Learning Web Development with Bootstrap and AngularJS

If you're ready to progress your web development skills past HTML and CSS, this book will give you everything you need to build your own web app. You'll be taught the basics of Bootstrap and AngularJS, taking an in-depth look at what makes up each framework. After you've mastered the ideology, you'll put it into action as you build a contact manager throughout the course of the book.

We'll look at filters, routing, and views for your app and will teach you to utilize CRUD (Create, Read, Update, and Delete) to develop further. You'll learn how to connect your app to the server, customize Bootstrap, and build a directive. Along the way we'll touch on additional technologies such as Less, gulp, Grunt, and AngularStrap. Finally, we'll take a look at potential stumbling blocks and will give you tips to avoid them.

With this step-by-step guide packed with plenty of screenshots, you'll be building web apps in no time.

Who this book is written for

Whether you know a little about Bootstrap or AngularJS, or you're a complete beginner, this book will enhance your capabilities in both frameworks and you'll build a fully functional web app. A working knowledge of HTML, CSS, and JavaScript is required to fully get to grips with Bootstrap and AngularJS.

What you will learn from this book

- Explore Angular's built-in filters and create your own
- Create a multi-view web app using AngularJS's router and templating system
- Integrate AngularStrap to utilize the many plugins originally created by the Bootstrap team
- Customize Bootstrap using Less to create a unique theme
- Maintain the codebase of your app using the gulp and Grunt.js task-runner tools
- Discover the meaning of CRUD and how to implement it
- Troubleshoot any issues you may encounter while developing


$ 49.99 US
£ 32.99 UK

Prices do not include local sales tax or VAT

Stephen Radford

Learning Web Development with Bootstrap and AngularJS

Build your own web app with Bootstrap and AngularJS, utilizing the latest web technologies
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Hello, {{name}}'
- A synopsis of the book’s content
- More information on Learning Web Development with Bootstrap and AngularJS

About the Author

Stephen Radford is a full-stack web developer based in the heart of Leicester, England. Originally from Bristol, Stephen moved to Leicester after studying graphic design in college to accept a job at one of UK's largest online marketing companies.

While working at a number of agencies, Stephen developed several side projects, including FTPloy, a SaaS designed to make continuous deployment available to everyone. The project was subsequently a finalist in the .Net Awards Side Project of the Year category.

He and his business partner now run Cocoon, a web development company that builds and maintains web apps such as FTPloy and Former. Cocoon also works closely with a handful of startups and businesses to develop ideas into websites and apps.
Learning Web Development with Bootstrap and AngularJS

I've worked on projects of various sizes over the course of my career, ranging from small brochure sites to building entire social networks. One thing in common with all of them was the need for well-structured JavaScript and CSS.

This book covers two fantastic open source projects that stemmed from this need—Bootstrap and AngularJS.

What This Book Covers

Chapter 1, Hello, {{name}}, looks at the basics of AngularJS and Bootstrap whilst building a simple "Hello, World" app.

Chapter 2, Let's Build with AngularJS and Bootstrap, introduces to the main app we'll be building over the course of the book, a look at Bootstrap's grid system, and some of the components that make up AngularJS.

Chapter 3, Filters, takes a look at some of AngularJS's built-in filters and also build our own.

Chapter 4, Routing, uses AngularJS's built-in router, and we'll learn how to utilize partials to create a multiview web app.

Chapter 5, Building Views, covers Bootstrap's grid system, and we'll flesh out the partials.

Chapter 6, CRUD, shows that our views are in place we can implement create, read, update, and delete functions.

Chapter 7, AngularStrap, covers the third-party module, which will allow us to use all of Bootstrap's plugins via AngularJS.

Chapter 8, Connecting to the Server, looks at the two official ways of connecting to a server.

Chapter 9, Using Task Runners, minifies all of our JS and Less files using Grunt and gulp.

Chapter 10, Customizing Bootstrap, allows you to easily customize Bootstrap now that Grunt.js is setup.
Chapter 11, Validation, includes validation out of the box; we'll implement that and manage server errors.

Chapter 12, Community Tools, takes a look at some of the tools built by the AngularJS community.

Appendix A, People and Projects to Watch, covers some key people in the AngularJS and Bootstrap worlds as well as the projects to watch.

Appendix B, Where to Go for Help, provides answers to the questions you might have.

Appendix C, Self-test Answers, provides all the answers enlisted in the self-test questions sections of the book.
Hello, {{name}}

The best way to learn code is to write code, so we're going to jump right in. To show you just how easy it is to get up and running with Bootstrap and AngularJS, we're going to make a super simple application that will allow us to enter a name and have it displayed on the page in real time. It's going to demonstrate the power of Angular's two-way data binding and the included templating language. We'll use Bootstrap to give the app a bit of style and structure.

Before we install our frameworks, we need to create our folder structure and the index.html file that will be the basis of our web app.

**Setting up**

In order to create our Angular and Bootstrap application, we need to do a little bit of setting up, which just involves creating an HTML page and including a few files. First, create a new directory called `chapter1` and open it up in your editor. Create a new file called `index.html` directly inside it and pop in this boilerplate code:

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title></title>
</head>
<body>
</body>
</html>
```

This is just a standard HTML page that we can do something with once we've included Angular and Bootstrap.
Now, create a couple of folders inside chapter1 folder: css and js. Your completed folder structure should look something like this:

![Folder Structure](image)

**Installing AngularJS and Bootstrap**

Installing both of our frameworks is as simple as including a CSS or JavaScript file on our page. We can do this via a content delivery network (CDN) like Google Code or MaxCDN, but we're going to fetch the files manually for now. Let's take a look at what steps you should be aware of when including AngularJS and Bootstrap in your project.

**Installing Bootstrap**

Head to [http://getbootstrap.com](http://getbootstrap.com) and hit the Download Bootstrap button. This will give you a zip of the latest version of Bootstrap that includes CSS, fonts, and JavaScript files. Previous versions of Bootstrap included an images directory but Version 3 brings the change to icon fonts.

For our app, we're only interested in one file at the moment: bootstrap.min.css from the css directory. The stylesheet provides the entire structure and all of the lovely elements, such as buttons and alerts, that Bootstrap is famous for. Copy it over to your project's css directory and open up the index.html file in your text editor.
Including Bootstrap is as easy as linking that CSS file we just copied over. You just need to add the following within your `<head>` tag. Pop this script tag within the `<head>` of your page:

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

## Installing AngularJS

Okay, now that we've got Bootstrap included in our web app, we need to install Angular. Visit [https://angularjs.org/](https://angularjs.org/) and click on the **Download** button. You'll be presented with a few options; we want the minified stable version.

Copy the downloaded file over to your project's `js` directory and open up your `index.html` file. Angular can be included in your app just like any other JavaScript file.

It's recommended that Angular is included in the `<head>` tag of your page or certain functions we'll be taking advantage of throughout the course of the book won't work. While it's not necessary, there will be extra steps you'll need to take if you choose to load Angular further down your HTML file.

Pop this `<script>` tag within the `<head>` of your page.

```html
<script src="js/angular.min.js"></script>
```

Ready to go? Well, almost. We need to tell Angular that we want to utilize it in our app. Angular calls this bootstrapping and the framework makes this extremely simple for us. All we need to do is include an additional attribute in our opening `<html>` tag:

```html
<html lang="en" ng-app>
```

That's it! Angular now knows we want to take advantage of it.

> Angular also allows us to prefix these attributes with `data-` (for example, `data-ng-app`) should we be concerned about writing valid HTML5.

## Using AngularJS

So we've got a lot of the theory behind Angular down; it's time to actually put it in place. Once we've got our application working, we'll take a look at how we can make it shine with Bootstrap.
Let's open that `index.html` file again, but this time also open it up in your browser so we can see what we're working with. This is what we've got so far:

```html
<html lang="en" ng-app>
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" href="css/bootstrap.min.css">
  <title></title>
  <script type="text/javascript" src="js/angular.min.js"></script>
</head>
<body>
  
</body>
</html>
```

So, we've got Bootstrap and Angular there and we've initialized our app with the `ng-app` attribute in the opening `<html>` tag; let's get cracking.

We're going to have a Hello, World app with a bit of a difference. Instead of saying hello to the world, we're going to have an input field that will bind the data and echo it out in our view automatically, and we're going to do all of this without writing a line of JavaScript.

Let's start out by getting an `<h1>` tag in our `<body>` tag:

```
<h1>Hello, World</h1>
```

If you view this in your browser, you should notice that Bootstrap has tidied up the default. We no longer have Times New Roman but instead Helvetica and those excess margins around the edge have been removed:
We now need to include our text input and also specify the model we want to use. Remember, a model can be any type, but in this case it will be a string that the input will return:

```html
<input type="text" ng-model="name">
```

The `ng-model` attribute declares a model binding on that element, and anything we type into the input box will be automatically bound to it by Angular. Obviously this isn't going to be displayed on our page by magic; we need to tell the framework where we want it echoed. To display our model on the page, we just need to wrap the name of it in double curly braces:

```html
{{name}}
```

Pop this in place of `World` in your `<h1>` tag and refresh the page in your browser. If you pop your name in the input field, you'll notice that it's automatically displayed in your heading in real time. Angular is doing all of this for us and we haven't written a single line of JavaScript.

Now, while that's great, it would be nice if we could have a default in place so it doesn't look broken before a user has entered their name. What's awesome is that everything in between those curly braces is parsed as an AngularJS expression, so we can check and see if the model has a value, and if not, it can echo `World`. Angular calls this an expression and it's just a case of adding two pipe symbols as we would in JS:

```html
{{name || 'World'}}
```
Angular describes an expression as the following:
"JavaScript-like code snippets that are usually placed in bindings such as {{ expression }}."

It's good to remember that this is JavaScript, and that's why we need to include the quotation marks here, to let it know that this is a string and not the name of a model. Remove them and you'll notice that Angular displays nothing again. That's because both the name and World models are undefined.

These models can be defined directly from within our HTML using an attribute as we've seen, but we can also assign a value to them from a controller. To do this, we're going to need to create a new JS file called controller.js and include it in our app:

```html
<script type="text/javascript" src="js/controller.js"></script>
```

Pop this in after you've included Angular on your page to avoid any errors being thrown.

Controllers are just functions that Angular can utilize; let's take a look at one:

```javascript
function AppCtrl($scope){
}
```

Here, we've declared our controller (essentially just a plain JavaScript constructor function) and have injected the scope service into it. The scope is what we can access from within our view. There can be multiple controllers and multiple scopes on a single page. It's essentially a JavaScript object of our models and functions that Angular works its magic with, for example, the scope of our application so far looks like this:

```javascript
{
    name: "Stephen"
}
```

The scope changes depending upon what's entered into the input field. This can then be accessed from our view as well as the controller.

Now that we've created our controller, we need to tell Angular we want to use it. For our application we only need a single controller, so let's add a second attribute to the <html> tag again:

```html
ng-controller="AppCtrl"
```
This attribute tells Angular we want to use the `AppCtrl` function we've just created as our controller for the page. We could of course add this to any element on the page including the body if we so wish.

To check everything's working okay, we're going to specify an initial value for our model. This is as easy as setting a property on any object:

```javascript
function AppCtrl($scope) {
  $scope.name = "World";
}
```

If you refresh your app in your browser, you'll notice that `World` is now pre-filled as the model's value. This is a great example of Angular's powerful **two-way data binding**. It allows us to use pre-defined data perhaps from an API or database and then change this in the view directly before picking it back up in the controller.

Angular describes data binding as "the automatic synchronization of data between the model and view components". Two-way data binding means that if you change the value of a model in your view or in your JavaScript controller, everything will be kept up-to-date.

**Bootstrap**

Now that we've created our Hello World application and everything is working as expected, it's time to get involved with Bootstrap and add a bit of style and structure to our app.

The application is currently misaligned to the left, and everything is looking cramped so let's sort that out first with a bit of scaffolding. Bootstrap comes with a great **mobile first** responsive grid system that we can utilize with the inclusion of a few divs and classes. First though, let's get a container around our content to clean it up immediately:

```html
<div class="container">
  <h1>Hello, {{name || 'World'}}</h1>
  <input type="text" ng-model="name">
</div>
```

Mobile first is a way of designing/developing for the smallest screens first and adding to the design rather than taking elements away.
Hello, {{name}}

If you resize your browser window, you should start to notice some of the responsiveness of the framework coming through and see it collapsing:

Now, I think it's a good idea to wrap this in what Bootstrap calls a Jumbotron (in previous versions of Bootstrap this was a Hero Unit). It'll make our headline stand out a lot more. We can do this by wrapping our `<h1>` and `<input>` tags in a new div with the jumbotron class:

```html
<div class="container">
  <div class="jumbotron">
    <h1>Hello, {{name || 'World'}}</h1>
    <input type="text" ng-model="name">
  </div>
</div>
```

```html
<div class="container">
  <div class="jumbotron">
    <h1>Hello, {{name || 'World'}}</h1>
    <input type="text" ng-model="name">
  </div>
</div>
```
It's starting to look a lot better but I'm not too happy about our content touching the top of the browser like that. We can make it look a lot nicer with a page header but that input field still looks out of place to me.

First, let's sort out that page header:

```html
<div class="container">
  <div class="page-header">
    <h2>Chapter 1 <small>Hello, World</small></h2>
  </div>
  <div class="jumbotron">
    <h1>Hello, {{name || 'World'}}</h1>
    <input type="text" ng-model="name">
  </div>
</div>
```

I've included the chapter number and title here. The `<small>` tag within our `<h2>` tag gives us a nice differentiation between the chapter number and the title. The `page-header` class itself just gives us some additional margin and padding as well as a subtle border along the bottom.

**Downloading the example code**

You can download the example code files for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit http://www.packtpub.com/support and register to have the files e-mailed directly to you.
The last thing I think we could improve upon is that input box. Bootstrap comes with some great input styles so let's include those. First, we need to add the class of form-control to the text input. This will set the width to 100% and also bring out some nice styling such as rounded corners and a glow when we focus on the element:

```html
<input type="text" ng-model="name" class="form-control">
```

Much better, but to me it looks a little small when you compare it with the heading. Bootstrap provides two additional classes we can include that will either make the element smaller or larger: input-lg and input-sm respectively. In our case, the input-lg class is the one we want, so go ahead and add that to the input.

```html
<input type="text" ng-model="name" class="form-control input-lg">
```
That's better but we still need to sort the spacing out, as it looks a bit snug against our `<h1>` tag. It's probably also a good idea that we add a label in so it's clear what the user should be entering in the box. Bootstrap allows us to kill two birds with one stone as it includes a margin on the label:

```
<label for="name">Enter Your Name</label>
<input type="text" ng-model="name" class="form-control input-lg" id="name">
```

Self-test questions

1. How is Angular initialized on the page?
2. What is used to display a model's value on the page?
3. What does MVC stand for?
4. How do we create a controller and tell Angular we want to use it?
5. In Bootstrap 3, what's the new name for a Hero Unit?
Summary
Our app's looking great and working exactly how it should, so let's recap what we've learnt in the first chapter.

To begin with, we saw just how easy it is to get AngularJS and Bootstrap installed with the inclusion of a single JavaScript file and stylesheet. We also looked at how an Angular application is initialized and started building our first application.

The Hello, World app we've created, while being very basic, demonstrates some of Angular's core features:

- Expressions
- Scopes
- Models
- Two-way data binding

All of this was possible without writing a single line of JavaScript, as the controller we created was just to demonstrate two-way binding and wasn't a required component of our app.

With Bootstrap, we utilized a few of the many available components such as the jumbotron and the page-header classes to give our application some style and substance. We also saw the framework's new mobile first responsive design in action without cluttering up our markup with unnecessary classes or elements.

In Chapter 2, Let's Build with AngularJS and Bootstrap we're going to explore some more AngularJS and Bootstrap fundamentals and introduce the project we're going to be building over the course of this book.
Where to buy this book

You can buy Learning Web Development with Bootstrap and AngularJS from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.