sitepoint

BUILD YOUR OWN WEBSITE WITH BOOTSTRAP

LEARN BOOTSTRAP. ANYWHERE. ANYTIME.

Build Your Own Website With Bootstrap

- **Authors:** Ilya Bodrov
- **Cover Designer:** Alex Walker

Notice of Rights

Notice of Liability

Trademark Notice

Published by SitePoint Pty. Ltd.

48 Cambridge Street Collingwood
VIC Australia 3066
Web: www.sitepoint.com
Email: books@sitepoint.com

Ilya Bodrov

Ilya Bodrov is a personal IT teacher, a senior engineer working at Campaigner LLC, author and teaching assistant at SitePoint and lecturer at Moscow Aviations Institute. His primary programming languages are Ruby (with Rails) and JavaScript. He enjoys coding, teaching people and learning new things. Ilya also has some Cisco and Microsoft certificates and was working as a tutor in an educational center for a couple of years. In his free time he tweets, writes posts for his website, participates in open-source projects, goes in for sports and plays music.

About SitePoint

SitePoint specializes in publishing fun, practical, and easy-to-understand content for web professionals. Visit sitepoint.com to access our blogs, books, newsletters, articles, and community forums. You'll find a stack of information on JavaScript, PHP, Ruby, mobile development, design, and more.

Table of Contents

Preface

Bootstrap is one of the most popular open-source, front-end frameworks on the Web. Since its official release in 2011, it has undergone several changes, and it's now one of the most stable and responsive frameworks available.

Bootstrap is loved by web developers of all levels, as it gives them the capability to build a functional, attractive website design within minutes. A novice developer with just some basic knowledge of HTML and a little CSS can easily get started with Bootstrap.

Today's websites should be modern, sleek, responsive, and "mobile first". Bootstrap helps us to achieve these goals with minimum fuss, providing a solid foundation for any website, irrespective of project size.

This book was born out of an email course published by SitePoint. It's a simple introduction to building websites with Bootstrap 4, covering the basic principles involved in creating your next blog or portfolio site.

Conventions Used

You'll notice that we've used certain typographic and layout styles throughout this book to signify different types of information. Look out for the following items.

Code Samples

Code in this book is displayed using a fixed-width font, like so:

```
<h1>A Perfect Summer's Day</h1>
<p>It was a lovely day for a walk in the park.
```

```
The birds were singing and the kids were all back at school.</p>
```

Some lines of code should be entered on one line, but we've had to wrap them because of page constraints. An ↪ indicates a line break that exists for formatting purposes only, and should be ignored:

```
URL.open("http://www.sitepoint.com/responsive-web-
↪design-real-user-testing/?responsive1");
```

Code Samples and Live Demos

GitHub

This example has a code repository or gist available at GitHub.com.

Live Code

This example has a Live Codepen.io Demo you can play with.

Chapter

1

Introduction to Bootstrap 4

Hi, and welcome to this introductory Bootstrap 4 book, based on the email course of the same name. In the following chapters, we're going to be exploring one of the most popular front-end frameworks in the world, and see it in action by creating a simple yet attractive personal website.

What Is This Thing?

So, what is *Bootstrap*, you ask? Well, this is a free, open-source framework, created by Twitter in 2011. It contains lots of predefined templates for grids, buttons, images, forms, and more, which allows developers to design websites blazingly fast. This is especially important when preparing prototypes of minimal-value products that should look moderately appealing, but where the programmer doesn't wish to bother with creating a custom design or JavaScript plugins.

Bootstrap has significantly evolved over the years. It now boasts a mobile-first approach, a responsive, Flexbox-based grid, a handful of JavaScript extensions, and much more. The most recent version of the framework is 4.1, and we're going to be utilizing it throughout the course.

Where Do I Get It?

In order to get Bootstrap, simply navigate to the Download page and choose one of the options that works for you. You can grab the compiled CSS and JS, compile from source, utilize a CDN, or use one of the package managers (like NPM or RubyGems).

We're going to hook Bootstrap 4 from the CDN in this tutorial. So, create a new directory for your website, and inside that, place a file called `index.html`, provide all the necessary tags and include Bootstrap's files in the following way:

```
<!-- index.html -->

<!DOCTYPE html>
<html>
<head>
  <!-- meta, and other stuff... -->
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/
    ↪bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/
    ↪SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
    crossorigin="anonymous">
  <!-- other styles -->
</head>
<body>
  <!-- content... -->
  <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
    integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH
    ↪+8abtTE1Pi6jizo"
    crossorigin="anonymous">
  </script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/
    ↪popper.min.js"
    integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/
    ↪l8WvCWPIPm49" crossorigin="anonymous">
  </script>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/js/
    ↪bootstrap.min.js"
    integrity="sha384-ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE241rYiqJxyMiZ6OW/JmZ
    ↪Q5stwEULTy" crossorigin="anonymous">
  </script>
</body>
</html>
```

JavaScript files are required only for specific components like dropdown menus, carousels etc. Since we'll be using some of them, I propose including the scripts right away.

How Do I Use It?

Throughout this course, we're going to build a personal website with the

following pages:

- a home page with a welcome message and some info about the owner
- a portfolio listing work samples
- a blog page
- a contacts page

For now, let's concentrate on a generic template and see how to get started with Bootstrap.

First and foremost, it's advised that you provide a `viewport` meta tag to ensure your site works properly on mobile. Therefore, add this tag into the `index.html`:

```
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1,
    shrink-to-fit=no">
  <!-- styles -->
</head>
```

The next step is to wrap your content with a container, which is the most basic component enabling us to employ Bootstrap's grid system (which will be covered later):

```
<body>
  <main class="container"></main>
</body>
```

This type of a container has a fixed maximum width for different screen sizes. If you'd like it to span 100% of the free space all the time, use the `.container-fluid` class instead.

Let's add an `h1` header now:

```
<main class="container">
```

```
  <h1>Welcome to my personal website!</h1>
</main>
```

Beneath, add some lead text. Customize the text as you see fit:

```
<main class="container">
  <h1>Welcome to my personal website!</h1>
  <p class="lead">I'm web developer, designer, and just a good person.</p>
</main>
```

Adding a Footer

Before wrapping up, let's also add a basic footer:

```
<main class="container">
  <!-- content here... -->
</main>
<footer class="footer">
  <div class="container">
    <span class="text-muted">&copy; 2018 My Personal Site LLC.</span>
  </div>
</footer>
```

`.text-muted` is a utility class that's going to give the text a grayish color.

We're done for this chapter! Open the resulting page and try shrinking the window. Note that the container adjusts properly, which is really great.

A Good Start

You can check out a live CodePen demo of what we've done so far.

Conclusion and Next Steps

Congratulations! You've just created your first page powered by Bootstrap 4! We've seen how to include the framework on your site, how to work with containers, and typography, and how to customize the predefined styles. As a small exercise, I recommend researching other Typography usage examples, and adding some more text to the main page.

Of course, this is just the beginning, and in the next chapter we're going to add a responsive navigational bar, and power it up with some JavaScript.

Chapter

2

Working With Navigational Bars

In the first chapter, we hooked Bootstrap 4 into our project, and created a basic home page. In this chapter, we'll learn how to get started with navigational bars (aka navbars).

Navbars in Bootstrap may host menu items as well as logos, form elements, dropdowns, etc. On top of that, you can introduce a "toggler" button to hide and show your menu on smaller screens, which is especially important on mobile.

Adding a Global Menu

Let's return to our home page (`index.html`). Each navbar is wrapped in a `nav` tag with a `.navbar` class. Bootstrap supports dark color schemes out of the box. Let's assign one to our menu:

```
<!-- index.html -->

<nav class="navbar navbar-dark bg-dark" role="navigation">
</nav>

<main class="container">
  <!-- main content -->
</main>

<!-- footer -->
```

By default, the navbar is fluid and occupies all available space. Let's center its contents by adding a `.container` inside:

```
<nav class="navbar navbar-dark bg-dark" role="navigation">
  <div class="container">
  </div>
</nav>
```

Logo and Menu Items

A navbar typically has a website's logo, so let's add it now.

```
<nav class="navbar navbar-dark bg-dark" role="navigation">
  <div class="container">
    <a class="navbar-brand" href="/">Personal Site</a>
  </div>
</nav>
```

Menu items should be grouped into a `ul` tag:

```
<!-- index.html -->

<nav class="navbar navbar-dark bg-dark" role="navigation">
  <div class="container">
    <a class="navbar-brand" href="/">Personal Site</a>

    <ul class="navbar-nav">
      <li class="nav-item active">
        <a class="nav-link" href="/">Home</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="/portfolio">Portfolio</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="/blog">Blog</a>
      </li>
    </ul>
  </div>
</nav>
```

Note the `.active` class assigned to the "Home" link. It's used to show that we're on this page at the moment. Here's the intermediate result:

2-1. The current layout of our navigation

This doesn't look particularly great yet. Let's add some responsiveness.

Making It Responsive

Our task is to display the menu items on the same line and hide the menu on smaller screens. To achieve that, we have to choose a breakpoint for responsive collapsing. On certain screen sizes, the menu elements will be hidden and a "hamburger" button will be shown instead. This button will expand or collapse the navbar.

We'll collapse our menu on medium and smaller screen sizes. For that, assign a `.navbar-expand-md` class, where `md` is "medium":

```
<!-- index.html -->

<nav class="navbar navbar-dark bg-dark navbar-expand-md" role="navigation">
  <div class="container">
    <a class="navbar-brand" href="/">Personal Site</a>

    <ul class="navbar-nav">
      <!-- items -->
    </ul>
  </div>
```

Next, wrap the navbar in a `.collapse` block so that it is hidden on smaller screens:

```
<!-- index.html -->

<nav class="navbar navbar-dark bg-dark navbar-expand-md" role="navigation">
  <div class="container">
    <a class="navbar-brand" href="/">Personal Site</a>

    <div class="collapse navbar-collapse" id="globalNav">
      <ul class="navbar-nav">
        <!-- items -->
      </ul>
    </div>
  </div>
</nav>
```

The `globalNav` ID is required to hide and show the contents later. Try to shrink the window, and note that the menu is now hidden after a certain breakpoint. This is good progress, but how are users supposed to access the menu when it's hidden?

Adding a Toggle Button

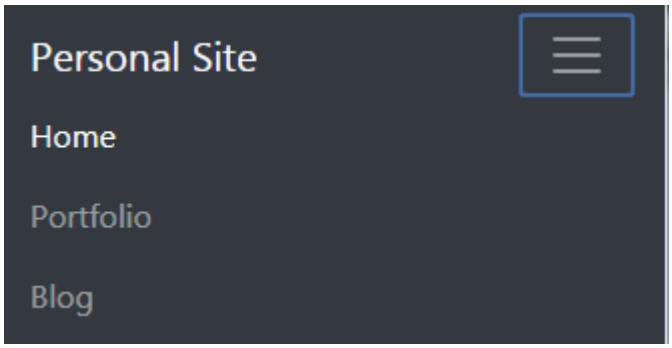
The answer is simple: we'll add a button to toggle the visibility of the collapsed menu to the footer. This button is powered by the Collapse JavaScript plugin and is only visible after a certain breakpoint (on medium and small screens, in our case).

Toggle Button

View the code for this toggle button in this GitHub gist.

The `data-toggle="collapse"` attribute is required for the plugin to work with the button. Another required attribute is `data-target="#globalNav"`, which specifies which navbar to show or hide. Its value should be the ID specified for the `.navbar-collapse`.

Try shrinking the window and pressing the toggle button. The menu should appear beneath:



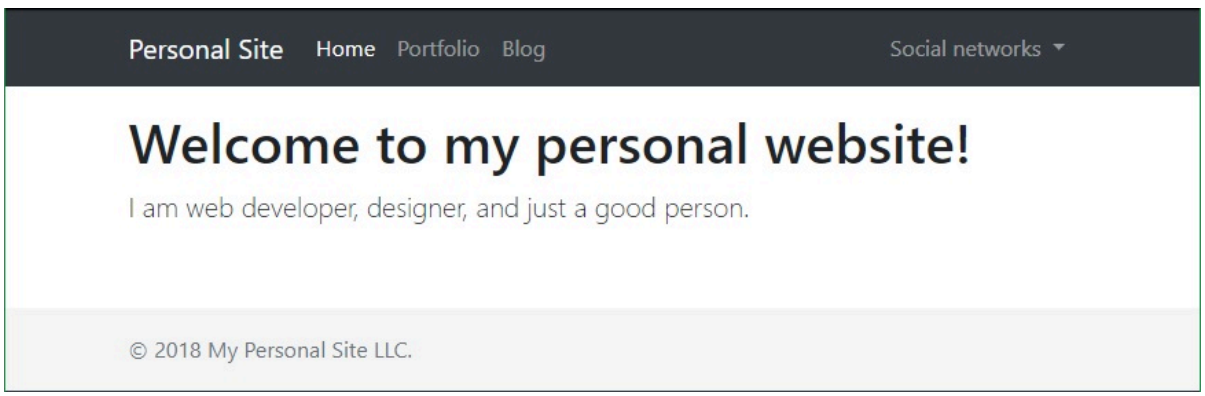
2-2. The menu appears after clicking the toggle button

The last small tweak is the addition of the `mt-3` (`margin-top`) class to the main container, so that there's some vertical space between it and the navbar.

```
<main class="container mt-3">
  <!-- content -->
</main>
```

Home Page with Menu

Check out the final result on CodePen



2-3. A screenshot of the CodePen demo

Next Steps

Our website is starting to look much better, even though we haven't written any custom CSS rules yet. Good job! In this chapter, you've learned how to work with navigational bars in Bootstrap, add elements to them, and make navbars responsive with the help of plugins.

In the next chapter, we're going to talk about a very important concept in Bootstrap—grids—which allow us to easily position elements on the page.

Chapter

Grid System 101

3

In the previous chapter, we implemented a responsive navbar with a logo, menu items and a dropdown. In this chapter, we'll learn about **grids**, which allow us to position elements on the page, while making the layout responsive.

12 Columns of Responsive Happiness

So, how does the grid work? The page is divided into twelve columns with gutters between them. You then specify how many columns each component should occupy. It's possible to specify a different number of columns to occupy for different screen sizes, thus making the layout responsive.

A set of columns should be wrapped with a row (columns must be immediate children of the row), whereas the rows must be placed into the container. Each column has to be assigned with a `.col-{BREAKPOINT}-{NUMBER}` class:

- `{BREAKPOINT}` (optional) is a grid breakpoint and can be either `sm` (small screens), `md` (medium), `lg` (large), or `xl` (extra large). If no value is given, then extra small screens are assumed. It applies to the given breakpoint as well as to all breakpoints above, so if you have `.col-md-3`, then it will also apply to large and extra large screens.
- `{NUMBER}` (optional) is the number of columns to occupy (from `1` to `12`). When not specified, the columns will be laid out automatically, trying to occupy the same amount of space. When set to `auto`, the column will have a variable width based on how much space its content requires.

Now, what's going to happen if you have two columns (say, `.col-md-7`, and `.col-md-5`) and the page is loaded on a small screen? The columns will be stacked one beneath another, and each column will occupy all twelve columns. It's also possible, however, to assign multiple classes to the same column in order to adjust its width on different screen sizes.

Grid System

The grid system is quite flexible and powerful! Check out this GitHub gist to see what the code for a typical column layout looks like.

Grid in Action

Let's tweak the home page (the `index.html` file) by pulling the main content area to the left. To the right, there will be a sidebar with recent blog posts. These two columns should stack vertically on smaller screens.

Columns Stacking Vertically

Here's a GitHub gist showing the code involved for getting the columns to stack vertically.

Inside the main content block I'd like to display some recommendations. Why don't we utilize a nested grid for that?

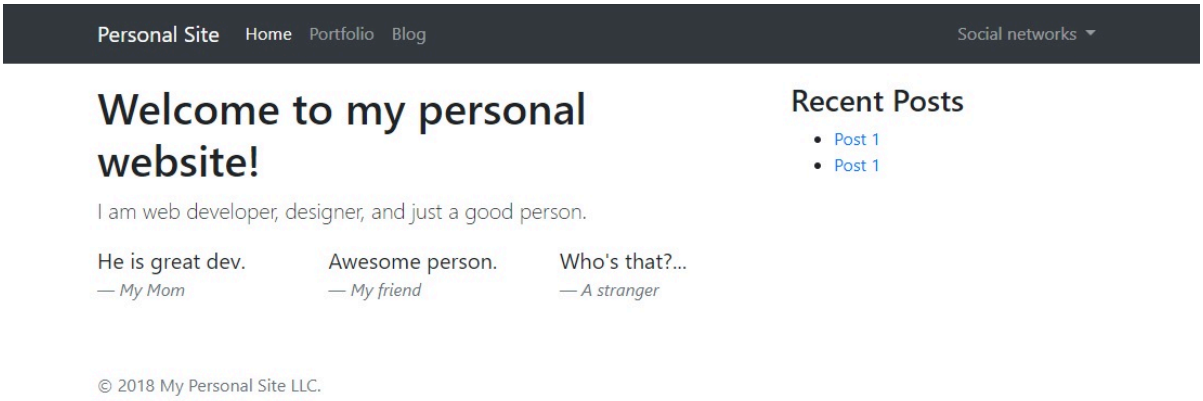
Nested Grids

The code for utilizing a nested grid can be seen in this GitHub gist.

Note that it's possible to turn blockquotes and other elements (like images) into grid columns as well. We haven't specified the number of columns to occupy, so the blockquotes will span an equal amount of space.

Progress So Far

Check out the finished product on CodePen.



3-1. A screenshot of the CodePen demo

As an exercise, you could add some more content to the main page and lay it out with a grid.

Next Steps

In this chapter, we've learned what Bootstrap's grid system is, discussed its main characteristics and features, and seen it in action. Great job!

The main page of the site now has some shape and can be further fleshed out with the content. In the next chapter, we'll proceed to the Portfolio page and discuss how to organize your sample works with Bootstrap cards.

Chapter

4

Using Cards to Organize Content

In the previous chapter, we covered the grid system in Bootstrap and saw how it could be used to lay out elements on the page. In this chapter, we'll talk about another solution for organizing your content: Bootstrap cards.

Cards are special, extensible containers that can contain plain text, images, links, buttons etc. They can be adjusted and laid out as needed, and can be used in conjunction with JavaScript plugins.

So, let's see cards in action!

Simple Cards

In this chapter, we're going to work on a new page called Portfolio. Create a new file called `portfolio.html`. Copy and paste all the necessary meta-tags, styles, and scripts from `index.html`. Also, copy and paste the top menu, main container and the footer.

Menu, Main, Footer

You can find the code required for the top menu, main container and the footer in this GitHub gist.

Portfolios usually list projects, and so in our case, individual projects will be represented by a card. So let's create a simple card:

```
<main class="container mt-3">
  <h1>My Portfolio</h1>

  <div class="card">
    <div class="card-body">
      <h5 class="card-title">Project X</h5>
      <p class="card-text">A very cool website created by me</p>
      <a href="#" class="btn btn-primary">Check it</a>
      <a href="#" class="btn btn-secondary">Source code</a>
```

```
    </div>
  </div>
</main>
```

`.card`, `.card-body`, and `.card-text` are required classes that style your card properly. One important thing to note is that the card will have a 100% width by default, but it would probably be better to make it smaller. One of the possible solutions is to use a grid.

Using a Grid

A code demo of using a grid can be seen in this GitHub gist.

On larger screens, each card will occupy four columns, and on smaller screens, six columns. On extra small screens, the cards will stack vertically (the `.mt-4` class is used to add some top margin for each block).

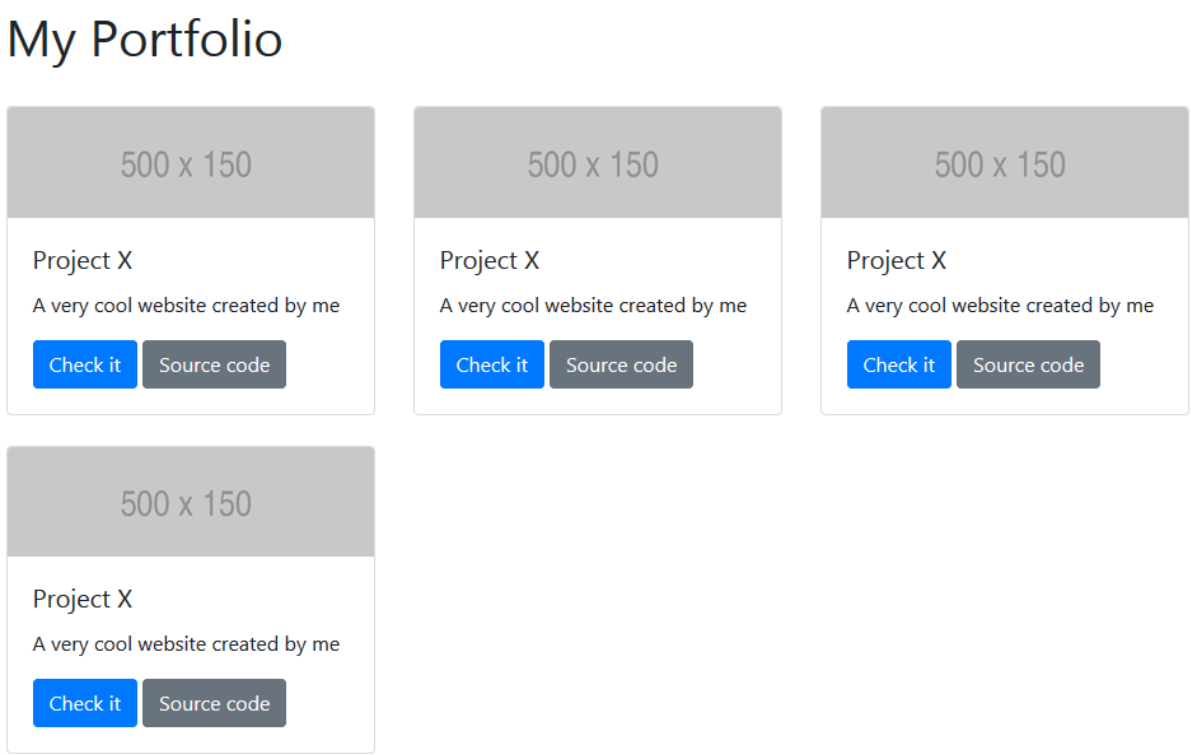
Adding Images

It might be a good idea to add a small preview image for each project. It's quite a simple task to do.

Preview Images

Here's a GitHub gist with the code you need for that.

`.card-img-top` stretches the top image to 100% of the card's width, but it's also possible to achieve a similar effect for a bottom image by applying a `.card-img-bottom` class. Here's the intermediate result:



4-1. The result of adding a `.card-img-bottom` class

A card image can also be set as an overlay if needed.

List Groups

You may also want to add something more fancy than just some text. Suppose we'd like to display a list of responsibilities you were carrying out while working on a project. List group seems like a nice candidate here.

List Group

Demo code for this is available in this GitHub gist.

Note that the `.list-group` is not actually a part of the `.card-body`, but rather a separate block. The `.card-body`, however, can be closed and reopened later, which effectively means that the card may have multiple bodies. The

`.list-group-flush` class removes all outer borders of the list group, as the card already has its own borders.

Cards and Colors

Bootstrap provides us with lots of classes to change the cards' colors as we see fit. You can tweak the color of background, text and border by using the `.bg-{TYPE}`, `.text-{TYPE}`, and `.border-{TYPE}` classes (the `TYPE` should have one of the predefined values).

For instance, you can make the cards' borders darker by assigning a `.border-dark` class.

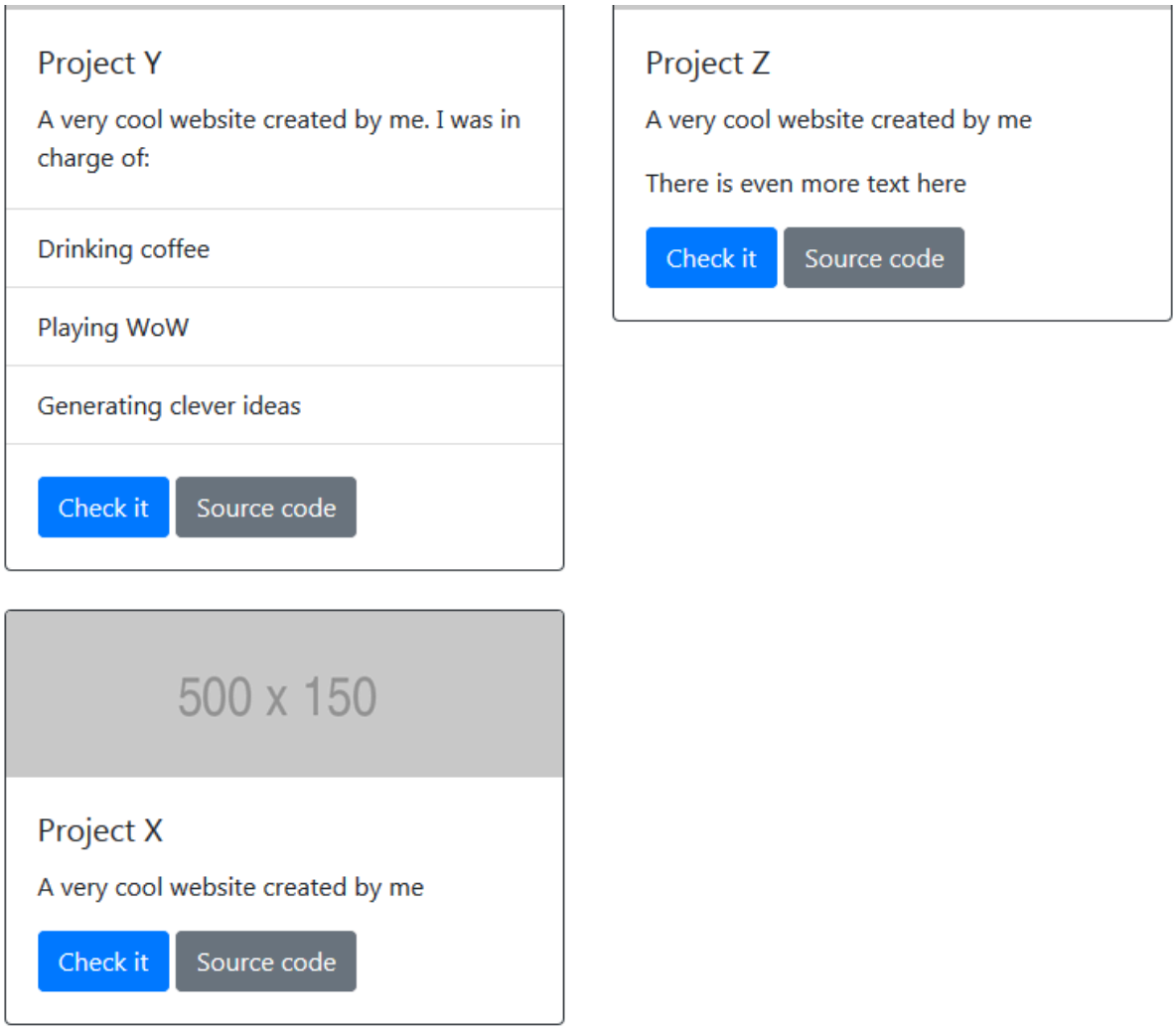
Assigning `.border-dark`

A code example of assigning a `.border-dark` class is seen in this GitHub gist.

As an exercise, try playing around with these classes, and also read up on how to change colors for the list group.

Card Layout

We've already seen how to lay out our columns with the grid, but it doesn't look particularly great, as some cards may be taller than the others:



4-2. Some cards are taller than others

Out of the box, we have three potential solutions:

- Card group. This will apply uniform sizing to all the cards. The cards themselves will be rendered as a single entity (there won't be any spacing between them).
- Card deck. The cards will have uniform sizing, but there will be spacing between them.
- Card columns. The cards will be rendered in a Masonry style. Cards will have variable heights.

While you're free to stick with any of these approaches, I propose we try the latter and employ card columns. It will make the page a bit more informal and unusual. Implementing card columns will require us to get rid of the grid system and wrap the container with a `.card-columns` block.

Wrapping the Container

Here's a GitHub gist showing how to wrap the container with a `.card-columns` block.

And that's it! The cards aren't laid out in a fancy way, and they're also responsive, which is great. Once again, we didn't have to write any custom CSS at all to achieve this result.

Portfolio Page

Check out a live demo of this on CodePen.

Conclusion

In this chapter, we've seen card columns in action, and created a Portfolio page presenting your projects. We've learned how to size the columns, add various elements to them, tweak colors, and lay the columns out in a Masonry style.

The Portfolio page is basically done, and in the next chapter we'll work on the Contacts page and talk about styling form elements with Bootstrap.

Chapter

Styling Forms

5

In the previous chapter, we talked about organizing content with cards while creating a Portfolio page. In this chapter, you're going to learn how to create nice-looking forms with Bootstrap. After all, nearly every site these days has forms that allow users to perform searches, enter credit card details, post comments, and so on.

To see these concepts in action, we'll work on a new page called "Contact Me", so create a `contact.html` file before proceeding.

Vertical Form

In order to take advantage of Bootstrap's styles, you have to assign form components with proper classes, and also wrap them with a special kind of `div`:

```
<main class="container mt-3">
  <h1>Contact Me</h1>

  <form>
    <div class="form-group">
      <label for="email">Your email</label>
      <input type="email" class="form-control" id="email"
        placeholder="test@example.com">
    </div>

    <div class="form-group">
      <label for="message">Your message</label>
      <textarea class="form-control" id="message" rows="3"></textarea>
    </div>
  </form>
</main>
```

So, the input and textarea have `.form-control` classes assigned, and we also wrap them with a `.form-group` which is going to add some bottom margin.

What about the submit button? There are lots of buttons to choose from! Usually the submit button is considered to be a "primary button" that has a blue

background. Of course, you're free to choose any other color:

```
<form>
  <!-- ... -->

  <div class="form-group">
    <input type="submit" value="Send!" class="btn btn-primary">
  </div>
</form>
```

Want to make your form controls bigger? That's no problem at all: just utilize `.form-control-lg` and `.btn-lg` classes.

```
<form>
    <div class="form-group">
      <label for="email">Your email</label>
      <input type="email" class="form-control form-control-lg" id="email"
        placeholder="test@example.com"> <!-- 1 -->
    </div>

    <div class="form-group">
      <label for="message">Your message</label>
      <textarea class="form-control form-control-lg" id="message"
        rows="3"></textarea> <!-- 2 -->
    </div>

    <div class="form-group">
      <input type="submit" value="Send!" class="btn btn-primary btn-lg"> <!-- 3 -->
    </div>
  </form>
```

Checkboxes

Let's also add a checkbox to our form asking whether the user would like to subscribe to the newsletter. For this, a set of classes should be used as well:

```
<div class="form-check form-group">
  <input class="form-check-input" type="checkbox" value="" id="newsletter">
  <label class="form-check-label" for="newsletter">
    Subscribe to newsletter
  </label>
</div>
```

If you'd like to give your checkboxes (radios, or select boxes) a unique styling, refer to the Custom Forms documentation.

The form does look quite nice, but why don't we try to lay components out in a different way? After all, we already know how to work with the grid system, so let's put this knowledge into practice!

Form Powered By Grid

It appears that the `.form-group` can be safely nested into rows and turned into a column. Each row can be represented using a generic `.row` class, or with a more specialized `.form-row` which will add smaller horizontal margins.

`.row` and `.form-row`

See this gist demo of `.row` and `.form-row`.

To make things a bit more complex, there's a new "Name" input that resides on the same row as the email. On smaller screens these two inputs stack vertically.

Also note that the `.form-check` is wrapped with `.form-group`. If you assign these two classes to the same `div`, the checkbox would have improper left padding, and therefore would not align with other form controls.

Note that it's also possible to create horizontal forms, where the labels and inputs are placed on the same line. This code sample on the Bootstrap site demonstrates how to achieve this result.

Help Text

In some cases you may need to display some help text under the form input—for instance, to explain how long the password should be, or what the requirements are for usernames.

Let's display help text beneath the "Email" field:

```
<label for="email">Your email</label>
<input type="email" class="form-control form-control-lg" id="email"
  placeholder="test@example.com" aria-describedby="email-help">
<small id="email-help" class="form-text text-muted">Your email won't be
  shared with anyone!</small>
```

Note that the `aria-describedby` attribute was added for the `input` tag: it ensures that screen readers announce the help text when the user focuses the input.

Validation

Before wrapping up, let's talk about form validation a bit, because it's crucial to check the data sent by the users. Some visitors may have malicious intents, while others may simply enter an incorrect value by mistake. Therefore, it's your job to check all the entered values. I really recommend having not only client-side but also server-side validation as well, because unfortunately, it's quite easy to bypass client-side protection.

All modern browsers support validation natively, but the styling for the incorrectly filled-in inputs varys from system to system, so it may be preferable to utilize a custom approach. To achieve this, we'll need to define validation rules (like `required` and `minlength` attributes), set a `novalidate` attribute and a `.custom-validation` class for the form. This class will be employed in the JavaScript later.

Validation

This GitHub gist shows the form with these validation attributes.

Also, add some JavaScript to prevent form submission whenever errors are found:

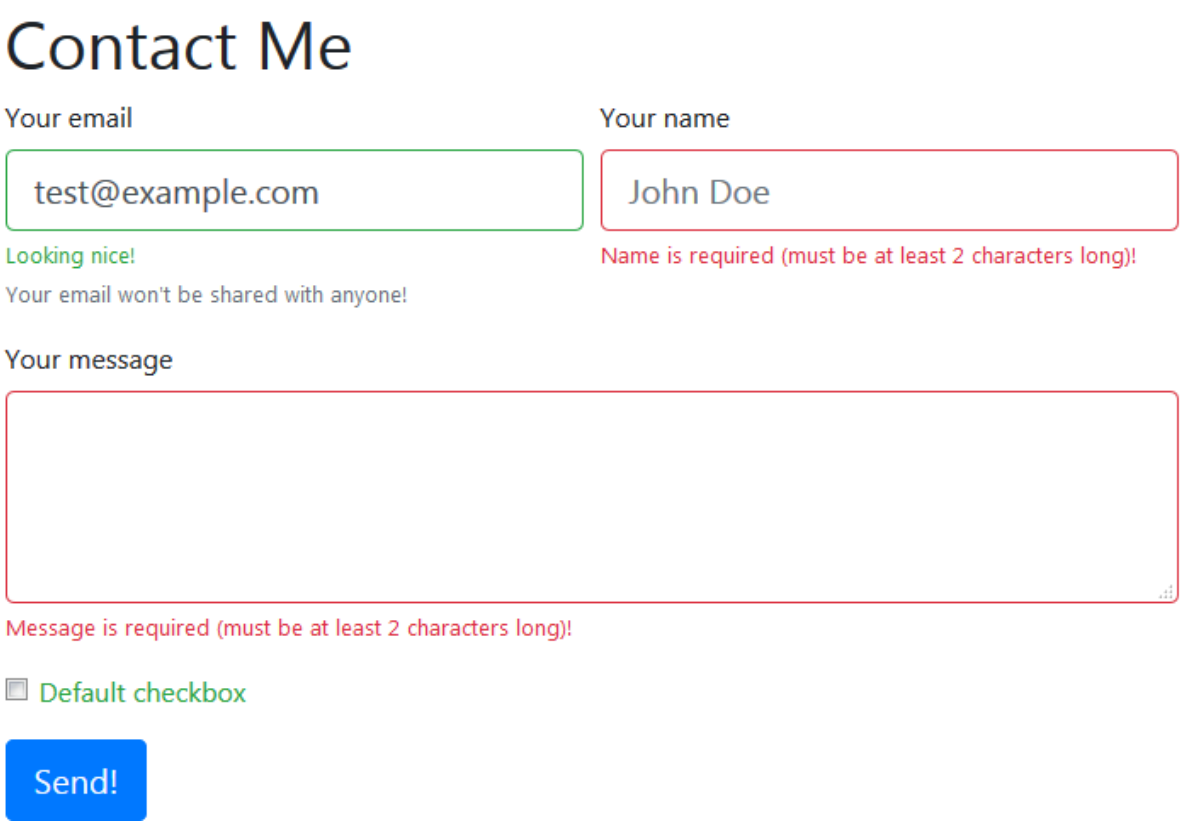
```
<!-- your form ... -->

<script>
  (function() {
  'use strict';
  window.addEventListener('load', function() {
    // Fetch all the forms we want to apply custom Bootstrap validation styles to
    var forms = document.getElementsByClassName('custom-validation');
    'custom-validation'// Loop over them and prevent submission
    var validation = Array.prototype.filter.call(forms, function(form) {
      form.addEventListener('submit', function(event) {
        if (form.checkValidity() === false) {
          event.preventDefault();
          event.stopPropagation();
        }
        form.classList.add('was-validated');
      }, false);
    });
  }, false);
})();
</script>
```

Now the properly filled-in inputs will have a green border, whereas fields with errors will be highlighted with red borders. It's also a good idea to add some feedback based on the user's input. For that, add an `.invalid-feedback` and `.valid-feedback` blocks beneath your inputs. These help blocks will be conditionally displayed after the validation is performed.

Form Feedback

A code demo of the form with valid and invalid feedback blocks.

Try to submit the form and notice that the feedback is shown!



5-1. The result of submitting invalid data

You can find other validation examples in the official Bootstrap documentation.

Contact Me Page

Here's a live CodePen demo of the final result of our work in this chapter.

Conclusion

In this chapter, you've learned how to power up your forms with Bootstrap. We've seen how to create basic vertical forms, size form controls, lay out elements with the grid, and add help text. The Contact page is finished, so your visitors can reach out to you and praise your website!

In the next chapter, we'll create a Blog page and learn about new Bootstrap components.

Chapter

Creating a Cozy Blog

6

In the previous chapter we discussed forms, and created a "Contact Me" page. In this chapter we'll create a page listing blog entries with pagination.

This will allow us to explore more Bootstrap components and see them in action, so let's dive straight into the code.

Listing the Posts

To start off, create a new `blog.html` file. Add a header, and a couple of demo posts.

Demo Posts

Here's a code example of our demo posts.

We're utilizing a card deck to organize the posts. It will ensure that the cards have equal heights, and that they stack vertically on smaller screens. Each post also has a preview image, some introductory text and a link to read the full article. We're using some utility classes like `.text-dark` and `.shadow-sm` to make the content look nicer.

Of course, your imagination is the only limit, and the page can be styled in a totally different way using other components. For instance, check this example on the Bootstrap site for some inspiration.

Displaying the Featured Posts

It's quite common to display featured posts at the top of the page, so let's do the same for our blog. One way to achieve this task is by using a "Jumbotron" component that displays various marketing messages or showcases products. However, as we'd like to show multiple posts, Jumbotron doesn't seem like the best choice.

Instead, let's take advantage of the Carousel component powered by JavaScript. In the simplest case, a carousel consists of multiple images (aka slides).

Carousel

A GitHub gist of how you might code the carousel.

You don't need to write any custom JavaScript, as Bootstrap takes care of it for us! Carousel has some default options, including the slide change interval, keyboard support, and more. You can tweak these options by adding `data-*` attributes for the carousel (for example, `data-interval`) or by writing some JavaScript as explained here.

Controls and Indicators

The carousel looks okay, but it's not very convenient: the user can't see how many slides there are, and there are no arrows to switch back and forth. In some designs this may be okay, but in our case this doesn't seem to be the best approach.

Let's add some control arrows.

Next and Previous

This GitHub gist shows how to use attributes for next and previous arrows.

Note that we have to add the `#featuredPosts` ID for the carousel, and reference it for the "next" and "previous" arrows in the `href` attribute.

Now the indicators. Each indicator must have a `data-target` with an ID of your carousel. Also, don't forget to specify the `data-slide-to` attribute that says

which slide to show when the indicator is clicked. The slides' indexes are zero-based, so the first indicator has `data-slide-to` set to `0`.

Indicators

A GitHub gist showing how the indicators are coded into our HTML.

Adding Captions

The carousel is looking better, and the last thing to do is display a brief summary for each post. For this task, we're going to be using captions.

Captions

A GitHub gist showing captions added to our code.

6-1. The current state of our carousel

The captions may not look particularly great on smaller screens (as the slides

themselves will be much smaller), so you might want to hide them altogether. To do that, employ the display utilities that hide or show the chosen elements on various screens. For example, to show a component only on medium and larger screens, assign `.d-none` and `.d-md-block` classes to it.

Pagination

Your blog may contain hundreds of posts (especially if you're an avid writer like me), and therefore displaying all of them on a single page isn't a great idea. First, it's not convenient for users, as they'll have a hard time finding the desired post. Second, the HTML page will load slower with so many elements on it. Third, the SQL query fetching all the records at once may take quite a lot of time (though this is only the case when there are thousands of records).

So, let's display a pagination block beneath your posts.

```
<nav>
    <ul class="pagination">
      <li class="page-item disabled">
        <span class="page-link">Previous</span>
      </li>
      <li class="page-item active"><a class="page-link" href="#">1
        <span class="sr-only">(current)</span></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="#">Next</a>
      </li>
    </ul>
</nav>
```

Pagination has quite a lot of required and optional classes:

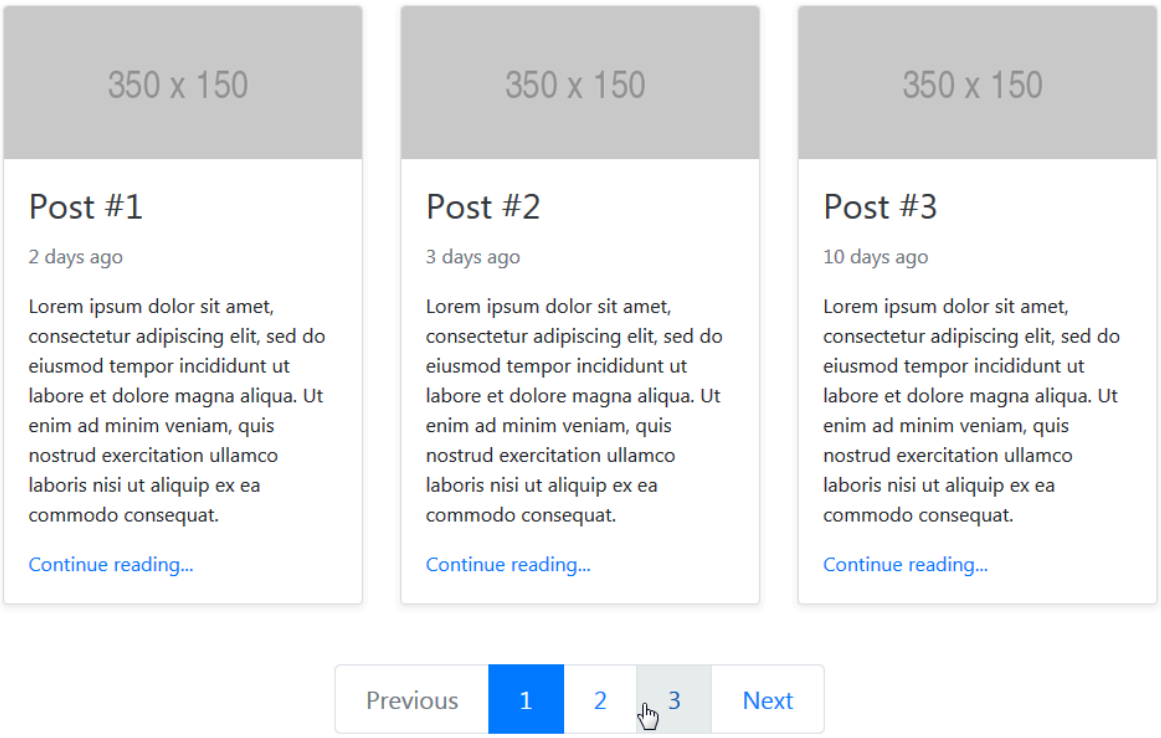
- The top-level element has to be `.pagination`.

- Each item is represented as a `.page-item` with a nested `.page-link`. The items may be set as disabled or active.
- Note that for the current page item we display an `.sr-only` block saying "current". This block won't be displayed, but will be announced by screen readers.

Let's also make our pagination larger by applying `.pagination-lg` and center it with the help of `.justify-content-center`:

```
<nav>
    <ul class="pagination justify-content-center pagination-lg">
      <li class="page-item disabled">
        <span class="page-link">Previous</span>
      </li>
      <li class="page-item active"><a class="page-link" href="#">1
        <span class="sr-only">(current)</span></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="#">Next</a>
      </li>
    </ul>
  </nav>
```

Looking quite nice, isn't?



6-2. Our pagination now looks much better

The Blog

Here's a CodePen demo of the final result.

Conclusion

Great job! We've sketched out a blog page that lists posts in a paginated way, with featured posts arranged in a carousel. As an exercise, try doing some further styling of this page. For example, try adding a sidebar with an archive to the right.

In the next chapter, we'll create an individual post page, and talk about more Bootstrap components, so hold on tight!

Chapter

7

Creating a Blog Entry Page

In the previous chapter, we created a page listing all our blog posts. In this chapter, we're going to concentrate on a single blog entry page and talk about styling the images and badges, as well as about the scroll spy feature.

A Simple Post

In the simplest case, blog posts consist of text and some images. The text is then separated into sections and paragraphs for better readability. So, for starters, let's use the `article`, `p` and header tags to format our post. Create a new `blog_post.html` file, copy all common components like top-level menu and a footer, and then tweak it as shown in the following gist.

Blog Post File

A GitHub gist of our blog post file.

Here are the main things to note:

- The top-level header is `h1` as usual.
- Then we have a `.text-muted` paragraph that shows when the article was posted and by whom.
- Next, there's a `.lead` text with the article's summary. `.pb-3` adds a small padding at the bottom.
- Lastly, there's the actual body of the post formatted with basic tags like `p` and `ul`.

That's a nice start!

Adding Search Tags

If your blog has lots of articles, you may want to tag them with labels like "html5", "javascript", "ux", etc. Let's display these labels right beneath the title of the article with the help of Bootstrap's badges:

```
<article>
  <h1>My First Post</h1>

  <section class="my-1">
    <span class="badge badge-pill badge-primary">html5</span>
    <span class="badge badge-pill badge-success">ux</span>
    <span class="badge badge-pill badge-info">beginner</span>
  </section>

  <p class="text-muted mt-0">3 days ago by <a href="#">admin</a></p>

  <!-- other text ... -->
</article>
```

Each badge is going to have its own background color thanks to the `.badge-primary`, `.badge-success`, and `.badge-info` classes. `.badge-pill` gives your badges rounded borders.

Dealing With Images

Currently our post only has some text and an unordered list. But what about images? How do we display them? Well, if your image is rather small, you can style it as a thumbnail, and float it to the left or to the right by using the appropriate class:

```
<p>
  <img src="https://via.placeholder.com/200x200"
    class="img-thumbnail float-left mr-3">
   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>
```

Note that `.img-thumbnail` is going to give the image a small border. `.mr-3` assigns a right margin to give the image some space. If you assign `.float-left` or `.float-right` to the image, don't forget to add clearfix to your `article`,

because otherwise `footer` will try to float the image as well:

```
<article class="clearfix">
  <!-- your post ... -->
</article>
```

If your image is quite wide, then perhaps it should occupy all available width. To do that, set its `display` property to `block` with the help of yet another utility class:

```
<img src="https://via.placeholder.com/600x250"
  class="rounded mx-auto d-block mb-2 img-fluid">
```

`.mx-auto` centers the image horizontally, while `.d-block` turns it to a block element. Great!

Adding a Scroll Spy Feature

The last thing we're going to implement here is a very nifty feature called **scroll spy**. Basically, it means that as the reader scrolls through the article, the currently browsed section is highlighted in the navigational block. In other words, you'll see which part of the article you're reading.

Our navigational block is going to act like an index containing the structure of the article. The index should always be visible to the reader (there's little sense in a scroll spy otherwise), so let's implement a grid.

Creating a Grid

A GitHub gist showing how to modify our code to create a grid.

Next, add the index itself. It will be styled with the help of the navigational pills component.

The Navigational Component

A GitHub gist showing how to code the navigational component.

Note that each navigational link has an `href` attribute—which is very important for ensuring the scroll spy works properly.

The next step is to wrap the body of the post with a `div` that enables scroll spy itself.

Wrapping Div

A GitHub gist showing the wrapping div in action.

Take a look at the comments in the gist above. Point #1 shows the scroll spy component. `data-spy="scroll"` is required to enable JavaScript magic, `data-target="#post-index"` points to our navigational block, and `.data-offset="0"` tells it to initially highlight the very first element of the index.

Point #2 illustrates the newly added header with an `id` set to `section-0`. This ID corresponds to the `href` of the first link inside our index.

Points #3, #4, and #5 illustrate how to add `id`s for the other headers.

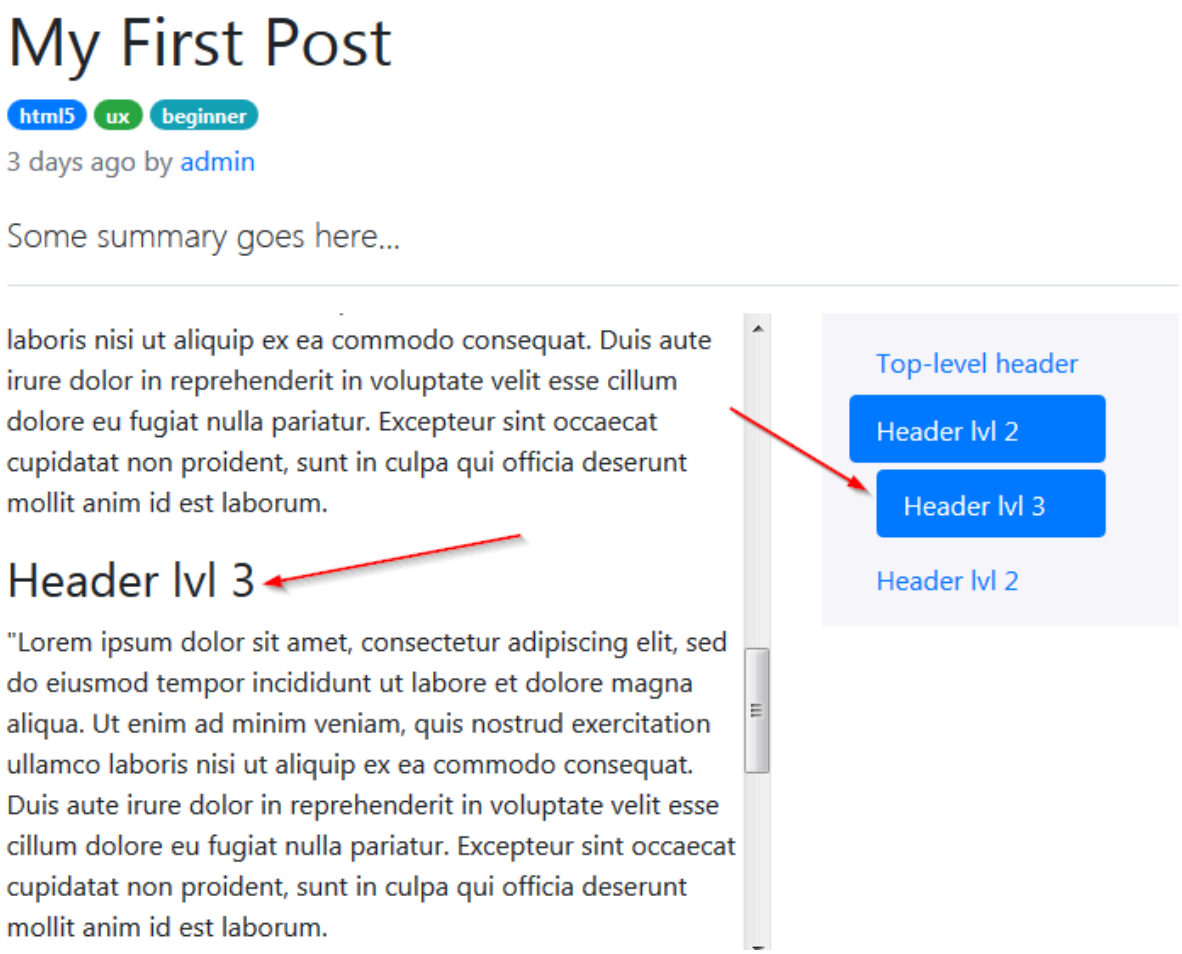
The last thing to do is to add some custom styles for the blog post text:

```
.post-body {
  height: 400px;
  overflow-y: auto;
}
```

This will set the height of the blog post to `400px` and display a scrollbar if the

text doesn't fit.

Now try to reload the page and scroll through your post. The index's items should be highlighted properly as you bring a new header into view.



7-1. Our post page now

First Post

Here's a live CodePen demo of the final result.

Conclusion

Great job! You've created a blog post page and equipped it with scroll spy functionality that looks pretty nice and allows users to understand what part of the article they're reading.

In the next chapter, we're going to be talking about modal windows in Bootstrap, so hold on tight!

Chapter

Working With Modals

8

In the previous chapter, we finalized our blog and created a single blog entry page with text, an unordered list, images, badges, and even a scroll spy feature. In this chapter, I'd like to show you another JavaScript-powered component—the modal.

The **modal** is a special box that's displayed on top of the content based on some condition. It's often used to interact with the user, or display some important information.

Modals Overview

The idea behind modals is rather simple. You have a block on the page with some content that's initially hidden. When a specific condition happens, this block is displayed above all other elements on the page.

Modals may contain nearly any type of content, but nested modals aren't supported (and I can hardly imagine why you'd want to use this feature!). Mobile browsers do support modals, but with some caveats, so be sure to check the "Browsers and devices page" in the Bootstrap docs. The modal may have a "close" button, but it can also be dismissed by pressing the `Esc` key or by clicking outside of the modal.

Creating a Modal

Let's suppose you'd like to display a limited offer for every visitor that opens the main page of the site. Our modal is going to have a title, a small close icon, some text, and also a "Learn more" button. Tweak the `index.html` file in the following way:

```
<!-- menu and main block goes here... -->

<div class="modal" tabindex="-1" role="dialog" id="modal-offer"> <!-- 1 -->
  <div class="modal-dialog" role="document">
    <div class="modal-content">
```

```
      <div class="modal-header"> <!-- 2 -->
        <h5 class="modal-title">A limited offer!</h5>
        <button type="button" class="close" data-dismiss="modal"
          aria-label="Close"> <!-- 3 -->
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="modal-body text-center"> <!-- 4 -->
        <p class="lead">This is a very limited offer! Hurry!</p>
      </div>
      <div class="modal-footer justify-content-center"> <!-- 5 -->
        <button type="button" class="btn btn-primary btn-lg">Learn more</button>
      </div>
    </div>
  </div>
</div>
```

These are the main things to note:

1. That's the wrapper for our dialog. The `#modal-offer` will be used to show the modal upon page loading.
2. This is the modal's header, which has a border at the bottom.
3. Here we're displaying a close icon that'll dismiss the modal.
4. That's the main part of the modal that hosts all the content.
5. This is the footer with a "Learn more" button that we center horizontally.

In order to display the modal as soon as the page loads, add a `script` tag at the bottom of your HTML with the following content:

```
$('#modal-offer').modal('show')
```

Of course, you may display the modal upon button click or some other event as explained here. Also note that the modal has some default options that you can

customize as needed.

Alignment, Sizing and Animation

It's quite simple to vertically align your modal by adding a `.modal-dialog-centered` class:

```
<div class="modal" tabindex="-1" role="dialog" id="modal-offer">
  <div class="modal-dialog modal-dialog-centered" role="document">
    <!-- add class here -->
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title">A limited offer!</h5>
        <button type="button" class="close" data-dismiss="modal"
          aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="modal-body text-center">
        <p class="lead">This is a very limited offer! Hurry!</p>
      </div>
      <div class="modal-footer justify-content-center">
        <button type="button" class="btn btn-primary btn-lg">Learn more</button>
      </div>
    </div>
  </div>
</div>
```

Note that this class should be added to the `.modal-dialog`, not to the `.modal` wrapper!

The modal has a default width, but you may make it smaller or larger by applying `.modal-lg` or `.modal-sm` classes respectively.

`.modal-lg` **and** `.modal-sm`

A GitHub gist showing how to apply `.modal-lg` or `.modal-sm` classes.

Modals' appearances are usually animated, and to do that simply set a `.fade` class for the wrapper.

The `.fade` Class

A GitHub gist showing the `.fade` class added to the wrapper.

More Complex Layout

As already mentioned above, modals may contain various types of content, and they also support the grid system. Let's take advantage of it to display more information about our limited offer.

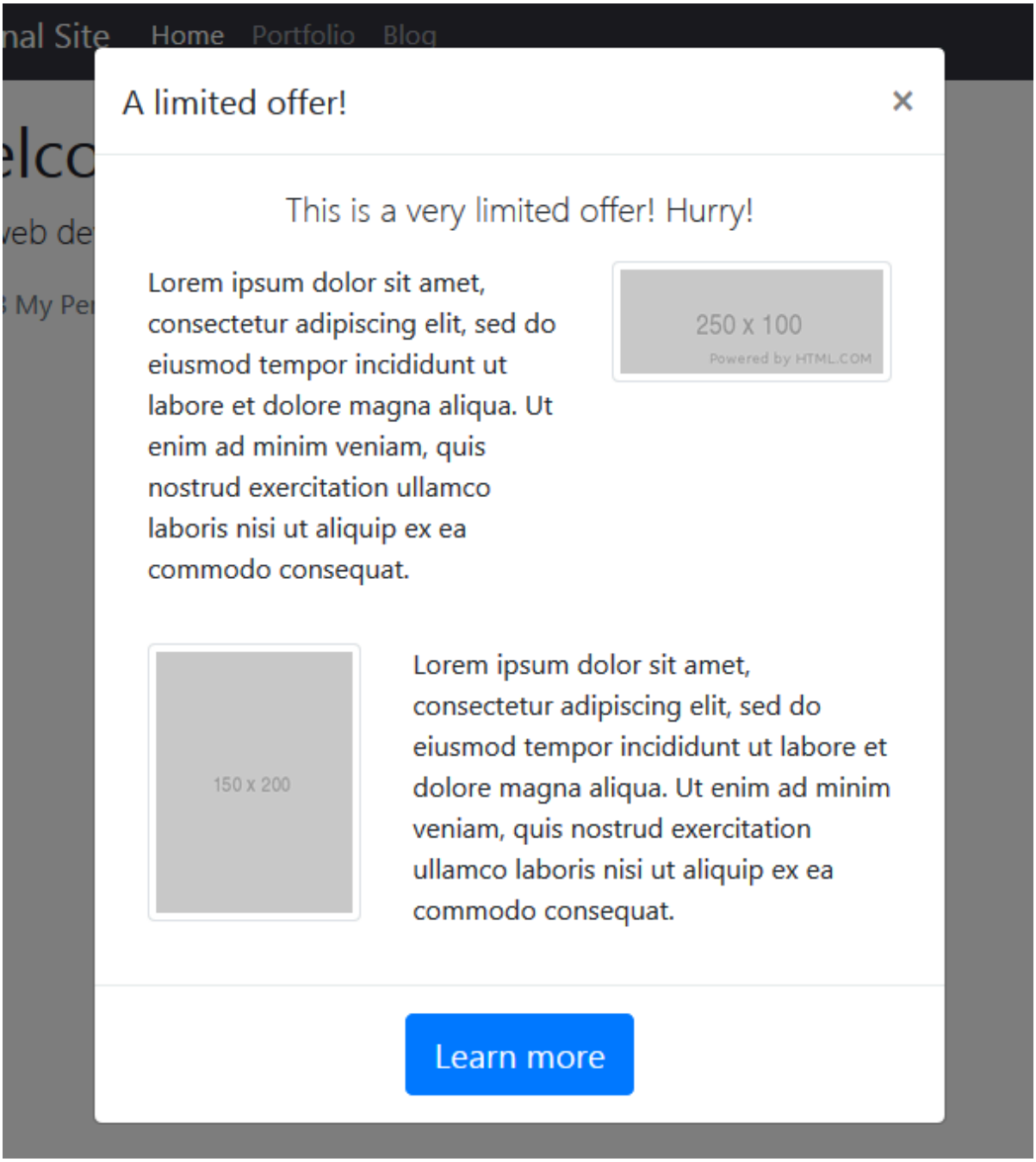
Displaying More Info

A GitHub gist showing how to take advantage of the grid to display more information.

1 We're wrapping the body of the modal with a `.container-fluid`. It will ensure that the content spans all the available space.

2 Here's the first row of the grid. It contains two columns that stack vertically on extra small screens. The image in the second column is centered horizontally, and is styled as a thumbnail.

3 That's the second row with a top margin.



8-1. How our modal appears now

As an exercise, feel free to tweak this modal further and make the content even more complex (for instance, by showing an embedded video).

Modal

Here's a live CodePen demo of our modal.

Conclusion

Well done! We've successfully implemented a modal window on the main page of the site, and now the user can learn about the exclusive offers.

We're reaching the end of this book, and the next chapter is going to be the last lesson. We'll make final tweaks to the site, and talk about some other Bootstrap components that we haven't covered.

Chapter

Final Tweaks to the Site

9

In the previous chapter, we created a “limited offer” modal that’s shown to every visitor of the site.

We’ve nearly reached the end of the course, and in this chapter we’ll make final tweaks to the site and talk about breadcrumbs, tooltips, and support for icons.

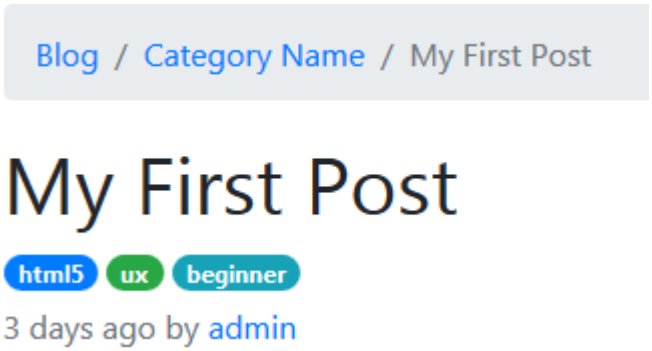
Breadcrumbs

You’ve probably heard the tale of Hansel and Gretel who followed the trail of pebbles. In web development, breadcrumbs are navigational elements that help users identify their current location on the site within a navigational hierarchy.

For example, you may add breadcrumbs to the blog entry page (which is especially useful if your posts are categorized somehow). Tweak the `blog_post.html` file as follows:

```
<main class="container mt-3">
  <nav aria-label="breadcrumb">
    <ol class="breadcrumb">
      <li class="breadcrumb-item"><a href="#">Blog</a></li>
      <li class="breadcrumb-item"><a href="#">Category Name</a></li>
      <li class="breadcrumb-item active" aria-current="page">My First Post</li>
    </ol>
  </nav>

  <article class="clearfix">
    <!-- your article... -->
  </article>
</main>
```

Pretty simple, isn’t it? Note that Bootstrap automatically adds forward slashes that delimit breadcrumbs.



9-1. Bootstrap's breadcrumbs

Tooltips

Another handy component that we haven't covered so far is the tooltip. It's powered by JavaScript and is utilized to display some additional information near the given element.

For example, if you're referring to complex terms in your blog posts, it may be a good idea to explain them in a tooltip. To enable it, simply assign a `data-toggle="tooltip"` attribute to a desired element, then instruct where the tooltip should be placed by using `data-placement` attribute, and also provide the actual tooltip's content inside the `title` attribute:

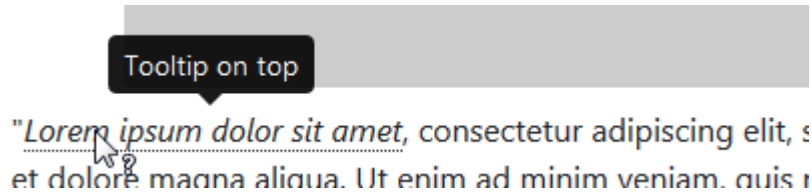
```
<em data-toggle="tooltip" data-placement="top" title="Tooltip on top"
  class="tooltip-show">Lorem ipsum dolor sit amet</em>
```

Then, add a `script` with the following content at the bottom of your HTML file:

```
$(function () {
  $('[data-toggle="tooltip"]').tooltip();
})
```

This way, we're enabling tooltip functionality for all the necessary elements. It may not be obvious, however, which element is supposed to display a tooltip, so it's a good idea to add a more distinctive styling:

```
.tooltip-show {
  cursor: help;
  border-bottom: 1px dotted #222;
}
```



9-2. A screenshot of our tooltip

Tooltip

Here's a live CodePen demo of our tooltip in action.

If you'd like to display richer content, I suggest checking out the popover component.

Icon Fonts

The icon font is a great way to easily display scalable glyphs on the page without the need to deal with bitmap images or SVGs. These glyphs are basically just text, and therefore they can be styled without any problems.

Previous versions of Bootstrap provided support for icon fonts out of the box, but this isn't the case anymore since Bootstrap 4. This decision was made because there are multiple fonts available on the Web, and ultimately it's up to the developer to choose a suitable solution.

As an exercise, let's try to add support for Font Awesome on the "Contact me" page. Add the following code to the `head` of the `contact.html` file:

```
<link rel="stylesheet" href="https://use.fontawesome.com/
```

```
    ↪releases/v5.3.1/css/all.css"
    integrity="sha384-mzrmE5qonljUremFsqc01SB46JvROS7bZs3IO2EmfFsd15u
    ↪HvIt+Y8vEf7N7fWAU" crossorigin="anonymous">
```

Now Font Awesome Free is ready to be utilized, and you can choose from hundreds of available icons.

For instance, let's show a pen icon on the "Send!" button. To do that, simply replace the submit input with the following button:

```
<button class="btn btn-primary btn-lg">
  <i class="fas fa-pen"></i>
  Send!
</button>
```

We can also make the form text fields look a bit more interesting by prepending a so-called add-on.

Pretty Fields

Here's a GitHub gist showing the code for this add-on.

1. The field with an "add-on" must be wrapped in an `.input-group` block.
2. That's the add-on itself, which may contain plain text or some tags.
3. Here, we're displaying an envelope icon.
4. The user icon is also wrapped into an `.input-group`.

So, this is how you can empower your website with icons!

9-3. Icons for email and username

Icons

Here's a live CodePen demo of our Contact Me layout with icons.

You Did It!

That's it! We've discussed breadcrumbs, tooltips, and have seen how to add support for an icon font. All in all, the first version of our website powered by Bootstrap 4 is ready. It does look quite nice, especially given the fact that we've written virtually no custom styles. Great job!

So, congratulations. You've reached the end of this introductory Bootstrap 4 book! During this course, we've covered the following topics:

- an introduction to Bootstrap
- creating responsive navigational bars
- Bootstrap's grid system and its usage
- using various types of cards to organize content
- styling and validating forms
- styling images, lists, blockquotes and other elements
- working with JavaScript-powered components like carousel, scroll spy, tooltip, and modal window
- applying various utility classes to modify elements' color, font, size, margin and so on.

Not that bad for nine chapters!

Even though we've covered quite a lot of Bootstrap's components, there's still room for improvement. The very first place I recommend checking out is the official documentation that thoroughly explains all the available components and utility classes. Second, try checking out an Examples page, where you can see Bootstrap in action and gain some inspiration to further enhance your site. Also, there's a resource listing web sites built with Bootstrap where you can see use cases and interesting applications for various components.

To learn more advanced concepts, I really encourage you to visit SitePoint's Bootstrap Hub, which contains dozens of useful articles and tutorials written by experienced authors—including me!

Hopefully this was useful and interesting for you. I really thank you for staying with me, and I'll see you in my other articles and videos!