined they would
                    turn out!</p>
            </div>
            <div>
                <h6>Jack & Jill</h6>
                <p>So happy with how the photos turned out! Thanks for
                    capturing the memories of our day!</p>
            </div>
            <div>
                <h6>Nick</h6>
```

```
<p>Perfectly captured the mood of our gig.  
Top notch.</p>  
</div>  
<div>  
    <h6>Tony</h6>  
    <p>Captured our Cup final win! Great stuff!</p>  
</div>  
</div>  
</div>  
</div>
```

Okay, so that's quite a change. Let's talk through the important pieces. As mentioned earlier, we are now simply listing testimonials rather than grouping them by row. We no longer have our `row` wrapper elements. We have added a parent `div` wrapper around all the testimonials, with the `myphoto-testimonial-grid` class. As we are no longer referencing Bootstrap's grid system, each column entry no longer has the `col-xs-3` class and we have also removed the `myphoto-testimonial-column` classes. The final thing we need to do with the markup is add a `data-columns` attribute to the parent `grid` element. Observe the following code snippet:

```
<div id="myphoto-testimonial-grid" data-columns>
```

The `data-columns` attribute is the hook Salvattore needs to know which elements to act upon. All we need to do now is add some CSS.

Open up `myphoto.css` and add the following:

```
.myphoto-testimonials-grid[data-columns]::before {  
    content: '4 .column.size-1of4';  
}  
.column {  
    float: left;  
}  
.size-1of4 {  
    width: 25%;  
}
```

So, what's happening here? First, we are using the `before` pseudo-selector to define how many columns we want and the classes we want applied. In this case, we want four columns and to apply the `column` class and the `size-1of4` class. The `column` class makes sure any column elements are rendered as close to the left of the element's container element as possible, and any elements with the `size-1of4` class are to only take up a quarter of the available space. If we planned on only having three columns, then we could define and apply a `size-1of3` class with a rule to only take up 33.333% of the available space, and so on. Salvattore will split the testimonials into four groups, as evenly as

possible, and apply the classes defined by `.myphoto-testimonials-grid[data-columns]::before` to the column groups. For instance, in our example, the first group of columns has the following markup:

```
<div class="myphoto-testimonial-grid" data-columns="4">
  <div class="column size-1of4">
    <h6>Debbie</h6>
    <p>Great service! Would recommend to friends!</p>
  </div>
  ...
  <div>
    <h6>Mary</h6>
    <p>Made a stressful event much easier! Absolute
      professionals!</p>
  </div>
  <div>
    <h6>Tony</h6>
    <p>Captured our Cup final win! Great stuff!</p>
  </div>
</div>
...
```

Let's see how this renders:

The screenshot shows a web page titled "Services" with a navigation bar containing "Events", "Personal", "Prints", and "Testimonials". The "Testimonials" tab is active, indicated by a highlighted background. Below the navigation, there are four columns of testimonial data. The first column contains Debbie's review: "Great service! Would recommend to friends!" followed by "Really high quality prints!". The second column contains Mary's review: "Made a stressful event much easier! they would turn out! Absolute professionals!" followed by "Wonderful! Exactly as I imagined". The third column contains Oscar's review: "Declared their greatness, exhibited greatness." followed by "5 stars! Thanks for the great photos!". The fourth column contains Tony's review: "Captured our Cup final win! Great stuff!" followed by "Perfectly captured the mood of our So happy with how the photos turned gig. Top notch. out! Thanks for capturing the memories of our day!". The entire testimonial section is contained within a white rectangular box.

Figure 7.2: The unstyled testimonials, created using Salvattore

Great! Now all we need to do is add some styling!

The first thing that we want to do is create some space between the top and the content. As you can see, the top of the panel is very close to the list tab. We have a parent class for our grid, `myphoto-testimonial-grid`, that we can leverage. Add the following style rules:

```
.myphoto-testimonial-grid {  
    padding-top : 30px ;  
}
```

Take a look at the following screenshot:

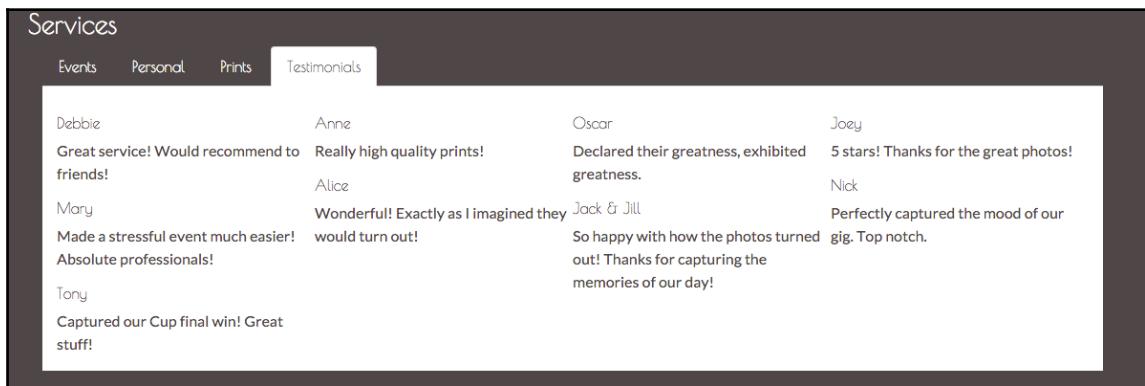


Figure 7.3: The unstyled testimonials, created using Salvattore with a top padding of 30px

Next up, we want to make the testimonials distinguishable. For this, we'll leverage the `myphoto-testimonial-column`. Remove any previous rules for this class and add the following:

```
.myphoto-testimonial-column {  
    padding: 10px;  
    border: 1px solid #000000;  
    margin: 5px;  
}
```

We have simply created a border around the element, added an internal spacing of 10px between the element contents and its border, and lastly, we have created a margin of 5px between each of the elements. Add the `myphoto-testimonial-column` class to each of our testimonial elements:

```
<div class="myphoto-testimonial-grid" data-columns>  
    <div class="myphoto-testimonial-column">  
        <h6>Debbie</h6>  
        <p>Great service! Would recommend to friends!</p>  
    </div>
```

```
<div class="myphoto-testimonial-column">
  <h6>Anne</h6>
  <p>Really high quality prints!</p>
</div>
<div class="myphoto-testimonial-column">
  <h6>Oscar</h6>
  <p>Declared their greatness, exhibited greatness.</p>
</div>
<div class="myphoto-testimonial-column">
  <h6>Joey</h6>
  <p>5 stars! Thanks for the great photos!</p>
</div>
<div class="myphoto-testimonial-column">
  <h6>Mary</h6>
  <p>Made a stressful event much easier! Absolute
professionals!</p>
</div>
<div class="myphoto-testimonial-column">
  <h6>Alice</h6>
  <p>Wonderful! Exactly as I imagined they would
turn out!</p>
</div>
<div class="myphoto-testimonial-column">
  <h6>Jack & Jill</h6>
  <p>So happy with how the photos turned out! Thanks
for capturing the memories of our day!</p>
</div>
<div class="myphoto-testimonial-column">
  <h6>Nick</h6>
  <p>Perfectly captured the mood of our gig. Top notch.</p>
</div>
<div class="myphoto-testimonial-column">
  <h6>Tony</h6>
  <p>Captured our Cup final win! Great stuff!</p>
</div>
</div>
```

That's a little better. Now we can at least differentiate each of the testimonials from one another (see *Figure 7.4*). Compare this with the `testimonial` component we built using just Bootstrap's grid system; the visual improvements are obvious. We have a much more natural, flowing visual now, compared to the clunky, almost awkward, original visual.

Now, let's see how we can leverage Bootstrap's grid system to ease the use of Salvattore.

Take a look at the following screenshot:

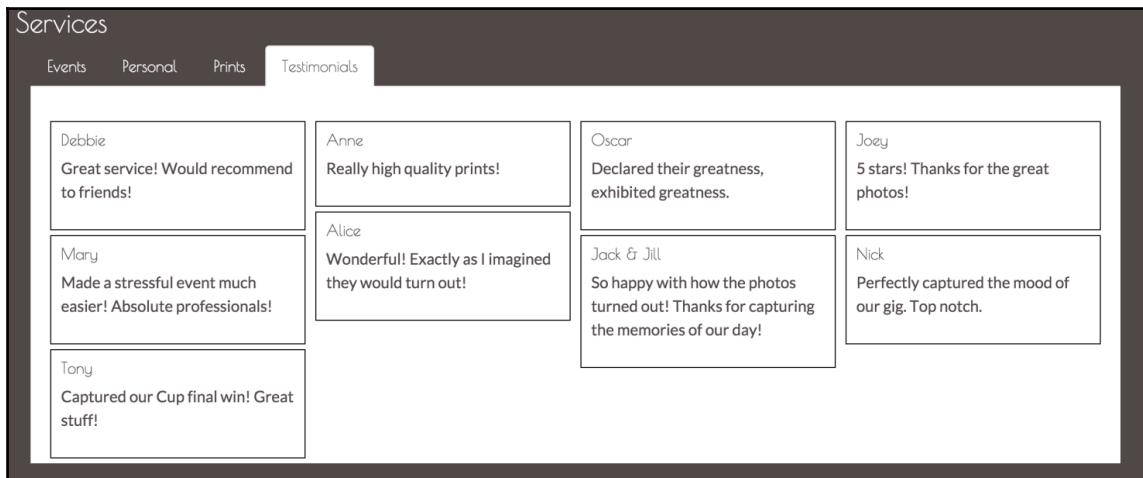


Figure 7.4: The testimonials arranged using Salvattore and with some custom styling

Integrating Salvattore with Bootstrap

In our example, we created two new classes to lay out our Salvattore grid, namely, `column`, which just applies `float: left`, and `size-1of4`, which just sets a width of 25% on the element. Doesn't Bootstrap already provide these classes? Of course it does.

In fact, our `col-***-***` classes all have the `float: left` property, and they all manage the width of our column. Let's use these classes instead!

First, we need to figure out which `col-***-***` class we want to use. We always want four horizontal columns with a width of 25%. Bootstrap's grid system is split into 12, so we want each of our columns to take up three column spaces; `col-xs-3` fulfills those requirements. Let's rewrite the `.myphoto-testimonial-grid[data-columns]`:

```
.myphoto-testimonial-grid[data-columns]::before {  
    content: '4 .col-xs-3';  
}
```

We have replaced the classes to be applied to the column groups with Bootstrap's `col-xs-3` class. Let's take a look at the results (see *Figure 7.5*):

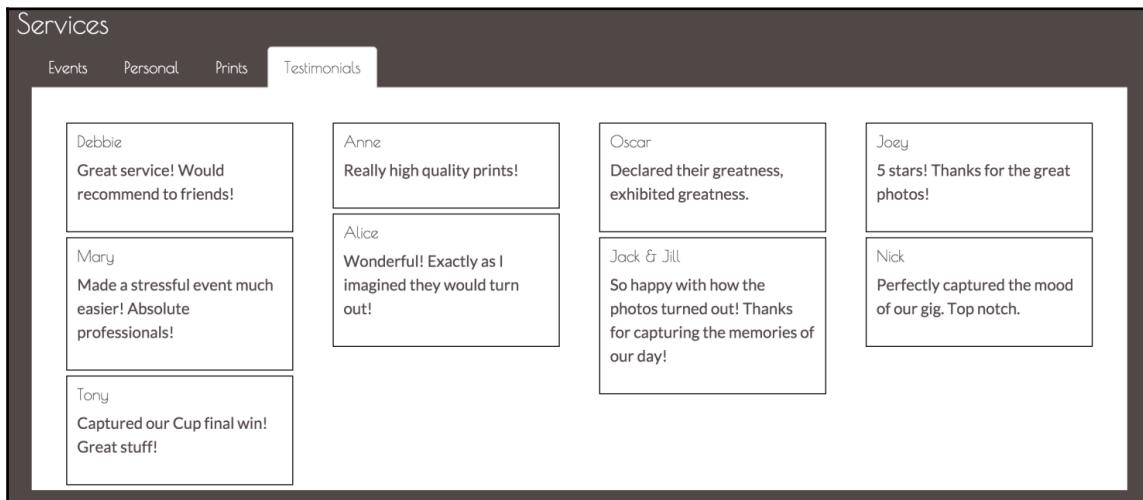


Figure 7.5: Using Bootstrap's `col-**-**` class to render the testimonials grid (note the additional padding)

Our `col-xs-3` class has added some extra padding that we don't really want in this scenario. Let's create a new class, `myphoto-testimonial-group`, to override the padding:

```
.myphoto-testimonial-group {  
    padding: 0px;  
}
```

We also need to make sure that the following class is applied to the columns groups:

```
.myphoto-testimonial-grid[data-columns]::before {  
    content: '4 .col-xs-3.myphoto-testimonial-group';  
}
```

Take a look at the following screenshot:

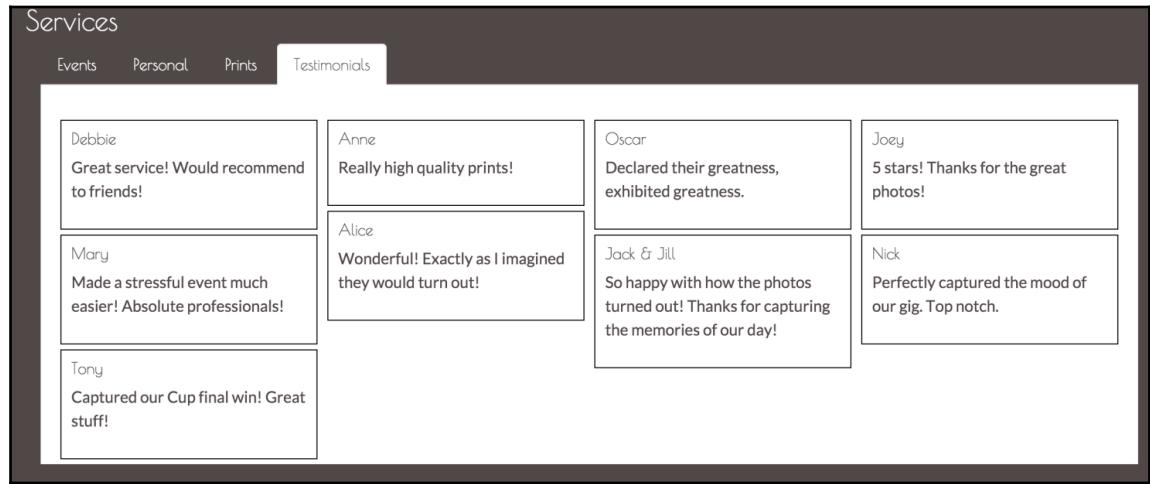


Figure 7.6: Using Bootstrap's col-**-** class together with the myphoto-testimonial-grid class to render the testimonials grid (note the reduced padding)

The content property



The content property is used to add content to a given element. For example, the rule `span::before {content: 'Hello World';}` will insert the text "Hello World" before the content within any `span` element on the page. Similarly, the rule `span::before {content: 'Hello World';}` inserts the text "Hello World" after the content within any `span` element on the page. It is important to note that the `content` property can only be used in conjunction with either the `after` or `before` selector.

Great! Our Salvattore grid is now powered by Bootstrap's grid system! This is quite important, as we now know how to use the flexibility and simplicity of Salvattore's grid system without having to worry about all the nitty-gritty work Bootstrap solves with its own grid system.

Let's animate. Animate.css (<https://daneden.github.io/animate.css/>) is a cross-browser, pure CSS library that provides a huge number of easy-to-use classes to add animations to a page. That is, the library has zero dependencies and can be layered upon any web page.

Adding Animate.css to MyPhoto

As it's a pure CSS library, integrating Animate.css with MyPhoto is exceedingly simple. First, use Bower to download Animate.css:

```
bower install animate.css
```

Next, include the library in the head of MyPhoto, after the other CSS includes:

```
<link rel="stylesheet"
      href="bower_components/bootstrap/dist/css/bootstrap.min.css" />
<link rel="stylesheet" href="styles/myphoto.css" />
...
<link rel="stylesheet"
      href="bower_components/DataTables/media/css/dataTables.
      bootstrap.min.css" />
<link rel="stylesheet"
      href="bower_components/animate.css/animate.min.css" />
```

To make sure we are set up correctly, let's add an animation to the body of our web page. To use any of the classes provided by Animate.css, we must include the `animated` class on the element:

```
<body data-spy="scroll" data-target=".navbar" class="animated">
```

If we take a look at `animate.min.css`, we will see that all selectors depend on the `animated` class to be defined and the `animated` class itself provides the base animation rules:

```
.animated {
    -webkit-animation-duration: 1s;
    animation-duration: 1s;
    -webkit-animation-fill-mode: both;
    animation-fill-mode: both
}
.animated.hinge {
    -webkit-animation-duration: 2s;
    animation-duration: 2s
}
```

To apply the animation itself is just as simple. Add the desired class to the element. To make MyPhoto fade in, we just need to apply the `fadeIn` class to the `body` element:

```
<body data-spy="scroll" data-target=".navbar" class="animated
fadeIn">
```

If you fire up the page, you should see the page slowly fade in as it renders. If you do not see this animation, make sure you have the correct path to `animate.min.css` in the head.

Now that we have Animate.css added to our project and working, let's add some emphasis to some of the core elements in our site.

Bouncing alerts

MyPhoto has a special offers alert in a prominent position on the page. However, it may still not grab the attention of the user initially. We need to focus the attention of the user immediately on the alert and add some emphasis. We need to tell the user that this is something they should read.

Animate.css has a range of `bounce` classes which are great for grabbing the attention of a user. We have a selection of 11 `bounce` classes here: `bounce`, `bounceIn`, `bounceInDown`, `bounceInLeft`, `bounceInRight`, `bounceInUp`, `bounceOut`, `bounceOutDown`, `bounceOutLeft`, `bounceOutRight`, and `bounceOutUp`.

We're going to go with the `bounceIn` class, which gives a throbber-like behavior. To get the `bounceIn` effect, add the `animated` and `bounceIn` classes to the special offers alert element:

```
<div class="alert alert-info alert-position animated bounceIn">
    <a href="#" class="close" data-dismiss="alert" aria-label=
        "close">&times;</a>
    <a href="#" class="close minimize" data-minimize="alert"
        aria-label="minimize">_</a>
    <a href="#" class="close expand hide" data-expand="alert"
        aria-label="expand">+</a>
    <strong>
        <i class="fa fa-exclamation"></i> Special Offer -
    </strong>
    <span>2 FOR 1 PRINTS TODAY ONLY WITH PROMO CODE
        <span style="font-style: italic">BOOTSTRAP</span>
    </span>
</div>
```

The alert should now be animated. Animate.css also provides an infinite animation option, where the animation will be infinitely repeated. To use this option, simply add the `infinite` class to the element:

```
<div class="alert alert-info alert-position animated bounceIn
infinite">
```

Now, the `bounceIn` effect will be infinitely repeated. However, it does not look great as the element actually transitions from hidden to visible during the animation, so this transition is also looped. This is a great example of how the `infinite` class does not play well with all animations. A more infinite-friendly animation is `pulse`, which offers a similar effect. Replace the `bounceIn` class with the `pulse` class:

```
<div class="alert alert-info alert-position animated pulse infinite">
```

Great, the special offers alert now animates infinitely, with a gentle pulse to gain the attention of the user. However, if we want this applied to all alerts, then we need to apply it to all alerts individually. In this scenario, if there is a design change and, for instance, we want to change the animation, we would again have to change them all individually. Let's manage it centrally, instead, by extending the Alert jQuery plugin by adding further customizations to the `js/alert.js` file we created in Chapter 6, *Customizing Your Plugins*.

So, what do we want to do? On page load, we want to add `animate.css` classes to our alert elements. First, let's add our `on load` listener to our IIFE. Add this directly after `$(document).on('click.bs.alert.data-api', '[data-expand="alert"]', Alert.prototype.expand)`:

```
$(window).on('load', function () {  
})
```

Next, we need a hook for our plugin. Let's use a `data-*` attribute, as is the general practice with Bootstrap jQuery plugins. We will call it `data-alert-animate`. Any element to which we want these classes applied will already have the `data-alert-animate` attribute. Our plugin will loop through each of these elements, applying the relevant classes:

```
$(window).on('load', function() {  
    $('[data-alert-animate]').each(function() {  
    })  
})
```

In our markup, let's update the special offers alert element to remove the `animate.css` classes and add the `data-alert-animate` attribute:

```
<div class="alert alert-info alert-position" data-alert-animate>
```

We want to add three classes to the `data-alert-animate` elements: `animated`, `pulse`, and `infinite`. Let's update the plugin to apply these classes to each `data-alert-animate` element:

```
$(window).on('load', function() {  
    $('[data-alert-animate]').each(function() {  
        $(this).addClass('animated pulse infinite')
```

```
    })
})
```

Now, the markup will be dynamically transformed on page load, adding the animated, pulse and infinite classes:

```
▼<div class="alert alert-info alert-position animated pulse infinite" data-alert-animate>
  <a href="#" class="close" data-dismiss="alert" aria-label="close">x</a>
  <a href="#" class="close minimize" data-minimize="alert" aria-label="minimize">_</a>
  <a href="#" class="close expand hide" data-expand="alert" aria-label="expand">+</a>
  ▷<strong>...</strong>
  ▷<span>...</span>
</div>
```

Figure 7.7: By examining the page source inside our browser, we can see the dynamically transformed markup: Our alert now has the animated, pulse, and infinite classes applied to it

Great, but it isn't exactly extensible. Our plugin does not allow for these classes to be overridden via the `data-alert-animate` attribute. Let's fix this by defining a default animation, using `pulse infinite`, but allowing a developer to override the animation via the `data-alert-animate` attribute. Update the `on load` function as follows:

```
$(window).on('load', function() {
  $('[data-alert-animate]').each(function() {
    var defaultAnimations = 'animated pulse infinite'
    var $animations = $(this).attr('data-alert-animate')
    if ($animations) {
      $(this).addClass('animated ' + $animations)
    } else {
      $(this).addClass(defaultAnimations)
    }
  })
})
```

Now, we are defining a `defaultAnimations` variable from `animated`, `pulse` and `infinite`. We then check if the `data-alert-animate` attribute has any value; if it has, add the classes referenced plus the `animated` class. If not, simply apply the classes defined in `defaultAnimations`.

Loading MyPhoto as-is, we should still see the `animated`, `pulse`, and `infinite` classes applied. However, the alert looks a little awkward, as it renders in a static state before the animation takes effect. To make the behavior of the alert more natural, let's make it invisible until `Alert.js` adds the animation classes. Add a new class, `hide-before-animated`, to `myphoto.css`:

```
.hide-before-animated {  
    visibility: hidden;  
}  
.hide-before-animated:animated {  
    visibility: visible;  
}
```

The `hide-before-animated` class simply sets the `visibility` of the element to `hidden`, unless the `animated` class is also present on the element, in which case `visibility` is set to `visible`. As the `data-alert-animate` attribute adds the `animated` class to an element, the element will be invisible until the element is ready. Add the `hide-before-animated` class to the `class` attribute of the special offers alert element and reload the page to see the results:

```
<div class="alert alert-info alert-position hide-before-  
animated" data-alert-animate>
```

Let's update our special offers alert element to override the `pulse` and `infinite` values with just the `bounceIn` class:

```
<div class="alert alert-info alert-position" data-alert-  
animate="bounceIn">
```

Now, we should see just the `animated` and `bounceIn` classes applied to the element:

```
▼<div class="alert alert-info alert-position animated bounceIn" data-alert-animate="bounceIn">  
  <a href="#" class="close" data-dismiss="alert" aria-label="close">x</a>  
  <a href="#" class="close minimize" data-minimize="alert" aria-label="minimize">_</a>  
  <a href="#" class="close expand hide" data-expand="alert" aria-label="expand">+</a>  
  ▶ <strong>...</strong>  
  ▶ <span>...</span>  
</div>
```

Figure 7.8: By examining the page source inside our browser, we can see how only the `animated` and `bounceIn` classes have now been applied to our alert

Perfect. Now, via a neat little `data-attribute`, we can initiate a default animation for our alerts, and override that animation if need be.

Another nice use of these animations is to add a natural feel to the rendering of elements that may initially be hidden. Let's apply this idea to the **Testimonials** tab we created earlier in this chapter.

Animating a Salvattore grid

The `testimonials` component is a very simple grid built with Salvattore. While it is a neat grid, we want to make it a little flashier. Let's simply add an animation from Animate.css, the `fadeIn` class, so that, when the **Testimonial** tab is open, the grid appears to fade into view:

```
<div class="myphoto-testimonial-grid animated fadeIn" data-columns>
```

When a user clicks on the **Testimonials** tab, the tab panel will open and then the grid will fade into view.

We can go a bit further and actually apply an animation to the `column` groups, to induce optimal motion sickness. In `myphoto.css`, we already leverage the `data-columns` attribute to allow Salvattore to create the column groups for the grid and apply the appropriate style:

```
.myphoto-testimonial-grid[data-columns]::before {  
    content: '4 .col-xs-3.myphoto-testimonial-group';  
}
```

We can make the columns bounce into view by simply extending this rule to include the `animated` and `bounceIn` classes:

```
.myphoto-testimonial-grid[data-columns]::before {  
    content: '4 .col-xs-3.myphoto-testimonial-  
group.animated.bounceIn';  
}
```

Now, when the tab is opened, the whole grid fades in while the column groups also have a bounce effect. A little crass, but effective. The resulting markup for the component should look like the following:

```
▼<div class="myphoto-testimonial-grid animated fadeIn" data-columns="4">  
►<div class="col-xs-3 myphoto-testimonial-group animated bounceIn">...</div>  
►<div class="col-xs-3 myphoto-testimonial-group animated bounceIn">...</div>  
►<div class="col-xs-3 myphoto-testimonial-group animated bounceIn">...</div>  
▼<div class="col-xs-3 myphoto-testimonial-group animated bounceIn">
```

Figure 7.9: By examining the page source inside our browser, we can see that the testimonial grid now has bounce and fade effects applied to it

As far as Animate.css goes, that is all there is. A nice, simple, and elegant library for providing nice, simple, and elegant animations. For a full list of effects, take a look at the Animate demo website at <https://daneden.github.io/animate.css/>.

Next, we are going to take a look at a complementary library, Hover.

Hover

Hover, <http://ianlunn.github.io/Hover/>, is a neat, pure CSS library that provides transition effects. Adhering to the “*Do one thing, do one thing well*” approach, Hover only concerns itself with hover transitions, as you may have guessed from the name. Hover comes baked in with a huge array of transitions and provides easy integration with CSS, Less, and Sass.

Hover breaks these transitions into seven distinct groups:

- 2D transitons
- Background transitions
- Icons, by leveraging Font Awesome icons
- Border transitons
- Shadow and glow transitions, simulating 3D transitions
- Speech bubbles
- Curls

Throughout this section, we will touch on a number of these different groups. An extensive list of the transitions is available on the Hover website. Before we get to that, let's add Hover to MyPhoto.

Adding Hover to MyPhoto

Add Hover to MyPhoto via Bower:

```
bower install Hover
```

Within the `bower_components` directory, there should now be a `Hover` directory. We will reference the minified CSS straight into our markup. Add the following `link` tag in the head of the MyPhoto HTML, below the existing CSS references:

```
<link rel="stylesheet" href="bower_components/Hover/css/  
hover-min.css" />
```

Let's add a Hover transition to make sure everything is in working order.

Making the navbar grow

As a quick test to make sure we have completed setting up of Hover correctly, let's apply some Hover transitions to the MyPhoto navbar.

Using Hover transitions is even simpler than Animate.css. We simply add one class to the element, and voila, a Hover transition is applied. As you will notice, all Hover classes are prefixed with `hvr-` to help avoid conflicts with other style sheets on the page.

We're going to spice up our `nav-link` links with one of the border transitions from Hover: `hvr-underline-from-center`. The `hvr-underline-from-center` class renders an underline on a given element, which grows from the center of the element. Let's add this to our five `nav-link` links:

```
<ul class="nav nav-pills">
    <li class="nav-item"><a class="nav-link hvr-underline-from-center" href="#welcome">Welcome</a></li>
    <li class="nav-item"><a class="nav-link hvr-underline-from-center" href="#services">Services</a></li>
    <li class="nav-item"><a class="nav-link hvr-underline-from-center" href="#gallery">Gallery</a></li>
    <li class="nav-item"><a class="nav-link hvr-underline-from-center" href="#about">About</a></li>
    <li class="nav-item"><a class="nav-link hvr-underline-from-center" href="#contact">Contact Us</a></li>
    <li class="nav-item dropdown pull-xs-right">
        ...
    </li>
</ul>
```

Take a look at the following screenshot:

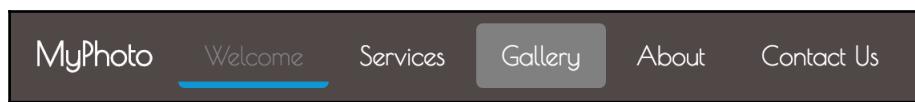


Figure 7.10: A border transition temporarily added to our navigation menu items.
As the user hovers over one of the navigation links, they receive a clear visual cue.

Now, when a user hovers on one of our navigation links, they get a nice indicator providing a clear visual aid.

Awesome Hover icons

As previously mentioned, Hover has a range of icon effects. Hover leverages the Font Awesome library, to render icons, which Hover then animates. Hover provides a range of different effects, but each effect actually only applies to one icon. All these effects are prefixed with `hvr-icon-<effect name>`. Open

`bower_components-hover/css-hover.css` and check out the `hvr-icon-bob:before` class:

```
.hvr-icon-bob:before {  
    content: "\f077";  
    position : absolute;  
    right: 1em;  
    padding: 0 1px;  
    font-family: FontAwesome;  
    -webkit-transform: translateZ(0);  
    transform: translateZ(0);  
}
```

Using the pseudo-selector `before`, `hvr-icon-bob` is creating an element with `content: \f077`, which relates to the icon specified by that id in Font Awesome, as the `font-family` is set to `FontAwesome`. Taking a look at `FontAwesome.css`, we can see that `\f077` represents the `fa-chevron-up` class. As you can see, the icon is hardcoded into the class, so we can only ever use `hvr-icon-bob` with the upward chevron icon, unless we override the `content` rule of that class. The `transform` properties with the value of `translateZ()` are used to make the transition smoother, as it creates a new stacking context. The new stacking context forces rendering of the animation to the GPU, creating a smoother transition.

Let's see `hvr-icon-bob` in action by updating the **Profile** drop-down button, replacing the `caret` class with the `hvr-icon-bob` class. Replace the **Profile** button markup with the following:

```
<a href="#" class="nav-link" data-toggle="dropdown" role="button"  
    aria-haspopup="true" aria-expanded="false">  
    <span class="hvr-icon-bob">Profile</span>  
</a>
```

Take a look at the following screenshot:



Figure 7.11: An inverted chevron that moves up and down as the user hovers over it

When a user hovers over the icon, the icon will animate up and down. Great! Well, except that it is upside down and misaligned. Let's fix that. In `myphoto.css`, we will override the `content` property of `hvr-icon-bob` with `\f078`, which is the `fa-chevron-down` class, and we will vertically align it correctly:

```
.hvr-icon-bob :before {  
    content: "\f078";  
}  
.hvr-icon-bob {  
    vertical-align: top;  
}
```

We also need to ensure that `myphoto.css` is loaded after Hover:

```
<link rel="stylesheet" href="bower_components/bootstrap/dist/css/  
bootstrap.min.css" />  
<link rel="stylesheet" href="bower_components/Hover/css/hover-min.css"  
/>
```

Take a look at the following screenshot:

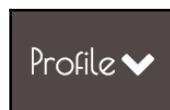


Figure 7.12: An animated chevron facing down

Better! However, this approach can easily become a maintenance nightmare. There are other approaches to managing Hover icon classes so they're more usable. For instance, we could use specific classes to determine which icon to display. Add the `.myphoto-chevron-down` selector to the `hvr-icon-bob:before` rule in `myphoto.css`:

```
.myphoto-chevron-down.hvr-icon-bob:before {  
    content: "\f078";  
}
```

Refresh the page, and the upwards chevron will be displayed again.

But, if we add the `myphoto-chevron-down` class to the element, the downward chevron will be rendered:

```
<span class="myphoto-chevron-down hvr-icon-bob">Profile</span>
```

The approach described here makes using the Hover icon animations far more maintainable, and adds much more context than just using `hvr-icon-bob`, which just describes the animation and not the icon. We could also simply use the actual Font Awesome classes to describe the behavior, but Font Awesome also provides display rules that may not be in line with how Hover designed these classes to work.

Salvattore Hover

We can also use Hover to improve a user's interaction with our Salvattore-powered `testimonials` component. The `testimonials` component already leverages Animate.css to add an interesting transition when rendering the grid, but we can use Hover to add an interesting transition when a user actually interacts with it.

The grid is quite flat and fails to grab the user's attention once loaded. The individual columns also fail to gain focus. To solve this, we can use Hover to increase the size of the column when a user hovers on the column. One of the classes provided by Hover is `hvr-grow-shadow`, which adds a hover state to an element that expands the column and adds a `drop-shadow`, but doesn't affect the other columns or rows within the grid. All we need to do here is add `hvr-grow-shadow` to each testimonial column:

```
<div class="myphoto-testimonial-grid animated fadeIn" data-columns>
  <div class="myphoto-testimonial-column hvr-grow-shadow">
    <h6>Debbie</h6>
    <p>Great service! Would recommend to friends!</p>
  </div>
  <div class="myphoto-testimonial-column hvr-grow-shadow">
    <h6>Anne</h6>
    <p>Really high quality prints!</p>
  </div>
  <div class="myphoto-testimonial-column hvr-grow-shadow">
    <h6>Oscar</h6>
    <p>Declared their greatness, exhibited greatness.</p>
  </div>
  <div class="myphoto-testimonial-column hvr-grow-shadow">
    <h6>Joey</h6>
    <p>5 stars! Thanks for the great photos!</p>
  </div>
  <div class="myphoto-testimonial-column hvr-grow-shadow">
    <h6>Mary</h6>
    <p>Made a stressful event much easier! Absolute professionals!</p>
  </div>
  <div class="myphoto-testimonial-column hvr-grow-shadow">
    <h6>Alice</h6>
```

```
<p>Wonderful! Exactly as I imagined they would turn out!</p>
</div>
<div class="myphoto-testimonial-column hvr-grow-shadow">
  <h6>Jack & Jill</h6>
  <p>So happy with how the photos turned out! Thanks for
  capturing the memories of our day!</p>
</div>
<div class="myphoto-testimonial-column hvr-grow-shadow">
  <h6>Nick</h6>
  <p>Perfectly captured the mood of our gig. Top notch.</p>
</div>
<div class="myphoto-testimonial-column hvr-grow-shadow">
  <h6>Tony</h6>
  <p>Captured our Cup final win! Great stuff!</p>
</div>
</div>
```

Take a look at the following screenshot:

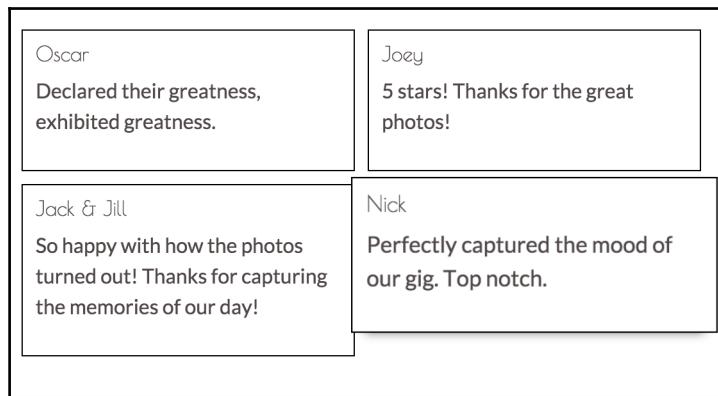


Figure 7.13: After applying the hvr-grow-shadow class, hovering over an individual testimonial will cause it to “grow” without effecting the rest of the grid

Now, when a user hovers over a column, the individual column *grows*, without affecting the rest of the grid. One of the cool things about Hover is that all the classes play nicely with each other, following the cascading nature of CSS. No JavaScript magic, just pure CSS. To see an example of this, we will add another Hover class. This one is from the range of Hover background transition classes, `hvr-sweep-to-top`. The `hvr-sweep-to-top` class animates a change in the background color with a fill effect from the bottom to the top. Let's add `hvr-sweep-to-top` to the testimonial columns:

```
<div class="myphoto-testimonial-grid animated fadeIn" data-columns>
  <div class="myphoto-testimonial-column hvr-grow-shadow
  hvr-sweep-to-top">
```

```
<h6>Debbie</h6>
<p>Great service! Would recommend to friends!</p>
</div>
<div class="myphoto-testimonial-column hvr-grow-shadow
hvr-sweep-to-top">
<h6>Anne</h6>
<p>Really high quality prints!</p>
</div>
<div class="myphoto-testimonial-column hvr-grow-shadow
hvr-sweep-to-top">
<h6>Oscar</h6>
<p>Declared their greatness, exhibited greatness.</p>
</div>
<div class="myphoto-testimonial-column hvr-grow-shadow
hvr-sweep-to-top">
<h6>Joey</h6>
<p>5 stars! Thanks for the great photos!</p>
</div>
<div class="myphoto-testimonial-column hvr-grow-shadow
hvr-sweep-to-top">
<h6>Mary</h6>
<p>Made a stressful event much easier! Absolute
professionals!</p>
</div>
<div class="myphoto-testimonial-column hvr-grow-shadow
hvr-sweep-to-top">
<h6>Alice</h6>
<p>Wonderful! Exactly as I imagined they would turn out!</p>
</div>
<div class="myphoto-testimonial-column hvr-grow-shadow
hvr-sweep-to-top">
<h6>Jack & Jill</h6>
<p>So happy with how the photos turned out! Thanks for
capturing the memories of our day!</p>
</div>
<div class="myphoto-testimonial-column hvr-grow-shadow
hvr-sweep-to-top">
<h6>Nick</h6>
<p>Perfectly captured the mood of our gig. Top notch.</p>
</div>
<div class="myphoto-testimonial-column hvr-grow-shadow
hvr-sweep-to-top">
<h6>Tony</h6>
<p>Captured our Cup final win! Great stuff!</p>
</div>
</div>
```

Take a look at the following screenshot:

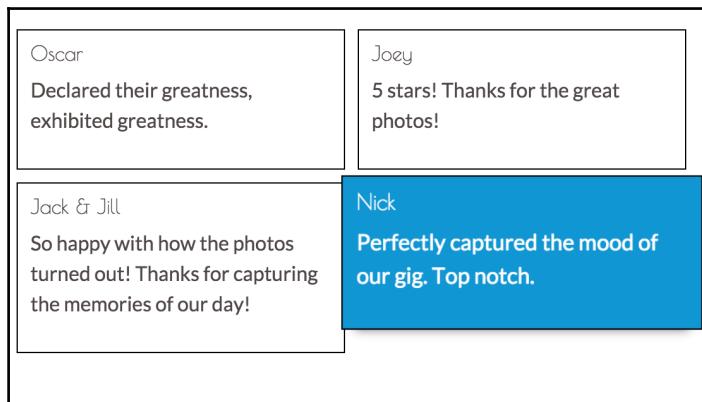


Figure 7.14: After applying the hvr-grow-shadow and hvr-sweep-to-top classes, hovering over an individual testimonial will cause it to “grow” and change color

Now we have both the expanded columns, with a drop-shadow, along with the fill effect provided by the `hvr-sweep-to-top` class. As we have seen, Hover provides very simple to use but very elegant transitions to add an extra layer of interaction for users. Being pure CSS, Hover is also exceedingly simple to integrate with most libraries and frameworks.

Summary

In this chapter, we have learned how to integrate third-party JavaScript and CSS libraries with our Bootstrap website, through Salvattore, Animate.css, and Hover.

We have been able to leverage Salvattore to add extra flexibility to Bootstrap's native grid system. We have also seen how to integrate Animate.css to add interesting transitions to MyPhoto, while integrating heavily with both Bootstrap and Salvattore. To improve the user experience of MyPhoto, we have seen how to leverage Hover, which has allowed us to provide improved visual cues to the user.

While Bootstrap provides a great amount of functionality out-of-the-box, using narrowly focused libraries such as Salvattore, Animate.css, and Hover can really improve the look and feel of a website.

Let's move on to Chapter 8, in which we will learn how to optimize MyPhoto.

8

Optimizing Your Website

Loosely put, *website optimization* refers to the activities and processes that improve your website's user experience and visibility while reducing the costs associated with hosting your website. In his book *Website Optimization*, Andrew B. King summarizes this notion succinctly with the question: “*How do we make our website better?*” (*Website Optimization*, Andrew B. King, O'Reilly). As such, the topic has been the sole subject of entire books, and a single chapter barely touches the tip of the iceberg. The topic is one of many facets, ranging from server-side optimization, search engine optimization, pay-per-click optimization, and client-side optimization. In this chapter, we will only discuss the latter. That is, we will be improving the loading and rendering time of MyPhoto. Specifically, this chapter is concerned with:

- Speeding up the loading time of our `MyPhoto index.html`
- Automating the tasks that achieve this objective

By the end of this chapter, you will understand the essential techniques behind client-side optimization. Within the context of MyPhoto you will therefore learn how to:

- Reduce the overall number of HTTP requests required to render our web page
- Automatically remove unused CSS rules
- Make our JavaScript and CSS files smaller (commonly referred to as minification)
- Automate the various optimization tasks
- Optimize CSS rules

CSS optimization

Before we even consider compression, minification, and file concatenation, we should think about the ways in which we can simplify and optimize our existing style sheet without using third-party tools. Of course, we should have striven for an optimal style sheet to begin with, and in many aspects we did. However, our style sheet still leaves room for improvement. Some of these improvements we have ignored on purpose within the previous chapters, as they would have detracted from the chapter's intended purpose. However, as this chapter is concerned with optimizing the client-side code of a web page, the time has come to talk a little about general tips and practices that will help you keep your style sheets small and your rules short. We will address each of these tips and practices in turn.

Inline styles

If, after reading this chapter, you only remember one thing, then please let it be the following: inline styles are bad. Period. Avoid using them whenever possible. Why? Because not only will they make your website impossible to maintain as the website grows, they also take up precious bytes as they force you to repeat the same rules over and over. Consider the following markup for our **Gallery** section:

```
<div class="carousel-inner" role="listbox">
    <div style="height: 400px" class="carousel-item active">
        <img data-modal-picture="#carouselModal" src=
            "images/brazil.png">
        <div class="carousel-caption">
            Brazil
        </div>
    </div>
    <div style="height: 400px" class="carousel-item">
        <img data-modal-picture="#carouselModal" src=
            "images/datsun.png">
        <div class="carousel-caption">
            Datsun 260Z
        </div>
    </div>
    <div style="height: 400px" class="carousel-item">
        <img data-modal-picture="#carouselModal" src=
            "images/skydive.png">
        <div class="carousel-caption">
            Skydive
        </div>
    </div>
</div>
```

Notice how the rule for defining a gallery item's height, `style="height: 400px"`, is repeated three times, once for each of the three gallery items. That's an additional 21 characters (or 21 bytes assuming that our document is UTF-8) for each additional image. Multiplying $3 * 21$ gives us 63 bytes. And 21 more bytes for every new image that you want to add to the **Gallery**. Not to mention that, if you ever want to update the `height` of the gallery images, you will need to manually update the `style` attribute for every single image. The solution is of course to replace the inline styles with an appropriate class. Let's go ahead and define an `img` class that can be applied to any carousel image:

```
.carousel-item {  
    height: 400px;  
}
```

Now let's go ahead and remove the style rules:

```
<div class="carousel-inner" role="listbox">  
    <div class="carousel-item active">  
        <img data-modal-picture="#carouselModal" src=  
            "images/brazil.png">  
        <div class="carousel-caption">  
            Brazil  
        </div>  
    </div>  
    <div class="carousel-item">  
        <img data-modal-picture="#carouselModal" src=  
            "images/datsun.png">  
        <div class="carousel-caption">  
            Datsun 260Z  
        </div>  
    </div>  
    <div class="carousel-item">  
        <img data-modal-picture="#carouselModal" src=  
            "images/skydive.png">  
        <div class="carousel-caption">  
            Skydive  
        </div>  
    </div>  
</div>
```

Great! Not only is our CSS now easier to maintain, we also shaved 29 bytes off our website (the original inline styles required 63 bytes; our new class definition, however, requires only 34 bytes). Yes, this does not seem like much, especially in the world of high-speed broadband. But remember, your website will grow, and every byte adds up.

There are several more inline styles spread around our HTML document. Go ahead and fix them before moving on to the next section.

Long identifier and class names

The longer your strings, the larger your files. It's a no-brainer. As such, long identifier and class names naturally increase the size of your web page. Of course, extremely short class or identifier names tend to lack meaning, and therefore will make it more difficult (if not impossible) to maintain your page. As such, one should strive for an ideal balance between length and expressiveness (we will be covering a handy little tool that will provide you with the benefits of both later on in this chapter). Of course, even better than shortening identifiers, is to remove them altogether. One handy technique for removing these is to use hierarchical selection. Take our events pagination code. For example, we are using the services-events-content identifier within our pagination logic as follows:

```
$('#services-events-pagination').bootpag({
    total: 10
}).on("page", function(event, num){
    $('#services-events-content div').hide();

    var current_page = '#page-' + num;
    $(current_page).show();
});
```

To denote the services content, we broke the name of our identifier into three parts, namely, services, events, content. Our markup is as follows:

```
<div id="services-events-content">
    <div id="page-1">
        <h3>My Sample Event #1</h3>
        ...
    </div>
</div>
```

Let's try and get rid of this identifier altogether by observing two characteristics of our **Events** section:

- The services-events-content is an indirect descendent of a `div` with the `id services-events`. This `id` we cannot remove, as it is required for the menu to work.
- The element with the `id services-events-content` is itself a `div`. If we were to remove its `id`, we could also remove the entire `div`.

As such, we do not need a second identifier to select the pages that we wish to hide. Instead, all that we need to do is select the `div` within the `div` that is within the `div` that is assigned the `id services-events`. How do we express this as a CSS selector? Easy. Use `#services-events div div div`. And as such, our pagination logic is updated as follows:

```
$('#services-events-pagination').bootpag({
    total: 10
}).on("page", function(event, num){
    $('#services-events div div div').hide();
    var current_page = '#page-' + num;
    $(current_page).show();
});
```

Save and refresh. What's that? As you clicked on a page, the pagination control disappeared. That is because we are now hiding all `div` elements that are two `div` elements down from the element with the `id services-events`. Move the pagination control `div` outside its parent element. Our markup should now look as follows:

```
<div class="tab-content bg-myphoto-light">
    <div role="tabpanel" class="tab-pane active" id="services-events">
        <div class="container">
            <div class="row" style="margin: 1em;">
                <div id="page-1">
                    <h3>My Sample Event #1</h3>
                    ...
                    <h3>My Sample Event #2</h3>
                    ...
                </div>
                <div id="page-2">
                    <h3>My Sample Event #3</h3>
                    ...
                </div>
            </div>
            <div id="services-events-pagination"></div>
        </div>
    </div>
</div>
```

Save and refresh. That's better! Last but not least, let us update `myphoto.css`. Take the following code:

```
#services-events-content div {
    display: none;
}
#services-events-content div img {
```

```
    margin-top: 0.5em;
    margin-right: 1em;
}
#services-events-content {
    height: 15em;
    overflow-y: scroll;
}
```

Replace this code with:

```
#services-events div div div {
    display: none;
}
#services-events div div div img {
    margin-top: 0.5em;
    margin-right: 1em;
}
#services-events div div div {
    height: 15em;
    overflow-y: scroll;
}
```

That's it, we have simplified our style sheet and saved some bytes in the process!

Shorthand rules

According to, the Mozilla Developer Network (shorthand properties, Mozilla Developer Network, https://developer.mozilla.org/en-US/docs/Web/CSS/Shorthand_properties, accessed November 2015), shorthand properties are:

“CSS properties that let you set the values of several other CSS properties simultaneously. Using a shorthand property, a Web developer can write more concise and often more readable style sheets, saving time and energy.”

– Mozilla Developer Network, 2015

Unless strictly necessary, we should never be using longhand rules. When possible, shorthand rules are always the preferred option. Besides the obvious advantage of saving precious bytes, shorthand rules also help increase your style sheet's maintainability. For example, border: 20px dotted #FFF is equivalent to three separate rules:

```
border-style: dotted;
border-width: 20px;
border-color: #FFF;
```

Grouping selectors

Organizing selectors into groups will arguably also save some bytes. Consider lines 80 to 93 in `myphoto.css`:

```
.navbar-myphoto .dropdown-menu > a:hover {  
    color: gray;  
    background-color: #504747;  
}  
.navbar-myphoto .dropdown-menu > a:focus {  
    color: gray;  
    background-color: #504747;  
}  
.navbar-myphoto .dropdown-menu > .active > a:focus {  
    color: gray;  
    background-color: #504747;  
}
```

Notice how each of the three selectors contains the same declarations. That is, the `color` and `background-color` properties are set to the exact same values for each selector. To prevent us from repeating these declarations, we should simply group them (reducing the code from 274 characters down to 181 characters):

```
.navbar-myphoto .dropdown-menu > a:hover,  
.navbar-myphoto .dropdown-menu > a:focus,  
.navbar-myphoto .dropdown-menu > .active > a:focus {  
    color: gray;  
    background-color: #504747;  
}
```

Voilà! We just saved 93 bytes! (assuming UTF-8 encoding).

Rendering times

When optimizing your style rules, the number of bytes should not be your only concern. In fact, it comes secondary to the rendering time of your web page. CSS rules affect the amount of work that is required by the browser to render your page. As such, some rules are more expensive than others. For example, changing the color of an element is cheaper than changing its margin. The reason for this is that a change in color only requires your browser to draw the new pixels. While drawing itself is by no means a cheap operation, changing the margin of an element requires much more effort. Your browser needs to both re-calculate the page layout and also draw the changes. Optimizing your page's rendering times is a complex topic, and as such beyond the scope of this book.

However, we recommend that you take a look at <http://csstriggers.com/>. This site provides a concise overview of the costs involved when updating a given CSS property.

Did you know?



Udacity now offers a free online course on *Browser Rendering Optimization*. Head over to <https://www.udacity.com> for more. We cannot recommend the course highly enough!

Minifying CSS and JavaScript

Once you have improved the `MyPhoto` style rules so they're as compact, efficient, and maintainable as possible, it is time to look into minification. Minification is the process of removing redundant characters from a file, without altering the actual information contained within it. In other words, minifying our `myphoto.css` file will reduce its overall size, while leaving the actual CSS style rules intact. This is achieved by stripping out any whitespace characters within our file. Stripping out whitespace characters has the obvious result that our CSS is now practically unreadable and impossible to maintain. As such, minified style sheets should only be used when serving a page (that is, during production), and not during development.

Clearly minifying your style sheet manually would be an incredibly time-consuming (and hence pointless) task. Therefore, there exist many tools that will do the job for us. One such tool is `npm minifier`. Visit <https://www.npmjs.com/package/minifier> for more.

Let's go ahead and install it:

```
sudo npm install -g minifier
```

Once installed, we can minify our style sheet by typing the following command:

```
minify path-to-myphoto.css
```

Here `path-to-myphoto.css` represents the path to our `MyPhoto` style sheet. Go ahead and execute the command. Once minification is complete, you should see the message **Minification complete**. A new CSS file (`myphoto.min.css`) will have been created inside the directory containing the `myphoto.css` file. The new file should be 3,358 bytes. Our original `myphoto.css` file is 3,945 bytes. Minifying our style sheet just reduced the number of bytes to send by roughly 15%!

Go ahead and update the head of our HTML document to reference the new, minified style sheet:

```
<link rel="stylesheet" href="styles/myphoto.min.css" />
```

It is worth noting that, aside from CSS minification, minifier also allows you to minify JavaScript files. For example, to minify our `alert.js` file, simply type:

```
minify path-to-alert.js
```

Once again, as soon as the minification is complete, you should see the message

Minification complete. Similar to before, a new file (`alert.min.js`) will have been created inside the directory containing the `alert.js` file.

Introducing Grunt

The minifier that we used in the previous section greatly reduced the size of our style sheet and JavaScript files, and also helped reduce the overall number of requests required to render `MyPhoto`. However, using it has one downside—every time that you make a change to either your CSS or JavaScript code during development, you are required to rerun the tool. This greatly slows down development and can even cause frustration and hair tearing. (Just imagine forgetting to run the minifier, thinking that you ran it, and not seeing your changes appear. You are likely to blame your code as opposed to your forgetfulness.) Therefore, would it not be nice if we could minify and concatenate our files automatically every time that we make a change to our source code?

Meet **Grunt**: The JavaScript Task Runner (<http://gruntjs.com/>). As implied by its name, Grunt is a tool that allows us to automatically run any set of tasks. Grunt can even *wait* while you code, and pickup changes made to your source code files (CSS, HTML, or JavaScript) and then execute a preconfigured set of tasks every time that you save your changes. This way you are no longer required to manually execute a set of commands in order for your changes to take effect.

Let's go ahead and install Grunt:

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

Before we can start using run with `MyPhoto`, we need to tell Grunt:

- What tasks to run. That is, what to do with the input (the input being our `MyPhoto` files) and where to save the output.
- What software is to be used to execute the tasks.

- How to name the tasks, so that we can invoke them when required.

With this in mind, we create a new JavaScript file (assuming UTF-8 encoding), called `Gruntfile.js`, inside our project root. We will also need to create a JSON file, called `package.json`, also inside our project root. Our project folder should have the following structure (note how we created one additional folder, `src`, and moved our source code and development assets inside it):

```
src
|__bower_components
|__images
|__js
|__styles
|__index.html
Gruntfile.js
package.json
```

Open the newly created `Gruntfile.js` and insert the following function definition:

```
module.exports = function(grunt) {
    grunt.initConfig({
        pkg: grunt.file.readJSON("package.json")
    });
};
```

As you can see, this is plain, vanilla JavaScript. Anything that we need to make Grunt aware of (such as the Grunt configuration) will go inside the `grunt.initConfig` function definition. Adding the configuration outside the scope of this function will cause Grunt to ignore it.

Now open `package.json` and insert the following:

```
{
  "name": "MyPhoto",
  "version": "1.0",
  "devDependencies": {
  }
}
```

The preceding code should be self-explanatory. The `name` property refers to the project name, `version` refers to the project's version, and `devDependencies` refers to any dependencies that are required (we will be adding to those in a while).

Great, now we are ready to start using Grunt!

Minification and concatenation using Grunt

The first thing that we want Grunt to be able to do is minify our files. Yes, we already have minifier installed, but remember that we want to use Grunt so that we can automatically execute a bunch of tasks (such as minification) in one go. To do so, we will need to install the `grunt-contrib-cssmin` package (a Grunt package that performs minification and concatenation. Visit <https://github.com/gruntjs/grunt-contrib-cssmin> for more information.):

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

Once installed, inspect `package.json`. Observe how it has been modified to include the newly installed package as a development dependency:

```
{
  "name": "MyPhoto",
  "version": "0.1.0",
  "devDependencies": {
    "grunt": "^0.4.5",
    "grunt-contrib-cssmin": "^0.14.0"
  }
}
```

We must tell Grunt about the plugin. To do so, insert the following line inside the function definition within our `Gruntfile.js`:

```
grunt.loadNpmTasks("grunt-contrib-cssmin");
```

Our `Gruntfile.js` should now look as follows:

```
module.exports = function(grunt) {
  grunt.initConfig({
    pkg: grunt.file.readJSON("package.json")
  });
  grunt.loadNpmTasks("grunt-contrib-cssmin");
};
```

As such, we still cannot do much. The preceding code makes Grunt aware of the `grunt-contrib-cssmin` package (that is, it tells Grunt load it). In order to be able to use the package to minify our files, we need to create a Grunt task. We need to call this task `cssmin`:

```
module.exports = function(grunt) {
  grunt.initConfig({
    pkg: grunt.file.readJSON("package.json"),
    "cssmin": {
```

```
        "target": {
          "files": {
            "src/styles/myphoto.min.css": [
              "src/styles/*.css"
            ]
          }
        }
      });
      grunt.loadNpmTasks("grunt-contrib-cssmin");
    };
  
```

Whoa! That's a lot of code at once. What just happened here? Well, we registered a new task called `cssmin`. We then specified the target, that is, the input files that Grunt should use for this task. Specifically, we wrote:

```
"src/styles/myphoto.min.css": ["src/styles/*.css"]
```

The property name here is being interpreted as denoting the output, while the property value represents the input. We are therefore in essence saying something along the lines of: *“In order to produce myphoto.min.css use any files ending with the file extension css within the src/styles directory”*.

Go ahead and run the Grunt task by typing:

```
grunt cssmin
```

Upon completion, you should see output along the lines of:

```
[Benjamins-MacBook-Pro:ch08 benjaminjakobus$ grunt cssmin
Running "cssmin:target" (cssmin) task
>> 1 file created. 4.99 kB → 3.25 kB

Done, without errors.
```

Figure 8.1: The console output after running `cssmin`

The first line indicates that a new output file (`myphoto.min.css`) has been created, and that it is **3.25 kB** in size (down from the original **4.99 kB**). The second line is self-explanatory; that is, the task executed successfully, without any errors.

Now that you know how to use `grunt-contrib-cssmin`, go ahead and take a look at their documentation for some nice extras!

Running tasks automatically

Now that we know how to configure and use Grunt to minify our style sheets, let us turn our attention to task automation. That is, how we can execute our Grunt minification task automatically as soon as we make changes to our source files. To this end, we will learn about a second Grunt package, called `grunt-contrib-watch` (<https://github.com/gruntjs/grunt-contrib-watch>). As with `contrib-css-min`, this package can be installed using npm:

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

Open `package.json` and verify that `grunt-contrib-watch` has been added as a dependency:

```
{
  "name": "MyPhoto",
  "version": "0.1.0",
  "devDependencies": {
    "grunt": "^0.4.5",
    "grunt-contrib-cssmin": "^0.14.0",
    "grunt-contrib-watch": "^0.6.1"
  }
}
```

Next, tell Grunt about our new package by adding `grunt.loadNpmTasks('grunt-contrib-watch');` to `Gruntfile.js`. Furthermore, we need to define the `watch` task by adding a new empty property called `watch`:

```
module.exports = function(grunt) {
  grunt.initConfig({
    pkg: grunt.file.readJSON("package.json"),
    "cssmin": {
      "target": {
        "files": {
          "src/styles/myphoto.min.css": [
            "src/styles/*.css", "src/styles!*.min.css"
          ]
        }
      }
    },
    "watch": {}
  });
  grunt.loadNpmTasks("grunt-contrib-cssmin");
  grunt.loadNpmTasks("grunt-contrib-watch");
};
```

Now that Grunt loads our newly installed `watch` package, we can execute the command `grunt watch`. However, as we have not yet configured the task, Grunt will terminate with:

```
Benjamins-MacBook-Pro:ch08 benjaminjakobus$ grunt watch
Running "watch" task
Waiting...
```

Figure 8.2: The console output after running the watch task

The first thing that we need to do, is tell our `watch` task what files to actually “watch”. We do this by setting the `files` property, just as we did with `grunt-contrib-cssmin`:

```
"watch": {
  "target": {
    "files": ["src/styles/myphoto.css"],
  }
}
```

This tells the `watch` task to use the `myphoto.css` located within our `src/styles` folder as input (it will only watch for changes made to `myphoto.css`).



Note that in reality, you would want to be watching all CSS files inside `styles/`; however to keep things simple, let's just watch `myphoto.css`.

Go ahead and execute `grunt watch` again. Unlike the first time that we ran the command, the task should not terminate now. Instead, it should halt with the message `Waiting....` Go ahead and make a trivial change (such as removing a whitespace) to our `myphoto.css` file. Then save this change. Notice how the terminal output is now:

```
Benjamins-MacBook-Pro:ch08 benjaminjakobus$ grunt watch
Running "watch" task
Waiting...
>> File "src/styles/myphoto.css" changed.
```

Figure 8.3: The console output after running the watch task

Great! Our `watch` task is now successfully listening for file changes made to any style sheet within `src/styles`. The next step is to put this achievement to good use. That is, we need to get our `watch` task to execute the minification task that we created in the previous section. To do so, simply add the `tasks` property to our `target`:

```
"watch": {  
    "target": {  
        "files": ["src/styles/myphoto.css"],  
        "tasks": ["cssmin"]  
    }  
}
```

Once again, run `grunt watch`. This time, make a visible change to our `myphoto.css` style sheet. For example, you could add an obvious rule such as `body {background-color: red;}`. Observe how, as you save your changes, our `watch` task now runs our `cssmin` task:

```
Running "watch" task  
Waiting...  
>> File "src/styles/myphoto.css" changed.  
Running "cssmin:target" (cssmin) task  
>> 1 file created. 21.22 kB → 20.73 kB  
  
Done, without errors.  
Completed in 1.912s at Fri Feb 26 2016 16:50:41 GMT-0300 (BRT) - Waiting...
```

Figure 8.4: The console output after making a change to the style sheet that is being watched

Refresh the page in your browser and observe the changes. Voilà! We now no longer need to run our minifier manually every time we change our style sheet.

Stripping our website of unused CSS

Dead code is never good. As such, whatever the project that you are working on may be, you should always strive to eliminate code that is no longer in use, as early as possible. This is especially important when developing websites, as unused code will inevitably be transferred to the client, and hence result in additional, unnecessary, bytes being transferred (although maintainability is also a major concern).

Programmers are not perfect, and we all make mistakes. As such, unused code or style rules are bound to slip past us during development and testing. Consequently, it would be nice if we could establish a safeguard to ensure that at least no unused style makes it past us into production. And this is where `grunt-uncss` fits in. Visit <https://github.com/addyosmani/grunt-uncss> for more.

UnCSS strips any unused CSS from our style sheet. When configured properly, it can therefore be very useful to ensure that our production-ready website is as small as possible. Let's go ahead and install UnCSS:

```
sudo npm install grunt-uncss -save-dev
```

Once installed, we need to tell Grunt about our plugin. Just as in the previous sub-sections, update the `Gruntfile.js` by adding the line `grunt.loadNpmTasks('grunt-uncss');` to our Grunt configuration. Next, go ahead and define the `uncss` task:

```
"uncss": {  
    "target": {  
        "files": {  
            "src/styles/output.css": ["src/index.html"]  
        }  
    }  
},
```

In the preceding code, we specified a target consisting of the file `index.html`. This `index.html` will be parsed by Uncss. The `class` and `id` names used within it will be compared to those appearing in our style sheets. Should our style sheets contain selectors that are unused, then those are removed from the output. The output itself will be written to `src/styles/output.css`.

Let's go ahead and test this. Add a new style to our `myphoto.css` that will not be used anywhere within our `index.html`. For example:

```
#foobar {  
    color: red;  
}
```

Save and then run:

```
grunt uncss
```

Upon successful execution, the terminal should display output along the lines of:

```
[Benjamins-MacBook-Pro:ch08 benjaminjakobus$ grunt uncss
Running "uncss:target" (uncss) task
ReferenceError: Can't find variable: $

  file:///Users/benjaminjakobus/Desktop/ch08/src/index.html:34
File src/styles/output.css created: 298.19 kB → 29.69 kB

Done, without errors.
```

Figure 8.5: The console output after executing our uncss task

Go ahead and open the generated `output.css` file. The file will contain a concatenation of all our CSS files (including Bootstrap). Go ahead and search for `#foobar`. Find it? That's because UnCSS detected that it was no longer in use and removed it for us.

Now, we successfully configured a Grunt task to strip our website of the unused CSS. However, we need to run this task manually. Would it not be nice if we could configure the task to run with the other `watch` tasks? If we were to do this, the first thing that we would need to ask ourselves is, how do we combine the CSS minification task with UnCSS? After all, `grunt watch` would run one before the other. As such, we would be required to use the output of one task as input for the other. So how would we go about doing this?

Well, we know that our `cssmin` task writes its output to `myphoto.min.css`. We also know that `index.html` references `myphoto.min.css`. Furthermore, we also know `uncss` receives its input by checking the style sheets referenced in `index.html`. We therefore know that the output produced by our `cssmin` task is sure to be used by our `uncss` as long as it is referenced within `index.html`.

In order for the output produced by `uncss` to take effect, we would therefore need to reconfigure the task to write its output into `myphoto.min.css`. We would then need to add `uncss` to our list of `watch` tasks, taking care to insert the task into the list after `cssmin`. However, this leads to a problem: running `uncss` after `cssmin` will produce an un-minified style sheet. Furthermore, it also requires the presence of `myphoto.min.css`. However, as `myphoto.min.css` is actually produced by `cssmin`, the sheet will not be present when running the task for the first time. We therefore need a different approach. We will need to use the original `myphoto.css` as input to `uncss`, which then writes its output into a file called `myphoto.min.css`.

Our `cssmin` task then uses this file as input, minifying it as discussed previously. Since `uncss` parses the style sheet references in `index.html`, we would need to first revert our `index.html` to reference our development style sheet, `myphoto.css`. Go ahead and do just that. Replace the line: `<link rel="stylesheet" href="styles/myphoto.min.css" />` with: `<link rel="stylesheet" href="styles/myphoto.css" />`.

Processing HTML

For the minified changes to take effect, we now need a tool that replaces our style sheet references with our production-ready style sheets. Meet `grunt-processhtml`. Visit <https://www.npmjs.com/package/grunt-processhtml> for more.

Go ahead and install it using the following command:

```
sudo npm install grunt-processhtml --save-dev
```

Add `grunt.loadNpmTasks('grunt-processhtml');` to our `Gruntfile.js` to enable our freshly installed tool.

While `grunt-processhtml` is very powerful, we will only cover how to replace style sheet references. We therefore recommend that you read the tool's documentation to discover further features.

In order to replace our style sheets with `myphoto.min.css`, we wrap them inside special `grunt-processhtml` comments:

```
<!-- build:css styles/myphoto.min.css -->
<link rel="stylesheet" href="bower_components/bootstrap/
dist/css/bootstrap.min.css" />
<link href='https://fonts.googleapis.com/css?family=Poiret+One'
rel='stylesheet' type='text/css'>
<link href='http://fonts.googleapis.com/css?family=Lato&
subset=latin,latin-ext' rel='stylesheet' type='text/css'>
<link rel="stylesheet" href="bower_components/Hover/css/
hover-min.css" />
<link rel="stylesheet" href="styles/myphoto.css" />
<link rel="stylesheet" href="styles/alert.css" />
<link rel="stylesheet" href="styles/carousel.css" />
<link rel="stylesheet" href="styles/a11yhcm.css" />
<link rel="stylesheet" href="bower_components/components-
font-awesome/css/font-awesome.min.css" />
<link rel="stylesheet" href="bower_components/lightbox-for-
bootstrap/css/bootstrap.lightbox.css" />
<link rel="stylesheet" href="bower_components/DataTables/
```

```
media/css/dataTables.bootstrap.min.css" />
<link rel="stylesheet" href="resources/animate/animate.min.css" />
<!-- /build -->
```

Note how we reference the style sheet that is meant to replace the style sheets contained within the special comments on the first line, inside the comment:

```
<!-- build:css styles/myphoto.min.css -->
```

Last, but not least, add the following task:

```
"processhtml": {
  "dist": {
    "files": {
      "dist/index.html": ["src/index.html"]
    }
  }
},
```

Notice how the output of our `processhtml` task will be written to `dist`. Test the newly configured task through the command `grunt processhtml`.

The task should execute without errors:

```
[Benjamins-MacBook-Pro:ch08 benjaminjakobus$ grunt processhtml
Running "processhtml:dist" (processhtml) task
Done, without errors.
```

Figure 8.6: The console output after executing the `processhtml` task

Open `dist/index.html` and observe how, instead of the 12 `link` tags, we only have one:

```
<link rel="stylesheet" href="styles/myphoto.min.css">
```

Next, we need to reconfigure our `uncss` task to write its output to `myphoto.min.css`. To do so, simply replace the output path '`src/styles/output.css`' with '`dist/styles/myphoto.min.css`' inside our `Gruntfile.js` (note how `myphoto.min.css` will now be written to `dist/styles` as opposed to `src/styles`). We then need to add `uncss` to our list of `watch` tasks, taking care to insert the task into the list after `cssmin`:

```
"watch": {
  "target": {
    "files": ["src/styles/myphoto.css"],
    "tasks": ["uncss", "cssmin", "processhtml"],
```

```
        "options": {
          "livereload": true
        }
      }
    }
```

Next, we need to configure our `cssmin` task to use `myphoto.min.css` as input:

```
"cssmin": {
  "target": {
    "files": {
      "dist/styles/myphoto.min.css": [
        "src/styles/myphoto.min.css"
      ]
    }
  }
},
```

Note how we removed '`src/styles!*.min.css`', which would have prevented `cssmin` from reading files ending with the extension `min.css`.

Running `grunt watch` and making a change to our `myphoto.css` file should now trigger the `uncss` task and then the `cssmin` task, resulting in console output indicating the successful execution of all tasks. That is, the console output should indicate that first `uncss`, `cssmin`, and then `processhtml` were successfully executed. Go ahead and check `myphoto.min.css` inside the `dist` folder. You should see how:

- The CSS file contains an aggregation of all our style sheets
- The CSS file is minified
- The CSS file contains no unused style rules

However, you will also notice that the `dist` folder contains none of our assets—neither images, Bower components, nor our custom JavaScript files. As such, you will be forced to copy any assets manually. Of course, this is less than ideal. So let's see how we can copy our assets to our `dist` folder automatically.

The dangers of using UnCSS



UnCSS may cause you to lose styles that are applied dynamically. As such, care should be taken when using this tool. Take a closer look at the `MyPhoto` style sheet and see whether you spot any issues. You should notice that our style rules for overriding the background color of our navigation pills was removed. One potential fix for this is to write a dedicated class for gray nav pills (as opposed to overriding them with the Bootstrap classes).

Deploying assets

To copy our assets from `src` into `dist` we will use `grunt-contrib-copy`. Visit <https://github.com/gruntjs/grunt-contrib-copy> for more on this. Go ahead and install it:

```
sudo npm install grunt-contrib-copy -save-dev
```

Once installed, enable it by adding `grunt.loadNpmTasks('grunt-contrib-copy');` to our `Gruntfile.js`. Then configure the `copy` task:

```
"copy": {  
  "target": {  
    "files": [  
      {  
        "cwd": "src/images",  
        "src": ["*"],  
        "dest": "dist/images/",  
        "expand": true  
      },  
      {  
        "cwd": "src/bower_components",  
        "src": ["*"],  
        "dest": "dist/bower_components/",  
        "expand": true  
      },  
      {  
        "cwd": "src/js",  
        "src": ["*"],  
        "dest": "dist/js/",  
        "expand": true  
      },  
    ]  
  },  
},
```

The preceding configuration should be self-explanatory. We are specifying a list of copy operations to perform; `src` indicates the source and `dest` indicates the destination. The `cwd` variable indicates the current working directory. Note how, instead of a wildcard expression, we could also match a certain `src` pattern. For example, to only copy minified JS files, we could write:

```
"src": ["*.min.js"]
```

Take a look at the following screenshot:

```
[Benjamins-MBP:ch08 benjaminjakobus$ grunt copy
Running "copy:target" (copy) task
Created 86 directories, copied 585 files

Done, without errors.
```

Figure 8.7: The console output indicating the number of copied files and directories after running the copy task

Update the `watch` task:

```
"watch": {
  "target": {
    "files": ['src/styles/myphoto.css'],
    "tasks": ["uncss", "cssmin", "processhtml", "copy"]
  }
},
```

Test the changes by running `grunt watch`. All tasks should execute successfully. The last task that was executed should be the `copy` task.

Stripping CSS comments

Another common source for unnecessary bytes is comments. While needed during development, they serve no practical purpose in production. As such, we can configure our `cssmin` task to strip our CSS files of any comments by simply creating an `options` property and setting its nested `keepSpecialComments` property to 0:

```
"cssmin": {
  "target": {
    "options": {
      "keepSpecialComments": 0
    },
    "files": {
      "dist/src/styles/myphoto.min.css": ["src/styles
        /myphoto.css"]
    }
  }
},
```

Did you know?



You can minify class and identifier names using the lessons learned so far in this chapter. Recall our earlier discussion on class names and identifier names—long names may improve code readability and code maintainability. Short names, on the other hand, require fewer bytes to transfer. As such, developers who want a highly optimized site are caught between two fronts—maintainability versus size. Of course, in most cases, the few extra bytes caused by slightly more descriptive identifier names will not be a cause for major concern. However, it does become a point of consideration once your website reaches a specific volume. Therefore, meet the Grunt class and id minifier at <https://www.npmjs.com/package/grunt-class-id-minifier>. Another useful and alternative tool is Munch at <https://www.npmjs.com/package/munch>.

JavaScript file concatenation

Just as we minified and concatenated our style sheets, we shall now go ahead and minify and concatenate our JavaScript files. Go ahead and take a look at `grunt-contrib-uglify`. Visit <https://github.com/gruntjs/grunt-contrib-uglify> for more.

Install this by typing:

```
sudo npm install grunt-contrib-uglify --save-dev
```

And, as always, enable it by adding `grunt.loadNpmTasks('grunt-contrib-uglify');` to our `Gruntfile.js`. Next, create a new task:

```
"uglify": {  
  "target": {  
    "files": {  
      "dist/src/js/myphoto.min.js": ["src/js/*.js"]  
    }  
  }  
}
```

Running `grunt uglify` should produce the following output:

```
[Benjamins-MacBook-Pro:ch08 benjaminjakobus$ grunt uglify
Running "uglify:target" (uglify) task
>> 1 file created.

Done, without errors.
```

Figure 8.8: The console output after running the uglify task

The folder `dist/js` should now contain a file called `myphoto.min.js`. Open it and verify that the JavaScript code has been minified. As a next step, we need to be sure that our minified JavaScript file will actually be used by our production-ready `index.html`. We will use `grunt-processhtml`, which we installed in the previous section. All that we need to do is wrap our `link` tags inside a special build comment: `<!-- build:js js/myphoto.min.js -->`:

```
<!-- build:js js/myphoto.min.js -->
<script src="js/alert.js"></script>
<script src="js/carousel.js"></script>
<script src="js/a11yhcm.js"></script>
<!-- /build -->
```

Next will add our `uglify` task to our `watch` task list:

```
"watch": {
  "target": {
    "files": ["src/styles/myphoto.css"],
    "tasks": ["uncss", "cssmin", "processhtml", "uglify",
              "copy"],
  }
},
```

Golden rules when working with Grunt

When developing a Grunt file (or any build file for that matter), there are a few practices that you should keep in mind:



- The `dist` folder should not contain any unprocessed source files. `src` is the “unprocessed” code, `dist` is the result of “processing” `src` to create a distributable.
- Typically, you should create a Grunt build task to run the appropriate tasks to create and populate the `dist` directory. You should also create a Grunt `serve` task to run the



appropriate tasks for a development server, with the `watch` task being the final task that is run.

- The `watch` task should be watching all source files. In our case, we watched just one, in order to keep the example simple and easy to understand.

Summary

In this chapter, we touched upon the basics of website optimization, and how to use the build tool Grunt to automate the more common and mundane optimization tasks.

Specifically, we summarized the most important rules that help you to write better, more efficient CSS. We then showed you how to automatically compress and concatenate files, how to deploy assets, and how to strip source files off comments. The aim of these lessons was to provide you with a grounding that would allow you to perform further optimizations, using Grunt, although these are not explicitly covered in depth within this chapter. As such, we encourage you to read the documentation of the various Grunt tasks covered within this chapter. The majority of these tasks are highly customizable and offer many additional benefits not covered here. Knowing how to optimize MyPhoto prior to deployment, we can now move on to our final chapter and learn how to integrate both AngularJS and React with MyPhoto.

9

Integrating with AngularJS and React

At this stage of our journey through mastering Bootstrap, we have built the `MyPhoto` web page using Bootstrap, along with various third-party libraries and plugins, and have optimized the web page. `MyPhoto` is now complete in terms of functionality.

In this chapter, we are not going to develop any new functionality. Instead, we will integrate `MyPhoto` with two of the currently most popular and powerful JavaScript frameworks—AngularJS (<https://angularjs.org/>) and React (<https://facebook.github.io/react/>).

AngularJS is a Model-View-* (MVC, MVV, and so on) JavaScript framework, while React is a JavaScript library which concentrates solely on the View part of the **Model-View-Controller (MVC)** type stack. To readers unfamiliar with the MVC, the term refers to a design pattern whereby the logic for modeling and representing the data, and the logic for creating the bridge between the two are strictly separated. This development approach is extremely powerful, and consequently a vast amount of web pages are built with frameworks or libraries such as AngularJS and React, as they provide very useful abstractions on top of JavaScript and the DOM.

At this point, we will fork `MyPhoto`, creating an AngularJS version and a React version. We will concentrate only on how AngularJS and React can help improve reusability and maintainability, and handle dynamic data. While AngularJS and React have other great functionalities, they are beyond the scope of this book.

In this chapter we will:

- Integrate AngularJS with `MyPhoto`
- Integrate React with `MyPhoto`

Introducing AngularJS

AngularJS is a popular and powerful JavaScript framework created by Google. AngularJS provides easily consumable abstractions on top of JavaScript to aid in the development of web applications. These abstractions include easy-to-use form validation, two-way data binding, custom HTML attributes called *directives* for dynamic data and rendering, a simple interface for **XMLHttpRequest (XHR)**, the ability to create custom directives, single page application routing, and more.

We are not going to cover the intricacies and the vastness of AngularJS, but we will learn how to leverage AngularJS's built-in directives, how to create custom directives and services, and how to use AngularJS's XHR interface.

First, let's add AngularJS to our project.

Setting up AngularJS

The AngularJS team maintains a Bower package with the latest release. Let's install AngularJS. We are going to use version 1.4.8 of AngularJS:

1. In the terminal, from the `src` directory, run:

```
bower install angular#1.4.8
```

2. Create a copy of `src/index.html`, called `src/index-angular.html`. Let's add the minified version of AngularJS into the head of `index-angular.html`:

```
<script src="bower_components/angular/angular.min.js"></script>
```

AngularJS requires a module definition, which is basically your application container, to hook into, so that AngularJS knows which parts of the DOM to execute upon:

1. First, create a file, `src/app/myphoto.module.js`, and add the following module definition:

```
angular.module('MyPhoto', [])
```

The AngularJS Module Definition

```
angular.module('MyName', [])
```

This is the simplest of module definitions. We're creating a new AngularJS module, `MyPhoto`. The square brackets is the definition of dependencies the `MyPhoto` module requires.





This is an array of other modules, which AngularJS will then load via its **Dependency Injection (DI)** system. MyPhoto has no dependencies, so we leave this array empty.

2. Next, add the module definition to the head of the index-angular.html:

```
<script src="bower_components/angular/angular.min.js">
</script>
<script src="app/myphoto.module.js"></script>
```

3. Next, we need to *bootstrap*. In this instance, *bootstrap* means loading the module and hooking it to a part of the DOM, and is not to be confused with the framework that this book is based upon! To do this, we use the **ngApp** AngularJS directive. The ngApp directive will automatically bootstrap the defined module to the element it is attached to, using that element as the root element of the application. We will apply **ng-app** to the body element of index-angular.html:

```
<body ng-app="MyPhoto" data-spy="scroll" data-
target=".navbar" class="animated fadeIn">
```

As you can see, we add the **ng-app** attribute with the value of "MyPhoto", the name we used when defining the module in `myphoto.module.js`. Now, MyPhoto has been bootstrapped with an AngularJS module and is now technically an AngularJS application, although AngularJS doesn't execute or manipulate anything.

Now, let's see how we can leverage core AngularJS features, such as directives, data binding, and JavaScript abstractions to build reusable and dynamic components for MyPhoto.

Improving the testimonials component

In Chapter 7, *Integrating Bootstrap with Third-Party Plugins*, we built a testimonials component to demonstrate the powers of Salvattore, Hover, and Animate.css. When building this component, we hardcoded all the content and introduced a lot of repetition:

```
<div role="tabpanel" class="tab-pane" id="services-testimonials">
  <div class="container">
    <div class="myphoto-testimonial-grid animated fadeIn"
        data-columns>
      <div class="myphoto-testimonial-column hvr-grow-shadow
          hvr-sweep-to-top">
```

```
<h6>Debbie</h6>
<p>Great service! Would recommend to friends!</p>
</div>
<div class="myphoto-testimonial-column hvr-grow-shadow
hvr-sweep-to-top">
    <h6>Anne</h6>
    <p>Really high quality prints!</p>
</div>
<div class="myphoto-testimonial-column hvr-grow-shadow
hvr-sweep-to-top">
    <h6>Oscar</h6>
    <p>Declared their greatness, exhibited greatness.</p>
</div>
<div class="myphoto-testimonial-column hvr-grow-shadow
hvr-sweep-to-top">
    <h6>Joey</h6>
    <p>5 stars! Thanks for the great photos!</p>
</div>
<div class="myphoto-testimonial-column hvr-grow-shadow
hvr-sweep-to-top">
    <h6>Mary</h6>
    <p>Made a stressful event much easier!
        Absolute professionals!</p>
</div>
<div class="myphoto-testimonial-column hvr-grow-shadow
hvr-sweep-to-top">
    <h6>Alice</h6>
    <p>Wonderful! Exactly as I imagined they would
        turn out!</p>
</div>
<div class="myphoto-testimonial-column hvr-grow-shadow
hvr-sweep-to-top">
    <h6>Jack & Jill</h6>
    <p>So happy with how the photos turned out! Thanks
        for capturing the memories of our day!</p>
</div>
<div class="myphoto-testimonial-column hvr-grow-shadow
hvr-sweep-to-top">
    <h6>Nick</h6>
    <p>Perfectly captured the mood of our gig.
        Top notch.</p>
</div>
<div class="myphoto-testimonial-column hvr-grow-shadow
hvr-sweep-to-top">
    <h6>Tony</h6>
    <p>Captured our Cup final win! Great stuff!</p>
</div>
</div>
```

```
</div>
</div>
```

We could drastically improve the maintainability of this component by making the content dynamic and then leveraging AngularJS to recursively add individual testimonials to the DOM.

Let's learn how to load dynamic content using AngularJS.

Making testimonials dynamic

AngularJS provides an abstraction on top of XHR, the `$http` service, with a more usable interface than Vanilla JavaScript, using a Promise-based interface as opposed to Callbacks. A `service` is a singleton object that provides some core functionality across your application, increasing reusability. We can use `$http` to dynamically load data to use in our testimonials component.

It is good practice to use `$http` within an AngularJS service. In other words, any interaction between the application and a server should be wrapped within a `service`. Let's create a `testimonialsService`. Create a file, `src/app/services/testimonials.service.js`, with the following content:

```
angular.module('MyPhoto')
.service('testimonialsService', function($http) {
})
```

Here, we are attaching a new service, `testimonialsService`, to the `MyPhoto` module, and declaring that it has a dependency on the core AngularJS `$http` service. The `testimonialsService` will now be instantiated only when a component within `MyPhoto` depends on it, and that dependency can be declared in the same way as the `$http` service is declared here. Let's add some functionality. We want this `service` to provide a way to load data for the `testimonials` component in a JSON format. Ideally, this would come from a database backed API, but here we will just load it from the filesystem. Let's create a JSON file, `src/data/testimonials.json`, with the data for `testimonials`:

```
[
{
  "name": "Debbie",
  "message": "Great service! Would recommend to friends!"
},
{
  "name": "Anne",
  "message": "Really high quality prints!"
},
```

```
{  
    "name": "Oscar",  
    "message": "Declared their greatness, exhibited greatness."  
,  
{  
    "name": "Joey",  
    "message": "5 stars! Thanks for the great photos!"  
,  
{  
    "name": "Mary",  
    "message": "Made a stressful event much easier!  
Absolute professionals!"  
,  
{  
    "name": "Alice",  
    "message": "Wonderful! Exactly as I imagined they would turn out!"  
,  
{  
    "name": "Jack & Jill",  
    "message": "So happy with how the photos turned  
out! Thanks for capturing the memories of our day!"  
,  
{  
    "name": "Nick",  
    "message": "Perfectly captured the mood of our gig. Top notch."  
,  
{  
    "name": "Tony",  
    "message": "Captured our Cup final win! Great stuff!"  
}  
]
```

With the data in place, let's update `testimonialsService` with a function to retrieve `testimonials.json`:

```
angular.module('MyPhoto')  
.service('testimonialsService', function($http) {  
    function getTestimonials() {  
        $http.get('./data/testimonials.json')  
        .then(  
            function(success) {  
                return success.data  
            },  
            function(error) {  
                return error  
            }  
        )  
    }  
})
```

```
        return {
            getTestimonials: getTestimonials
        }
    })
}
```

Making a Promise with \$q

AngularJS includes a service based on Promises to allow for asynchronous functions, called `$q`. As the `getTestimonials` function includes an asynchronous request, we need to make the function itself asynchronous. To do this, first we add a dependency on `$q` to `testimonialsService`. We then create a `deferred` object, which will `resolve` when the HTTP request succeeds, or `reject` when the request fails. Finally, we return a Promise, which will eventually resolve:

```
angular.module('MyPhoto')
//Declare the service and any dependencies, attaching
it to the MyPhoto module..
.service('testimonialsService', function($http, $q) {
    function getTestimonials() {
        //Create the deferred object
        var deferred = $q.defer()
        //Use $http.get to create a promise to load testimonials.json
        $http.get('/data/testimonials.json')
        //Call the then method of the promise
        .then(
            //Define what happens if the promise returns
            //successfully
            function(success) {
                //Resolve the deferred and return the data
                //property of the success object
                deferred.resolve(success.data)
            },
            //Define what happens if the promise returns an error
            function(error) {
                //Reject the deferred, returning the error value
                deferred.reject(error)
            }
        )
        //Return the deferred promise
        return deferred.promise
    }
    return {
        getTestimonials: getTestimonials
    }
})
})
```

Now, our function returns a Promise, which will resolve to either the data part of our success object, or reject and return the error object. The usage of getTestimonials would now be something like:

```
testimonialsService.getTestimonials()
  .then(
    function(response) {
      console.log(response)
    },
    function(error) {
      console.error(error)
    }
  )
```

What is happening here is self-explanatory. We call the getTestimonials function of testimonialsService. The getTestimonials function has a then property. We pass two functions to then: the first function takes the successful response as a parameter and defines what to do when the Promise resolves; the second function takes the rejected response and defines what to do when the Promise is rejected. Now that we have a service that will return the list of testimonials, let's create an AngularJS directive to render the component.

Creating an AngularJS directive

AngularJS provides an API for extending HTML with custom elements, attributes, comments, and classes. The AngularJS compiler will recognize a custom directive in the DOM and execute a certain specified behavior on the attached element. We are going to build the testimonial's directive using the directive interface. Let's create a new file, src/app/directives/testimonials.directive.js, with the following content:

```
angular.module('myPhoto')
.directive('testimonials', function(testimonialsService) {
  return {
    restrict: 'EA',
    replace: true,
    templateUrl: './app/templates/testimonials.html',
    controller: function($scope) {
    },
    link: function(scope, elem, attr, ctrl) {
    }
  }
})
```

Here, we are adding a new directive—`testimonials`—to the `MyPhoto` module, which has a dependency on `testimonialsService`. Directives return an object with a set of properties that are interpreted by AngularJS. We will touch on a few of them here.

First, we have `restrict: 'EA'`. This means that the directive can be used as either an element or an attribute. For instance, we can use as the directive in either of the following ways:

```
<testimonials></testimonials>
<div testimonials></div>
```

There are two other ways of using a directive—as a class, by adding `C` to the `restrict` property, and as a comment, by adding `M` to the `restrict` property.

Next, we have the `replace` property. By setting this to `true`, the DOM elements generated by the directive will directly replace the DOM element calling it. If `replace` is set to `false`, then the generated elements will be nested within the calling element.

After `replace`, we have the `templateUrl` property. The `templateUrl` is a path to a partial HTML template which will be generated and executed upon by the directive. There is a `template` property also available, to allow for inline HTML in the directive. We are going to store the `testimonials` template in `src/app/templates/testimonials.html`. As `src` will effectively be the root of our deployed application, we will use an absolute path to the application directory.

The `controller` property is next, where we pass in the `$scope` object. The scope in AngularJS represents the data model of the current application, or the current context of the application. The `$scope` model here is exclusive to this instance of the `testimonials` directive, and cannot be manipulated by any other part of the application. The `controller` code is the first to be executed when a directive is instantiated, so makes for the perfect place to gather necessary data or set scope variables for the directive to use.

Finally, we have the `link` function. The `link` function is the last code to be executed during the directive life cycle. The `link` function is executed immediately after the directive template has been added to the DOM, so is perfect for setting event listeners or emitters, or for interacting with third-party scripts. We pass in four variables into the `link` function:

- `scope`: This is a reference to the `$scope` of the directive
- `elem`: This is a reference to the rendered DOM element
- `attr`: This is a reference to the attributes of the element
- `ctrl`: This is a reference to the previously defined controller

The variable names are unimportant here, they can be anything, but these names are pretty standard.

This is just a skeleton of a directive. AngularJS directives have many more features and intricacies than described here, and this example is just one way of writing a directive; there are many other styles. For the purposes of this example, the form of this directive is perfect.

We want the `testimonials` directive to render the `testimonials` component. To do that, it will need a list of said testimonials. In the `controller` function, we can use `testimonialsService` to retrieve the list:

```
.directive('testimonials', function(testimonialsService) {
    return {
        restrict: 'EA',
        replace: true,
        templateUrl: './app/templates/testimonials.html',
        controller: function($scope) {
            testimonialsService.getTestimonials()
                .then(function(response) {
                    $scope.testimonials = response
                }, function(error) {
                    console.error(error)
                })
        },
        link: function(scope, elem, attr, ctrl) {
        }
    }
})
```

Writing the `testimonials` template

In the `controller` function, we call `testimonialsService.getTestimonials`. When `getTestimonials` resolves, we create a scope variable, `testimonials`, with the value of the response. If the Promise does not resolve, we output an error to the console. With this, our directive has a list of testimonials before it renders, as the `controller` is the first step of the directive life cycle. Now, let's write the `testimonials` template.

Create `src/app/templates/testimonials.html` with the following content:

```
<div class="myphoto-testimonial-grid animated fadeIn" data-columns>
    <div ng-repeat="testimonial in testimonials track by $index"
        class="myphoto-testimonial-column hvr-grow-shadow hvr-sweep-to
        -top">
        <h6>{{testimonial.name}}</h6>
        <p>{{testimonial.message}}</p>
```

```
</div>
</div>
```

That's it. Compare this to the hard coded version and notice the difference in the amount of HTML we wrote. So, what is going on here? Well, we took the raw HTML for the testimonial component and removed the individual testimonial elements. We then added a new attribute, `ng-repeat`, to the `myphoto-testimonials-column` div element. The `ng-repeat` attribute is actually an AngularJS directive itself. The `ng-repeat` attribute loops through the data passed to it, repeatedly adding the element which is an attribute of the DOM. We give `ng-repeat` the value of "`testimonial` in `testimonials` track by `$index`". Simply, we are saying repeat this element for every entry in the `testimonials` property of the directive's scope, giving each value the reference `testimonial`. We are also telling `ng-repeat` to track each entry by `$index`, which is the position of the entry in `testimonials`. Using `track by` has great performance benefits for `ng-repeat`. Without `track by`, AngularJS will only identify the entries by its own built-in unique identifier, `$id`. If the data used for the entries is reloaded, AngularJS will recreate each DOM element in the list again. Using `track by $index` allows AngularJS to just reuse the entries, as it now knows which DOM elements need to be recreated and which can be reused. One caveat with using `$index` for tracking is that AngularJS will expect the reloaded data to be in the same order. You can use any property of the entry with `track by`. For example, if each object in `testimonials.json` had an `id` property, we could use `track by testimonial.id`. Within the `myphoto-testimonial-column` div, we create a `h6` and `p` element, just like the raw HTML testimonial markup. Instead of hard coding values, we use the reference to the entries in the `testimonials` array, `testimonial`, provided by `ng-repeat`. Using `testimonial` along with handlebar notation, we can access the properties of each entry as `ng-repeat` loops through `testimonials`. As we loop through, AngularJS will execute on the handlebar notation, replacing them with the correct values.

Testing the testimonial directive

Let's test the `testimonials` directive out. First, add `testimonials.service.js` and `testimonials.directive.js` to the head of `index-angular.html`:

```
<script src="app/services/testimonials.service.js"></script>
<script src="app/directives/testimonials.directive.js"></script>
```

Next, replace the markup for the `testimonials` component with the directive markup. We will use the attribute form of the `testimonials` directive, as an attribute of a `div` element:

```
<div role="tabpanel" class="tab-pane" id="services-testimonials">
  <div class="container">
    <div testimonials></div>
```

```
</div>  
</div>
```

With that in place, AngularJS will replace this element with the template defined in `testimonials.directive`, and with the testimonials from `testimonials.json`, served by `testimonialsService.getTestimonials`. Let's check it out:

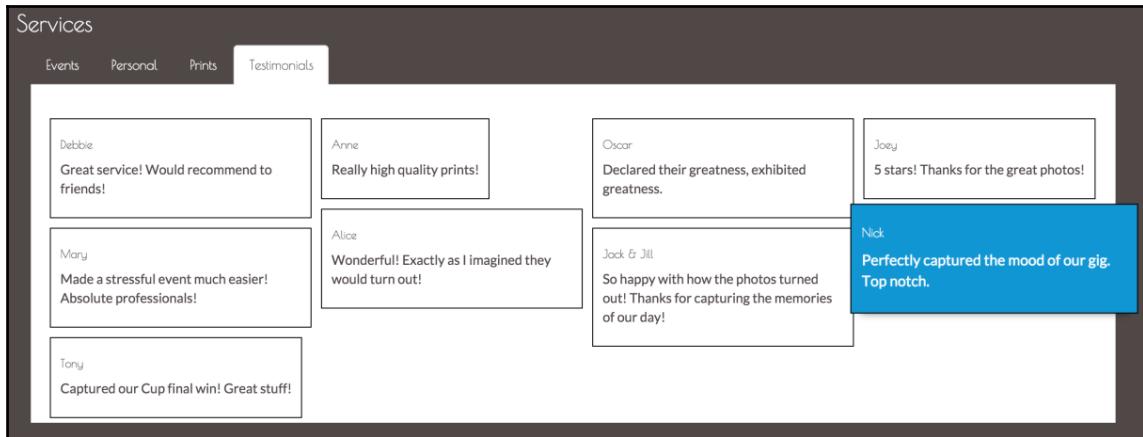


Figure 9.1: The improved testimonials section, displaying testimonials dynamically

Awesome! We now have a dynamic **Testimonials** tab, thanks to AngularJS. Something is not right here, though. Salvatorre, the dynamic grid library we introduced in Chapter 7, *Integrating Bootstrap with Third-Party Plugins* does not seem to be taking effect on this component anymore.

The reason for this is simple—by the time AngularJS has rendered the `testimonials` component, Salvatorre has already instrumented the DOM.

Importing the Salvatorre library

We need to register the `testimonials` component with Salvatorre after it has rendered. We can do this through the `link` function. First, let's add `$timeout` service as a dependency:

```
.directive('testimonials', function(testimonialsService, $timeout)
```

The `$timeout` service is the AngularJS wrapper for the `window.setTimeout` function. As you may know, AngularJS works on a digest cycle, where it uses dirty-checking techniques to see which parts of the application needs to be updated. This happens routinely, or can be forced. We can use `$timeout` to ensure that certain code is executed in a later digest cycle.

Let's update the `link` function with the following:

```
link: function(scope, elem, attr, ctrl) {
    $timeout(function() {
        salvattore.registerGrid(elem[0])
    }, 1000)
}
```

Here, we are using `$timeout` with two parameters. The latter parameter is a delay of 10 milliseconds, to ensure the code is executed in a later digest cycle; 1,000 milliseconds should be enough to ensure the `testimonial` component has completed rendering. We pass in a function as the first parameter, responsible for calling Salvattore's `registerGrid` function. The `registerGrid` function forcibly instruments the passed element with Salvattore. We pass the first element in the `elem` array, which is the rendered `testimonial` component. With this in place, the **Testimonial** tab will have a dynamic grid layout.

As such, we have managed to replicate the **Testimonial** tab—which leverages Bootstrap, Salvattore, Hover, and Animate.css—through AngularJS services and directives, using dynamic content instead of hardcoded values. Time to move onto React.

Introducing React

React is a JavaScript library created by Facebook. While AngularJS positions itself as a framework, React is very clear in its position as a library. React prides itself on being responsible for the visual aspect of the application, “*Just the UI*”, as the React landing page professes. By concerning itself modularly with this single aspect, React is relatively small in size compared to AngularJS and other one-stop-shop frameworks.

React employs a modular approach to the UI, with the idea of components. Components are similar to the directives we used with AngularJS and to the idea of web components. That is, components are reusable pieces of UI functionality, adhering to the “*do one thing, do one thing well*” ideology. React really pushes the modular approach in how components are usually composed, with tight coupling between styles, HTML, and JavaScript.

Typically, all component specific code is contained within one file. The HTML, the CSS rules, and the JS logic are all included within this file. While at first glance, this approach arguably flies in the face of the approach of separation of concerns, it does totally separate the concerns of components from each other. In other words, making changes to one part of the application should have no impact on another.

React is famously fast when manipulating the DOM, using the virtual DOM approach to figuring out which parts of the UI to update, as opposed to the dirty-checking technique employed by AngularJS. The virtual DOM is essentially the idea of keeping a copy of the real DOM in memory, and updating the copy with necessary changes. The virtual DOM is then periodically compared with the *real* DOM; any differences then result in that specific piece of the DOM being re-evaluated and re-rendered.

React also promotes the usage of JSX with React applications. JSX is a programming language which compiles into JavaScript, thus requiring a compilation step in the development process. JSX offers a layer of object-oriented style programming on top of JavaScript, such as Java-like class systems and static typing.

Now, let's set up React.

Setting up React

There are several ways of getting setup with React. As the React team maintain a Bower package, we will use Bower as we have done throughout this book. From the terminal, let's pull down React through Bower. We are going to use 0.14.6 version of React:

```
bower install react#0.14.6
```

With that, we have a downloaded React to `src/bower_components/react`. Here you will see `react.js`, `react-dom.js`, and `react-dom-server.js`, along with their minified versions. Here `react.js` is the core React library, `react-dom.js` takes responsibility for the actual rendering of the React components in the DOM, and `react-dom-server.js` allows for server-side rendering of React components.

Create a copy of `src/index.html` to `src/index-react.html`, and add the minified versions of React and ReactDOM to the head of the page:

```
<script src="bower_components/react/react.min.js"></script>
<script src="bower_components/react/react-dom.min.js"></script>
```

We also need to include Babel, a JavaScript compiler which caters for JSX. Babel does not maintain a Bower package, but it is available on npm. However, Babel maintains a version of its library for browsers on a CDN. Include the following in the head of our page:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/
5.8.23/browser.min.js"></script>
```

The browser.min.js file will transform any JSX code in HTML, within `script` tags with a `type` attribute with the value `text/babel`. Let's test it out, to make sure we have everything set up correctly. Above our footer, add a `div` with an `id` of `test`:

```
<div id="test"></div>
```

Next, let's write the simplest of React components. At the bottom of `index-react.js`, after the `footer` element, include the following:

```
<script type="text/babel">
  ReactDOM.render(
    <div className='container-fluid myphoto-section bg-myphoto-dark'>Test</div>,
    document.getElementById('test')
  );
</script>
```

Let's walk through what is happening here. First, as we said before, our `script` tag needs a `type` attribute with the value `text/babel` so that Babel knows to compile it into JavaScript before execution. Within the `script` tags, we have our first real interaction with React: `ReactDOM` and its `render` function. The `render` function takes two arguments here: the first is raw HTML, the second is an element selector. What is happening here is pretty self-explanatory: we want React to find the `test` element, and replace it with the HTML we passed as the first parameter. You may notice that in our HTML, we have a `className` attribute. We use `className` instead of `class`, as `class` is a reserved word in JavaScript. The `className` attribute is converted to `class` when the component is rendered in the DOM. Open `index-react.html` and check if we now have a **Test** section in our page:

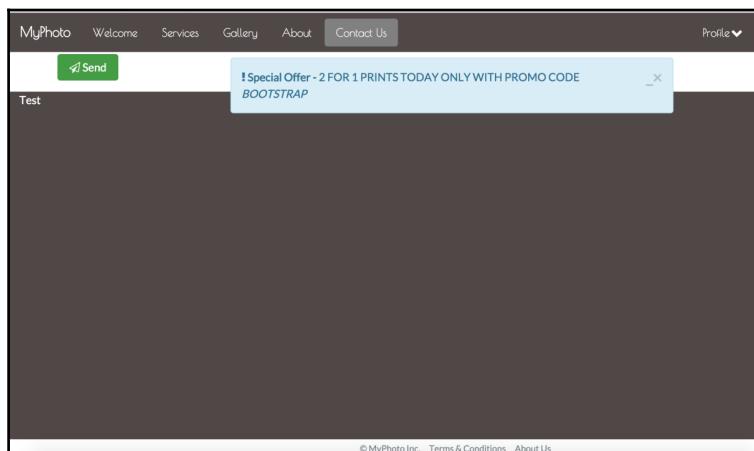


Figure 9.2: The Test section displaying as expected

We are now all set up to integrate React into MyPhoto. Let's do something more substantial. We are going to convert the carousel in the **Gallery** component into a React component. Before we get to that, remove the `test` component, both the `div` and the `JSX` we added.

Making a Gallery component in React

To understand how we can integrate React with our application, we are going to make a reusable **Gallery** component using React. The first thing we are going to do is create a new file, `src/app/components/gallery.js`, and include it in the head of `index-react.html`:

```
<script type="text/babel" src="app/components/gallery.js"></script>
```

Notice the `type` attribute, again set to `text/babel`, so `gallery.js` will be compiled to JavaScript at runtime. As you can imagine, this is a slow operation. This method is recommended to be used only for development purposes. For production, all `JSX` should be precompiled into JavaScript. For the purposes of this example, we will continue with the runtime-compile method.

Let's add some code to `gallery.js`. Take the `gallery` markup, the `div` element with the `id` `gallery-carousel` and its nested elements, and add it to `gallery.js` as the first argument for `ReactDOM.render`. Make sure to also change the `class` attributes to `className`. We also need to make sure that we use handlebar expressions when defining inline styles, as React parses the `style` attribute as an object, rather than a string:

```
ReactDOM.render(
  <div id="gallery-carousel" className="carousel slide"
    data-ride="carousel" data-interval="3000">
    <div className="carousel-inner" role="listbox">
      <div style={{height: 400px}} className="carousel-item"
        active>
        
        <div className="carousel-caption">
          Brazil
        </div>
      </div>
      <div style={{height: 400px}} className="carousel-item">
        
        <div className="carousel-caption">
          Datsun 260Z
        </div>
      </div>
    </div>
  </div>
)
```

```
<div style={{height: 400px}} className="carousel-item">
  
  <div className="carousel-caption">
    Skydive
  </div>
</div>
<a className="left carousel-control" href="#gallery-
carousel" role="button" data-slide="prev">
  <span className="icon-prev" aria-hidden="true"></span>
</a>
<a className="right carousel-control" href="#gallery-
carousel" role="button" data-slide="next">
  <span className="icon-next" aria-hidden="true"></span>
</a>
<ol className="carousel-indicators">
  <li data-target="#gallery-carousel" data-slide-to="0"
    className="active"></li>
  <li data-target="#gallery-carousel" data-slide-to="1"></li>
  <li data-target="#gallery-carousel" data-slide-to="2"></li>
</ol>
</div>,
  document.getElementById('react-gallery')
)
```

The second argument we pass is an element selector, targeting an element with an `id` of `react-gallery`. Replace the `gallery-carousel` element in `index-react.js` with the target element for the `gallery.js` React component:

```
<div id="react-gallery"></div>
```

Open `index-react.js` and we should see the **Gallery** component, but this time it is being generated by React:



Figure 9.3: The React Gallery component

Oh, that isn't what we want. Obviously something has gone wrong here. Let's check out the browser's Developer Console to see if there are any errors reported:



```
① ► Uncaught SyntaxError: http://localhost:8000/app/components/gallery.js: Expected corresponding JSX closing tag for <img> (9:6) browser.min.js:41
7 |         Brazil
8 |     </div>
> 9 |     </div>
   |     ^
10 |     <div className="item md">
11 |         
12 |         <div className="carousel-caption">
```

Figure 9.4: Chrome's Developer Console displaying errors

So, Babel has thankfully given us an explicit error, it is expecting a closing tag for the `img` tags in `gallery.js`. Let's make sure all our `img` tags are closed in `gallery.js`:

```
<div style={{height: 400px}} className="carousel-item active">
    
    <div className="carousel-caption">
        Brazil
    </div>
</div>
<div style={{height: 400px}} className="carousel-item">
    
    <div className="carousel-caption">
        Datsun 260Z
    </div>
</div>
<div style={{height: 400px}} className="carousel-item">
    
    <div className="carousel-caption">
        Skydive
    </div>
</div>
```

Let's give `index-react.js` another go and we should now have our React-powered **Gallery** tab:



Figure 9.5: The functioning React Gallery component displaying an image of the Botanical Garden in Rio de Janeiro

Great! We now have a functioning React component. But, it isn't exactly reusable in terms of a carousel. If we wanted another carousel elsewhere, we would need to create another component. Let's make the carousel reusable by passing in options to the React component.

Using carousel in React

To write a reusable component like this, we create the component as a custom React class. This class essentially returns the markup to be rendered by `ReactDOM.render`, but gives us more power to manipulate our template. Like AngularJS directive custom React classes are extensions of the DOM, allowing us to create new DOM tags. For example, we could create a `Carousel` element. Note that custom React classes always begin with an uppercase letter:

```
<Carousel></Carousel>
```

Before we do anything, let's analyze the component we have and understand which values we want to make mutable. In the root element, the `id` and `data-interval` values need to be changeable. The component also needs to allow the images and caption to be set. Ideally, this should be passed to the component as an array. Finally, the component needs to take in the value for `data-modal-picture`.

In all, the component needs to take four values. So, the markup for the component would look something like:

```
<Carousel id=<value> interval=<value> carousel-modal-picture=<value> carousel-images=<[images]></Carousel>
```

In `gallery.js`, we can access component attributes using `this.props`. Wrapping `this.props` with curly braces allows these attribute values to be accessed within the markup of the component code. Add the following to the beginning of `gallery.js`:

```
var Carousel = React.createClass({
    render: function () {
        var props = this.props
        return (
            <div id={props.id} className="carousel slide"
                data-ride="carousel" data-interval={props.interval}>
                <div className="carousel-inner" role="listbox">
                    { props.images.map(function(item, index) {
                        var itemClass;
                        if (index === 0)
                            itemClass = "active"
                        else
                            itemClass = ""
                        return (
                            <div className={'carousel-item ' + itemClass}>
                                <img data-modal-picture={'#' +
                                    props.carouselModalPicture}
                                    src={item.src} />
                                <div className="carousel-caption">
                                    {item.caption}
                                </div>
                            </div>
                        )
                    })
                }
            </div>
            <a className="left carousel-control" href={'#' + props.id}
                role="button" data-slide="prev">
                <span className="icon-prev" aria-hidden="true"></span>
            </a>
            <a className="right carousel-control" href={'#' + props.id}
                role="button" data-slide="next">
                <span className="icon-next" aria-hidden="true"></span>
            </a>
            <ol className="carousel-indicators">
                { props.images.map(function(item, index) {
                    var liClass;
```

```
        if (index === 0)
            liClass = "active"
        else
            liClass = ""
        return (
            <li data-target={'#' + props.id} data-slide-to={index} className={ liClass }></li>
        )
    })
}
</ol>
</div>
)
}
})
```

We have created a new React class using `React.createClass`, which has a `render` property. The `render` property is simply a function which returns an HTML template. The template is essentially the markup for the `gallery-carousel` component, except we are accessing some dynamic properties. We have replaced all references to the `carousel id` with `this.props.id`, all references to the `id` of the modal window to open up the `carousel modal` to `this.props.carouselModalPicture`, and the `data-interval` to `this.props.interval`. We will come back to the images and the captions later. We assign this custom React class to the variable `Carousel`.

Now that we have this reusable class, we no longer need the template within the `ReactDOM.render` function. Replace the function with the following:

```
ReactDOM.render(
    <Carousel id="gallery-carousel" interval="3000"
    carouselModalPicture="carouselModal"></Carousel>,
    document.getElementById('react-gallery')
)
```

We are now using the `Carousel` tag in the `render` method. We are passing three attributes to `Carousel` – `id`, `interval`, and `carouselModalPicture`. The values of these attributes will then be used in the template returned by `Carousel.render`. `ReactDOM.render` will then replace the `react-gallery` element in `index-react.js` with the `carousel` template, with these defined attributes. Check it out and see if we have a fully functioning `carousel` in the **Gallery** tab. Change some of the attribute values and see if the `carousel` component works as expected.

Now, let's put the images and image captions in as an option. In reality, these values would come from an API, or will be otherwise dynamically generated. For the sake of this example, we are going to create a variable with the array of values. To demonstrate that the values are being passed through as an attribute, we will change the captions slightly. Add the following to the beginning of `gallery.js`:

```
var carouselImages = [
  {
    src: "images/brazil.png",
    caption: "Lake in Brazil"
  },
  {
    src: "images/datsun.png",
    caption: "Datsun Fairlady Z"
  },
  {
    src: "images/skydive.png",
    caption: "Team Skydive"
  }
]
```

Now, we can pass `carouselImages` as an attribute of the `Carousel` tag:

```
<Carousel id="gallery-carousel" interval="3000"
carouselModalPicture="carouselModal" images={carouselImages}>
</Carousel>
```

In the `carousel` template, we need to loop through the data passed into the `images` attribute, and create a new slide for each entry, as well as a new indicator in the `carousel-indicators` list. We will loop through the dataset using the `map` function. As `map` creates a closure, we first need to create a reference to `this.props`, as `this` will be different in the context of the closure. At the beginning of the `render` function, assign `this.props` to `props`:

```
var props = this.props
```

Next, remove the slides from the `carousel-inner` element and add the following:

```
{ props.images.map(function(item, index) {
  var itemClass;
  if (index === 0) {
    itemClass = "active"
  } else {
    itemClass = ""
  }
  return (
    <div key={index} class={itemClass}>
      <img alt={item.caption} src={item.src} />
    </div>
  )
})}
```

```
<div className={ 'item md ' + itemClass }>
  <img data-modal-picture={'#' + props.carouselModalPicture}
       src={item.src} />
  <div className="carousel-caption">
    {item.caption}
  </div>
</div>
)
}
})}
```

We are looping through `props.images` using the `map` function. We want to set the first slide in the array to be the active slide, so we check its `index` and assign the `itemClass` variable accordingly. We then define the template for the slide. We pass `itemClass` into the `className` attribute to denote the initially active slide, we then use the `item.src` property as the `src` of the `img` element, and `item.caption` as the caption of the slide. Next, remove all the list items from the `carousel-indicators` list, replacing them with the following:

```
{ props.images.map(function(item, index) {
  var liClass;
  if (index === 0) {
    liClass = "active"
  } else {
    liClass = ""
  }
  return (
    <li data-target={'#' + props.id} data-slide-to
        ={index} className={ liClass }></li>
  )
})
}
```

Similarly, we loop through the images and assign the first slide as the active slide. That is everything our component needs to create our **Gallery** carousel. Let's check it out:

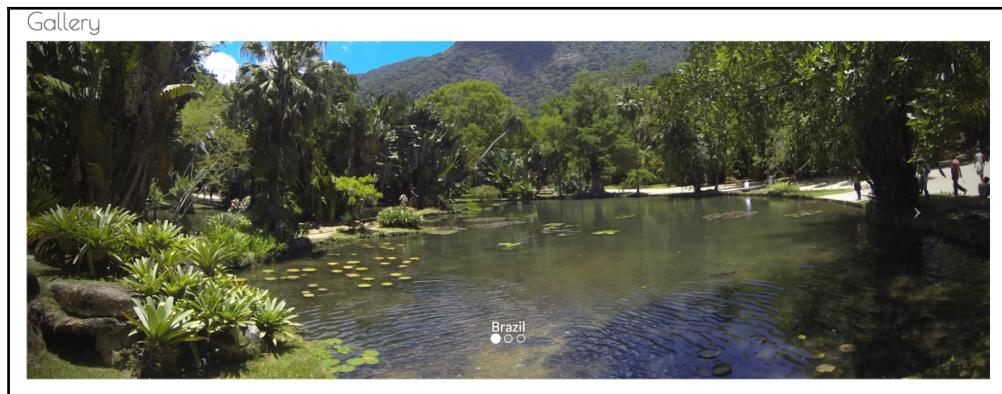


Figure 9.6: The functioning React Gallery component displaying an image of the Botanical Garden in Rio de Janeiro

As you can see from the caption, the carousel is loading from the `imagesCarousel` array. Now, we have a customizable, reusable, and easily maintainable React-powered `carousel` component that can be used anywhere across `MyPhoto`.

Summary

In this chapter, we have learned how to integrate AngularJS into our Bootstrap site. We are now able to load data dynamically, pass it into a reusable directive, and render a component which mixes AngularJS and Bootstrap, along with other third-party libraries.

We have also seen how we can leverage React to easily transform our static content into reusable, customizable, and maintainable components.

Being able to integrate frameworks and libraries such as AngularJS and React into a Bootstrap driven website is critical when building dynamic and useful user interfaces. Bootstrap complements, and is complemented by, dynamically driven content, and this chapter has hopefully given you the understanding you need to be able to use these tools to build powerful and useful websites.

Bibliography

This Learning Path is a blend of content, all packaged up keeping your journey in mind.
It includes content from the following Packt products:

- ▶ Bootstrap 4 by Example, Silvio Moreto
- ▶ Learning Bootstrap 4, Second Edition, Matt Lambert
- ▶ Mastering Bootstrap 4, Benjamin Jakobus & Jason Marah



Thank you for buying Bootstrap 4 – Responsive Web Design

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 that focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike.

For more information, please visit our website at www.packtpub.com.

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, then please 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.

Please check www.PacktPub.com for information on our titles

