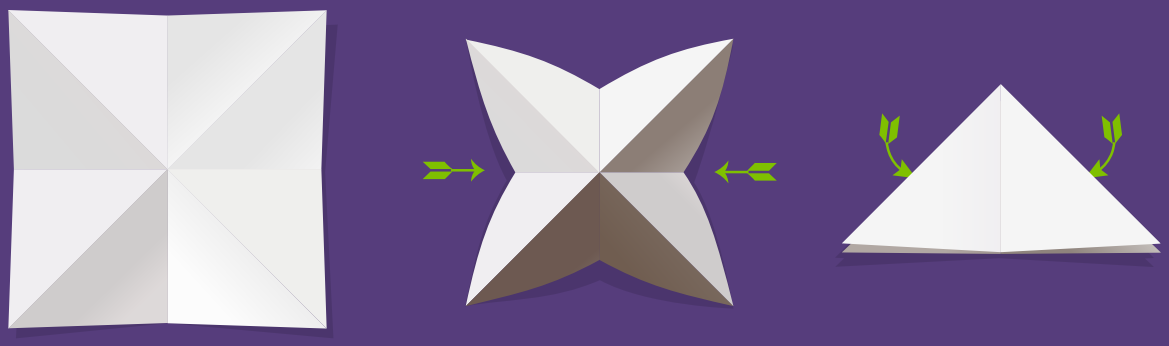
sitepoint

YOUR FIRST WEEK WITH BOOTSTRAP



BUILD RESPONSIVE, MOBILE-FIRST SITES WITH EASE

Your First Week With Bootstrap

- **Product Manager:** Simon Mackie
- **English Editor:** Ralph Mason
- **Project Editor:** Maria Antonietta Perna
- **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

Printed and bound in the United States of America

About SitePoint

SitePoint specializes in publishing fun, practical, and easy-to-understand content for web professionals. Visit http://www.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 ..... **ix**

Who Should Read This Book? ix

Conventions Used ix

Chapter 1: **Why I Love Bootstrap, and Why You Should Too** **12**

Reason 1: The Powerful Grid System 13

Reason 2: Rapid Development 13

Reason 3: Browser Compatibility 14

Reason 4: Customization! 14

Reason 5: Open Source 14

Want to Learn Bootstrap? 14

Chapter 2: **Understanding Bootstrap: How it Works, and What's New** **16**

Getting Started 17

So What Exactly Are We Going to Build? 18

The Structure 19

Diving into Bootstrap 21

Chapter 3: **Super Smart New Features to Win You Over** **31**

#1 New Interactive Documentation 32

#2 Top-notch Modular Architecture 33

#3 Easier Scaling Across Screen Sizes 38

Conclusion 40

Chapter 4: **Understanding Bootstrap Modals** **42**

The Default Modal 43

Changing the Modal's Size 46

Activating Bootstrap Modals with jQuery 46

Bootstrap Modal Events 47

Conclusion 47

Chapter 5: **A Deep Dive into the Bootstrap Form Component** **49**

Getting Started 50

Creating a Simple Form 51

Forms With Grid 60

Inline Forms 69

Form Validation 70

Conclusion 75

Chapter 6: **The Card Component: A Complete Introduction** **76**

What Is the Bootstrap Card Component? 77

Including Bootstrap in Your Project 78

Minimal Styling for Cards 79

Controlling Bootstrap Card Component Width and Height 83

Bootstrap Card Component Header and Footer 84

Adding Navigation 86

Adding Links to the Bootstrap Card Component 87

Aligning and Transforming Text in Bootstrap Card Components 88

Customizing Bootstrap Card Component Background, Foreground and Border Colors 88

Creating Advanced Layouts with the Bootstrap Card Component 89

Conclusion 93

Chapter 7: **How to Build a Responsive Type Scale with Bootstrap** **94**

How Bootstrap Sets Up Typography by Default 95

Creating the Responsive Type Scale 98

Wrapping It Up 103

The Bootstrap Responsive Type Scale in Action 103

Chapter 8: **A Beginner's Guide to the Latest Bootstrap Utility Classes** **105**

Flex 106
Float 108
Display 108
Sizing 109
Spacing 109
Text 110
Colors 112
Borders 112
Embeds 115
Close Icon 116
Conclusion 116

Chapter 9: **3 Tips for Speeding Up Your Bootstrap Website** **117**

Only Download the Bootstrap Package You need 118
Opt for the Source rather than the Precompiled Download Package 119
Make Use of Proven Client-side Optimization Techniques 119
Conclusion 121

Chapter 10: **Customizing Bootstrap jQuery**

Plugins **122**

How to Customize the Appearance of Bootstrap Plugins 123

Customizing a Bootstrap Plugin's Functionality 126

Conclusion 133

Chapter 11: **8 Tips for Improving Bootstrap Accessibility** **134**

What We're Looking For with Design Accessibility 135

Let's Start Testing Bootstrap Accessibility 138

So Is Bootstrap Accessible? 148

Chapter 12: **Front-end Frameworks: Custom vs Ready-to-use Solutions** **149**

Advantages of Ready-to-use Front-end Frameworks 150

Downsides of Ready-to-use Front-end Frameworks 151

Advantages of Custom Solutions 152

Downsides of Custom Solutions 153

The Third Solution 154

Is "Reinventing the Wheel" a Real Problem? 154

How Can You Make the Right Choice? 155

Preface

Bootstrap stands as 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. It's 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 little CSS can easily get started with Bootstrap.

In tis book we'll take you through Bootstrap basics, introduce you to its major features, and get you building your first Bootstrap sites.

Who Should Read This Book?

This book is for all frontend developers who want to build responsive, mobile-first websites. You'll need to be familiar with HTML and CSS and have a reasonable level of understanding of JavaScript in order to follow the discussion.

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>
```

Where existing code is required for context, rather than repeat all of it, ⋮ will be displayed:

```
function animate() {
  ⋮
new_variable = "Hello";
}
```

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");
```

Tips, Notes, and Warnings

Hey, You!

Tips provide helpful little pointers.

Ahem, Excuse Me ...

Notes are useful asides that are related—but not critical—to the topic at hand. Think of them as extra tidbits of information.

Make Sure You Always ...

... pay attention to these important points.

Watch Out!

Warnings highlight any gotchas that are likely to trip you up along the way.

Live Code

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

Github

This example has a code repository available at Github.com.

Chapter

1

Why I Love Bootstrap, and Why You Should Too

Syed Fazle Rahman

Bootstrap stands as 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. It's 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 little CSS can easily get started with Bootstrap.

Bootstrap provides a solid foundation for any website, irrespective of project size. It contains Reboot, which is based on Normalize.css and helps level out browser differences for various page elements. Bootstrap also provides great typography. Even basic HTML form elements like checkboxes, radio buttons, select options, etc., have been restyled to give them a modern look. I use Bootstrap because it saves me a considerable amount of effort.

Today's websites should be modern, sleek, responsive, and "mobile first". Bootstrap helps us to achieve these goals with minimum fuss. Here are the top five reasons why I love Bootstrap:

Reason 1: The Powerful Grid System

Bootstrap has one of the best responsive, mobile-first grid systems available. It's built with Flexbox and it's easy to use. It helps in scaling a single website design from the smallest mobile device to high-definition displays, logically dividing the screen into 12 columns, so that you can decide just how much screen real estate each element of your design should take up.

Although developers have CSS Grid Layout for layout building, the Bootstrap Grid component can still be handy for quick prototyping — at least while we get more familiar with the new native CSS tool at our disposal today.

Reason 2: Rapid Development

Bootstrap comes complete with many reusable CSS and JavaScript components

that can help achieve the functionality needed in almost any kind of website. You just have to use some HTML to plug them into your template, with no need to spend huge amounts of time writing complex CSS and JavaScript. Plus, all these components are responsive, too!

Reason 3: Browser Compatibility

Bootstrap is compatible with the latest, stable releases of all major browsers and platforms. With regard to the Windows platform, Bootstrap works in Internet Explorer versions 10–11 and Microsoft Edge. If Bootstrap's instructions are followed properly, you can create a website design that works in all these browsers.

Of course, if your employer wants to make a website that looks exactly the same in IE7–8 as it does in modern, standards-compliant browsers, then it might be time to think about switching jobs!

Reason 4: Customization!

Bootstrap offers many ways to customize its default design. You can override all of its CSS and default JavaScript behavior. Bootstrap is even more interesting if you're a Sass developer, as it includes Sass customization options. These options let you smoothly create a new template using Bootstrap.

Reason 5: Open Source

Bootstrap is an open-source project that's hosted on GitHub and released under the MIT license. This is one of the biggest reasons I use Bootstrap. My clients won't have to deal with purchasing and licensing issues, and Bootstrap's license gives me the freedom to completely change and experiment with it.

Want to Learn Bootstrap?

SitePoint has published my book on Bootstrap called "Jump Start Bootstrap". It

teaches you how to get started with Bootstrap. Apart from teaching you how to use Bootstrap's components and plugins, it also covers how to customize them through options like Less and Sass. A major section has been dedicated to Bootstrap's grid system, which is the most important aspect of creating a responsive website design. You'll also learn to create many demo website designs throughout the book.

Hopefully you found this article interesting, and it has convinced you to try Bootstrap in your future projects.

Chapter

2

Understanding Bootstrap: How it Works, and What's New

Syed Fazle Rahman

Version 4 of Bootstrap is a major change from all its past versions. It's a mobile-first framework and can claim to be one of the best CSS frameworks for building responsive website designs.

Since Bootstrap is a mobile-first framework, by default whatever you design or create will be mobile compatible or responsive. Isn't that cool?

Getting Started

Bootstrap has a new website design which is itself built using the latest version of the Bootstrap framework (version 4 at the time of writing). You can either include the precompiled version of Bootstrap using a CDN or download the archive file here.

Once you unzip the archive file, you'll see lots of files and folders that aren't required for our tutorial. Jump directly into the `dist` directory and copy all the folders to a new destination, which will be your project home.

In previous versions of the framework, the download included an additional fonts folder. Now, no fonts are included, but we can easily grab some nice ones from Font Awesome, for example, or from your favorite resource for fonts. In our case, we have two directories, so let's look at each of them. The `css` folder contains six CSS files:

- `bootstrap.css`
- `bootstrap.min.css`
- `bootstrap-grid.css`
- `bootstrap-grid.min.css`
- `bootstrap-reboot.css`
- `bootstrap-reboot.min.css`

The latest version of Bootstrap is a lot more modular than previous ones. If you just need a nice CSS reset, just use `bootstrap-reboot.css` (and its minified

version for production). Similarly, if you just want to use the grid, include `bootstrap-grid.css` (or `bootstrap-grid.min.css` for production) in your project.

For this tutorial, our main CSS file will be `bootstrap.css`, and we must include that in all our HTML pages. `bootstrap.min.css` is just the minified version of `bootstrap.css`. It's needed when we're actually deploying our application online.

Moving on to the `js` folder, we have the following four files:

- `bootstrap.bundle.js`
- `bootstrap.bundle.min.js`
- `bootstrap.js`
- `bootstrap.min.js`

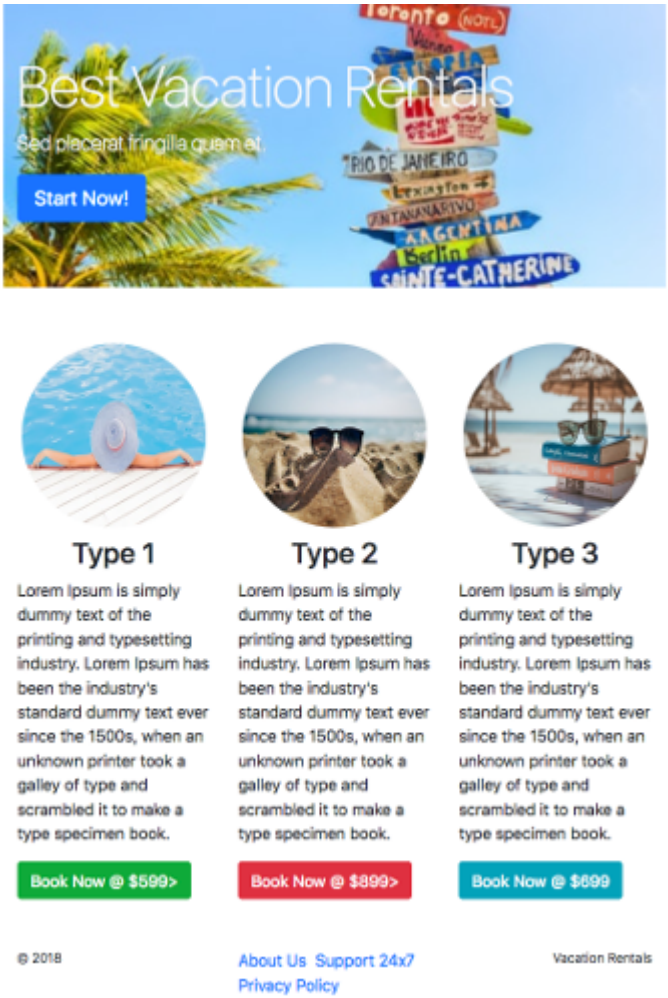
These files contain Bootstrap's main JavaScript libraries for things like carousels, drop-down menus, search auto suggest and many other powerful JavaScript functionalities. We'll use the minified version when the application is ready for deployment.

Since Bootstrap 4 beta 2, the `js` folder has included two new folders, `bootstrap-bundle.js` (along with its minified version for production), and also Popper.js, a smart JavaScript library for managing poppers in web applications.

So What Exactly Are We Going to Build?

Live Code

In this chapter, we're going to build a static landing page using Bootstrap 4, which will be called "Rental Vacations". Have a look at the demo page first.



2-1. Example project

Resize your browser window and you'll see some amazing transformations in the layout of the web page. It adjusts to the size of the window. You'll also notice that the menu bar is hiding automatically into a nice touch-compatible menu.

So we are going to build this! Excited? Yeah … me, too!

The Structure

Bootstrap understands HTML5 elements, so we need to add an appropriate `doctype` tag to our web page. Let's create a new HTML page and add the

following `doctype` tag.

```
<!DOCTYPE html>
```

Now we'll add the basic tags that are present in every HTML document:

```
<html>
  <head>
    <title>Bootstrap 101 Template</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <!-- Bootstrap -->
    <link href="css/bootstrap.min.css" rel="stylesheet" media="screen">
  </head>

<body>
    <h1>Hello, world!</h1>



</body>
</html>
```

Looking inside the `<head>`, we have the `title` element, which is easy enough to understand: it gives a title to the page.

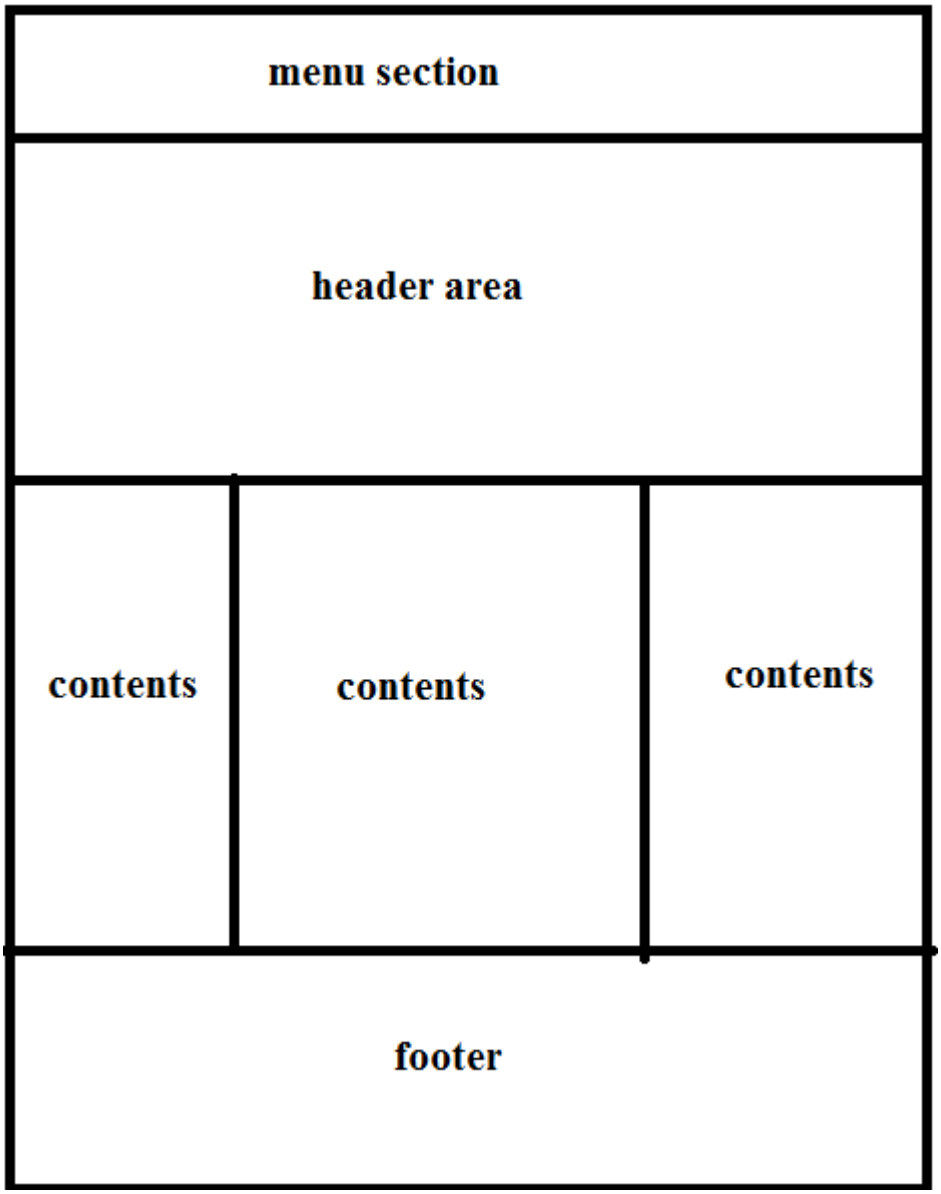
Then we have the `meta` element, which is very important to understand when using Bootstrap. Since this version of Bootstrap is built to be compatible with various types of devices (mobiles, tablets, desktops, Retina displays, etc.), you need to let the browser know that it has to scale your web page appropriately on every device.

The `viewport` meta element does this. Here, we've set the initial `content-width` to the width of the device and scaled it one time only.

After setting the `viewport` meta element, we've imported the development version of the Bootstrap CSS file, `bootstrap.css`.

Diving into Bootstrap

Now that we have our basic structure ready, we'll move on to add different components to our web page. These components are by far the most important part of every website, as we see them every day. I'm going to divide the demo page into various parts as shown below.



2-2. Bootstrap page structure

Menu Section

Designing a menu in Bootstrap is the easiest thing that can happen in the world of web designing. It's *that* easy. Let's start building a menu for our web page.

The menu is responsive by default, and gets a new look in smaller devices. Since we have the entire CSS ready in our `bootstrap.css` file, we just have to add the correct markup and correct classes to each element.

Let's start with the `<nav>` element:

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  /* navbar code here */
</nav>
```

Since Bootstrap is compatible with HTML5, we'll be using the `<nav>` element for our navigation menu. Let's understand each class applied to it.

- `navbar` is a wrapping class for navbars.
- `navbar-expand-lg` adds responsive functionality by taking charge of expanding or collapsing the navigation according to screen size. Instead of `-lg`, which stands for *large*, you can also opt for `-md` (medium screens), `-sm` (small screens), and `-xl` (extra large).
- `navbar-light` is a color class for navbars with a light background.
- `bg-light` further customizes the navbar's background color. In this case, the navbar have a light color. If we replace `bg-light` with `bg-dark`, the navbar's background color will be dark and the text color will be light.

Moving on, add the following snippet *inside the* `<nav>` *element*:

```
<a class="navbar-brand" href="#">Vacation Rentals</a>
<button class="navbar-toggler" type="button" data-toggle="collapse"
data-target="#navbarSupportedContent" aria-controls="navbarSupportedContent"
```

```
aria-expanded="false" aria-label="Toggle navigation">
  <span class="navbar-toggler-icon"></span>
</button>
```

The `navbar-brand` class is used for branding purposes. In our case, it displays the name of the website.

Next, notice the classes on the `<button>` element. This button is only visible on smaller screens, and it's used to toggle the menu's visibility on and off through `data-target="#navbarSupportedContent"`, which is a reference to the ID value of the `div` element that contains the menu. Let's add this element now, just below the previous code:

```
<!-- div containing the toggable navigation -->
<div class="collapse navbar-collapse" id="navbarSupportedContent">

<!-- navigation menu -->
   <ul class="navbar-nav mr-auto">
    <!-- active link corresponding to the current page -->
    <li class="nav-item active">
      <a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">Trip Description</a>
    <li class="nav-item">
      <a class="nav-link disabled" href="#">About Us</a>
    </li>
    <li class="nav-item">
      <a class="nav-link disabled" href="#">Book Travels</a>
    </li>
    <li class="nav-item">
      <a class="nav-link disabled" href="#">Reservations</a>
    </li>
  </ul>
  <!-- inline form inside the navbar -->
  <form class="form-inline my-2 my-lg-0">
    <input class="form-control mr-sm-2" type="search" placeholder="Search"
    aria-label="Search">
```

```
    <button class="btn btn-outline-success my-2 my-sm-0" type="submit">Search
    </button>
  </form>
</div>
```

The above code is the main content of our menu. `navbar-collapse` is applied to make the menu touch-compatible and also to change its form for smaller devices. The contents inside are pretty easy to understand. We've used basic `ul` and `li` elements for listing menu items. There's also a form that's classed `form-inline` so that it displays correctly inside the navigation bar. And finally, I've shown the basic structure to place a drop-down menu inside the navigation.

Two important things here are:

- The use of atomic utility classes like `mr-sm-2`, `mr-auto`, etc. These are pretty new to Bootstrap and there are plenty of them.
- Attention to accessibility by the use of the `sr-only` class, which handles content aimed at screen readers, and attributes like `aria-controls` and `aria-expanded`.

You can also refer to the documentation, which is highly recommended whenever you're in any kind of confusion.

So we're finally done with the navigation menu. Let's move on to building the rest of the markup inside the `<body>` section.

Header Area

Bootstrap offers a highly usable class called `jumbotron` that can be used to display large headers and contents. This is mostly used in product-based websites. For that, we need to add the following markup:

```
<div class="row">
```

```
  <header class="col jumbotron jumbotron-fluid">
    <div class="container-fluid">
      <h1 class="display-4 text-light">
        Best Vacation Rentals
      </h1>
      <p class="lead text-light">Sed placerat fringilla quam et.</p>
      <button type="button" class="btn btn-primary btn-lg">Start Now!</button>
    </div>
  </header>
</div>
```

We've put the *jumbotron* inside a `div` with the class of `row` and given the jumbotron itself an extra class of `col`. These are classes Bootstrap uses to build its 12-column grid.

Also, notice the use of the `container-fluid` and `jumbotron-fluid` classes. These classes ensure that the container and the jumbotron take 100% width of the parent container element, or of the browser if no parent element is present.

Inside the jumbotron `div` element we've placed an `h1` element, a `p` element and finally a fancy button.

The `display-4` class is one of the latest additions to Bootstrap. It's a display heading class that you can use when you need your headings to really stand out.

The `text-light` class is also a new addition to Bootstrap. It's one of the color utility classes, letting you quickly control the color and background color of elements.

Refresh your browser and see the header area. Awesome!

The Content Area

Now, we need to divide the content area into three equal areas and place them side by side. Thanks to Bootstrap's flexbox-based grid, this is going to be quick

and easy to do.

Bootstrap provides a 12-column grid system. It divides the screen into 12 equal parts, and we need to specify which HTML element occupies which parts of the grid. So in short, any element will occupy a minimum of one grid in the grid system.

Let's see the markup first.

```
<div class="row">

<!-- first column -->
  <div class="col-sm pb-4">

</div>

<!-- second column -->
  <div class="col-sm pb-4">

</div>

<!-- third column -->
  <div class="col-sm pb-4">

</div>

</div>
```

Firstly, there are three `div` elements that are contained inside a single row. So we start a new row by adding a new `div` with class `row`.

The row now contains three more `div` tags with the class `col-sm`. Before the release of Bootstrap 4, achieving this result would have required adding the `col-md-4` class to each of these `div` elements. The number 4 in the class meant that the grid spanned four columns. So three sibling `div`s spanning four columns each would occupy 12 columns, which was perfectly fine.

The new Bootstrap grid uses the strength of Flexbox to make things way less complex. The `col-sm` class on each column `div` by itself ensures that the content is evenly divided among the three `div`s until the screen size gets to 576px, at which point columns get stacked on top of each other.

We can choose to trigger this layout change at a screen size of less than 576px by using `col-xs`, or at a larger screen size of 768px with `col-md`. It's up to us, but just one grid class will do the job for us.

The `pb-4` class is one of the new spacing utility classes that come with Bootstrap 4. It creates some padding bottom inside the element on the basis of a consistent scale of values.

Since we have each sibling `div` placed appropriately, it's now time to populate them with some content. In our case, the content is unique for each one of them, with different images, but the structure is the same. Here's the content inside the first column as an example:

```
<!-- first column -->
<div class="col-sm pb-4">

<!-- column content here -->
  <a href="#">
    <img class="img-fluid mx-auto d-block mt-4 mb-2 rounded-circle"
    src="path/to/img" alt="">
  </a>
  <h3 class="text-center mb-2">Type 1</h3>
  <p>
    Lorem Ipsum...
  </p>
  <button class="btn btn-success">Book Now @ $599></button>

</div>
```

The image added has class `img-fluid` that makes it fit to the size of the parent `div` irrespective of its own size. The image is responsive to the size of each

sibling `div`. Then we have a normal `h3` element and `p` tags. You can populate the `p` tag in any manner you wish. In my demo page, I've used the auto-generated Lorem Ipsum text. Then finally, I've added a button.

`btn-success` is used to make it green. There are other classes as well, like `btn-info`, `btn-default`, `btn-warning` and `btn-danger`. You can browse the buttons docs page for all the details.

Finally, we've used a few more utility classes for margins, centering, and rounding the image's appearance into a circle.

Footer Area

The footer area follows the same principles as the content area. Here's the markup:

```
<footer class="row pt-4">

<!-- first column -->
  <div class="col-sm">
    <small><p>© 2018</p></small>
  </div>

<!-- second column -->
  <div class="col-sm">
    <ul class="list-inline">
      <li class="list-inline-item">
        <a href="#">About Us</a>
      </li>
      <li class="list-inline-item">
        <a href="#">Support 24x7</a>
      </li>
      <li class="list-inline-item">
        <a href="#">Privacy Policy</a>
      </li>
    </ul>
  </div>
```

```
<!-- third column -->
  <div class="col-sm">
    <small><p class="text-right">Vacation Rentals</p></small>
  </div>

</footer>
```

We've used the HTML5 `footer` element and created another row at the same time. Then we've divided the whole area into three equal sections, the same way we did with the main content area above. Then we populated each sub division.

Some More Bootstrap Classes

Table classes: if you're using the `<table>` element, you can use the `.table` class to make it look bit more fancy in the Bootstrap style.

Floats: you can use `.float-left` to float content to the left, `.float-right` to float content to the right, and `float-none` to remove floats from the element.

Clearfix class: this class is used to clear the float on any element.

Wrap-up

So this was the "get-me-started" tutorial for the latest version of Bootstrap. The main motive was to make you understand how Bootstrap works. As you might have seen, I haven't written more than a few lines of CSS code. Bootstrap handles everything on its own with its predefined set of CSS and JS files.

One of the main disadvantage of Bootstrap is that it isn't backwards compatible. So, if your website is built with Bootstrap 3 and you replace all the CSS and JS files with those of Bootstrap 4, the design will break. The Bootstrap creators have made huge changes to CSS class naming and have improved the responsive nature of the resulting websites.

You can also add custom CSS to your website. Create your own CSS file with any name — such as `customstyles.css` — and import the Bootstrap CSS into it. Then define your own style by using an appropriate element class or ID. In this demo, I would use `customstyle.css` to change the background of the `jumbotron` element.

Chapter

3

Super Smart New Features to Win You Over

Maria Antonietta Perna

The first stable release of Bootstrap 4 is here. And it's pretty cool. This article looks at some of its best features.

It was on 19th August, 2015, that Bootstrap 4 alpha was finally out. This was after months of anticipation, anxious tweets asking for the disclosure of a release date, and a few scattered scraps of news by Mark Otto and Jacob Thornton, having the effect of intensifying rather than quenching our curiosity. More than two years later, the wait for the first stable release is finally over.

As a designer, I love crafting my own CSS. However, I confess that I find Bootstrap a well thought out and strongly supported front-end framework that I've immensely enjoyed using — both for building my projects and for learning more about writing better, modular CSS.

As soon as news of the latest release was out, I downloaded the source files for Bootstrap 4 and spent some time going back and forth between reading the docs and digging into the code to find out more.

Here are the latest Bootstrap features I like the most. I hope you find them awesome too!

#1 New Interactive Documentation

The Bootstrap documentation has been exemplary since the framework's early days. It's always had the crucial role of being a *living document* — that is, a tool in sync with the collaborative effort of building the framework and communicating it to others:

> *Abstracting and documenting components became part of our process for building this one tool and Bootstrap in tandem. — Mark Otto in 2012*

Mark himself is quite a fan of great documentation. His Code Guide by @mdo is evidence of his attitude that high-quality documentation is part and parcel of writing high-quality code.

The documentation for version 4 has been rewritten from scratch using Markdown, and its appearance has been revamped with a new layout, color palette, and the use of system fonts.

The Bootstrap docs:

- are a pleasure to navigate, both using the traditional sidebar navigation and the brand new **search form**
- structure information in a logical manner; content is never overwhelming or confusing
- include instructions and how-tos covering all areas of the framework, from different ways of installing Bootstrap to using each component and dealing with browser quirks.

Finally, if you'd like to run the Bootstrap docs locally on your computer, follow these instructions.

#2 Top-notch Modular Architecture

Bootstrap has often been the target of complaints about code bloat, too opinionated CSS styling, and a profuse quantity of components. The good news is that Bootstrap 4 has both simplified and further modularized its structure.

To begin with, some components have been eliminated altogether. The Glyphicons icon library is not bundled with the framework any more. Panels, wells, and thumbnails are replaced by the Cards component. Also, all CSS reset/normalize code and basic styling are now dealt with in a single brand new module called Reboot.

It's safe to say that, now more than ever before, using Bootstrap feels like

assembling and arranging Lego blocks in different ways. Here are some examples to clarify what I mean.

Easy-to-Override Variables

Bootstrap's Sass variables use the `!default` flag, which makes it easy for you to override their values. Grab a copy of the latest release of Bootstrap source files and open `_variables.scss` in a code editor. Here are just three of the first variables you'll come across:

```
$white:    #fff !default;
$gray-100: #f8f9fa !default;
$gray-200: #e9ecef !default;
$gray-300: #dee2e6 !default;
```

These are color variables, which you can override by simply copying and pasting the variables to your own Sass file, changing the default value, and removing the `!default` flag. No need to mess with the original Bootstrap source code.

Ready-made, Light-weight Versions

Besides `bootstrap.scss`, which includes the entire framework, you'll also find `bootstrap-grid.scss` and `bootstrap-reboot.scss`.

Each of these files includes only *selected portions* of Bootstrap. If you don't need the full-blown framework in your project, this is a great head-start: just compile one of the light-weight options and you're good to go.

Corresponding cut-down compiled packages are available for download from the Bootstrap 4 docs page.

Reusable Components

You can skin and modify components by mixing and matching a few classes. For

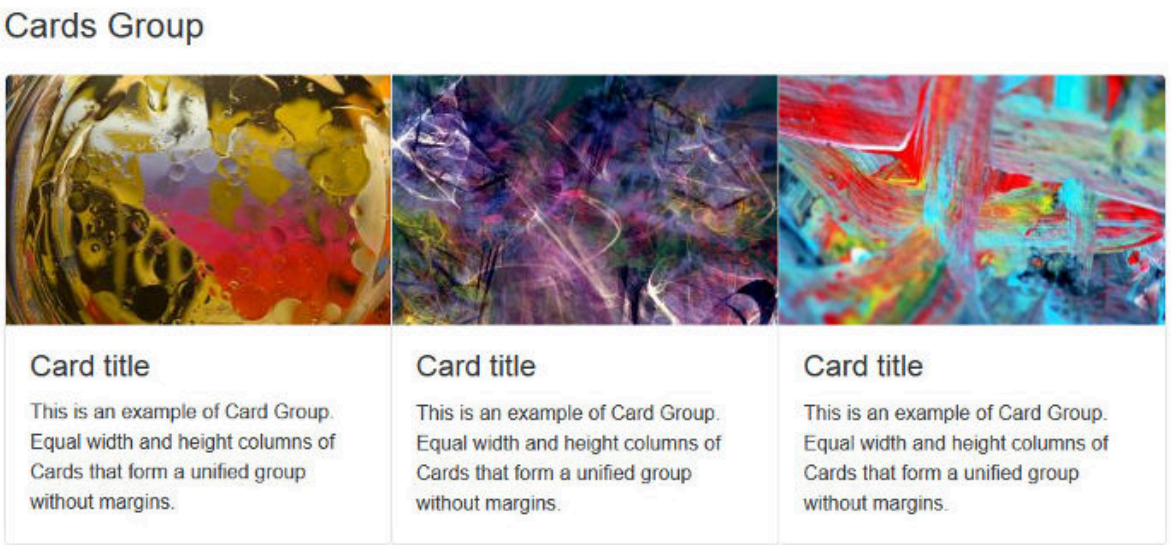
instance, the brand new cards component is a great example of this versatility in action.

Here's all the HTML you need for the simplest instance of this component:

```
<div class="card">
 <div class="card-body">
   This is some text within a card body.
 </div>
</div>
```

This flexible component easily adapts to a variety of content types and layouts. For instance, you can also arrange cards in touching equal width and height columns by wrapping them in a `.card-group` container:

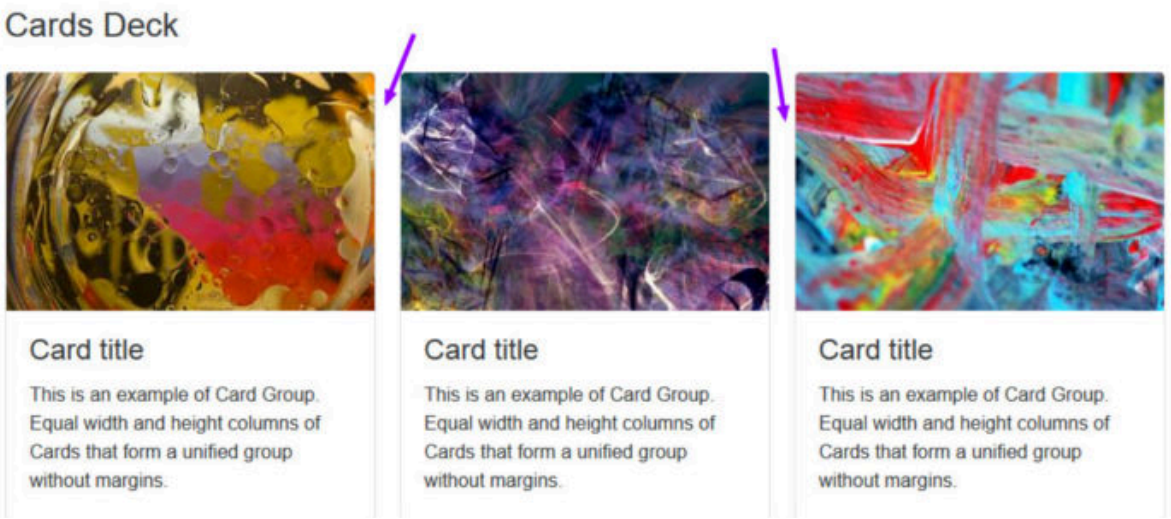
```
<div class="card-group">
 <div class="card">
   <!-- card code here -->
 </div>
 <div class="card">
   <!-- card code here -->
 </div>
 <div class="card">
   <!-- card code here -->
 </div>
</div>
```



3-1. Cards Group Component

Alternatively, you can group cards having equal width and height columns with margins, using the `.card-deck` class as follows:

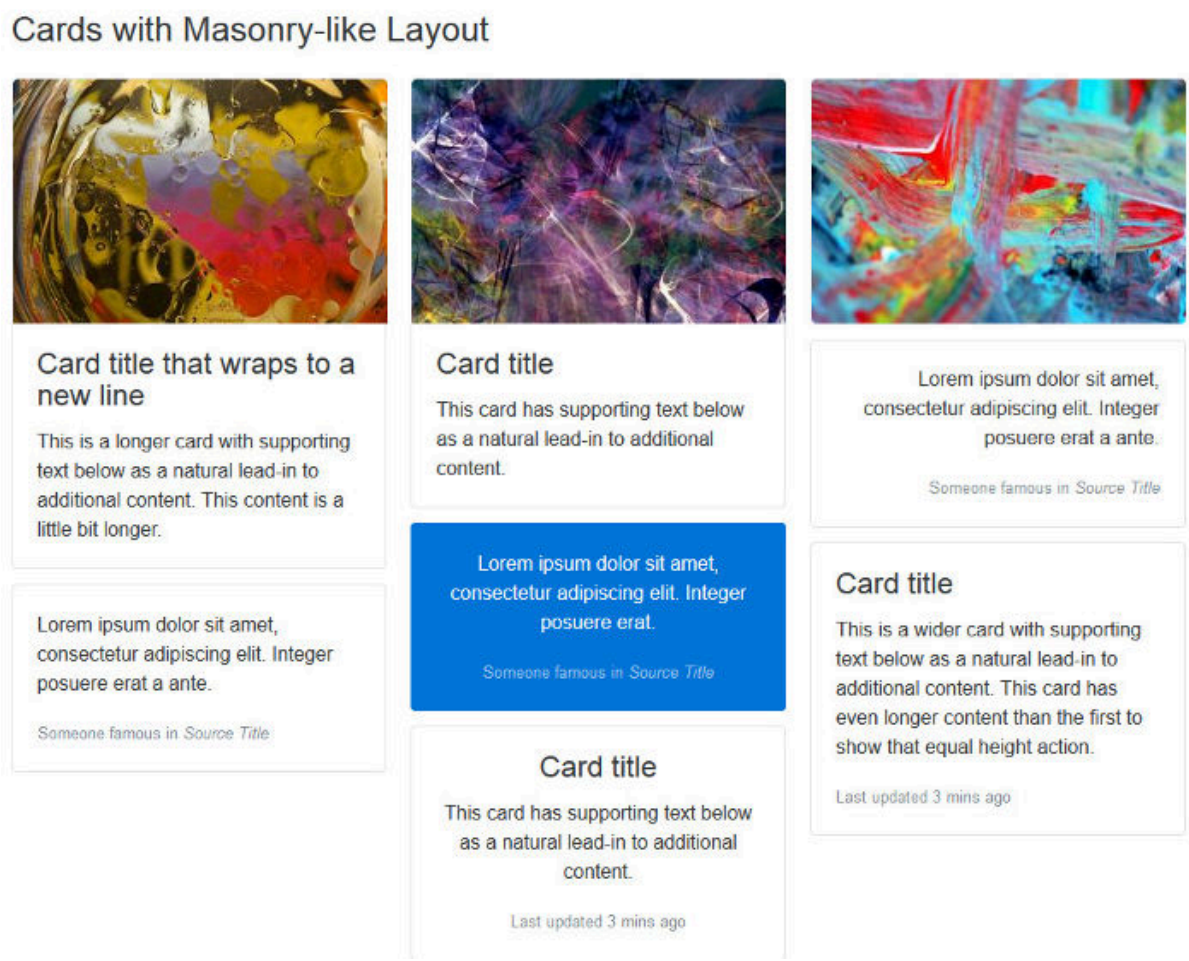
```
<div class="card-deck">
 <div class="card">
   <!-- card code here -->
 </div>
 <div class="card">
   <!-- card code here -->
 </div>
 <div class="card">
   <!-- card code here -->
 </div>
</div>
```



3-2. Cards Deck Component

Another cool thing you can do with cards is build a Masonry-like layout. Just wrap the cards in a container with the `.card-columns` class and leave the rest to Bootstrap:

```
<div class="card-columns">
 <div class="card">
   <!-- card code here -->
 </div>
</div>
```



3-3. Cards Masonry Layout

Here I've offered only a few examples of Bootstrap's modular architecture. I think these suffice to show how flexibility and extensibility are built into the framework as a whole, which makes it fun and convenient to use.

#3 Easier Scaling Across Screen Sizes

Since version 3, Bootstrap has introduced a **mobile-first** approach to web design. That is, start developing for smaller screens first and progressively add or adjust features as you target larger screens.

Version 4 makes further improvements towards adaptive web design by taking

the following steps.

The Introduction of a Flexbox-based Grid System

Bootstrap now uses Flexbox to build its grid system.

Since Flexbox is natively *flexible*, coding a responsive page layout is going to require fewer classes. For instance, to achieve a layout with three columns side by side on larger screens and stacked on top of each other on small screens, just add the `col-sm` class to each column div:

```
<div class="container">
 <div class="row">
   <div class="col-sm">
     One of three columns
   </div>
   <div class="col-sm">
     One of three columns
   </div>
   <div class="col-sm">
     One of three columns
   </div>
 </div>
</div>
```

The Move to `rem`

Where earlier versions of Bootstrap set `px` as the absolute unit of measurement, version 4 mostly uses the relative units `rem` and `em`. The goal is to have all elements on a web page harmoniously scale with the screen size.

For instance, if you dig into `_variables.scss`, you'll see that `$font-size-base` is set to `1rem`, which assumes the browser's default `font-size` (usually equivalent to `16px`). Bootstrap uses this variable's value to set the `font-size` for the document's `<body>` (see _reboot.scss).

This means that it's easier to build web pages where all elements proportionally scale up or down with the screen size without messing up your design.

Here Comes the Extra-large Breakpoint

The introduction of the new *extra large breakpoint* for the grid system further helps building layouts that scale well across different screen sizes.

This breakpoint is applied using the `.col-xl-` class and is triggered on screen sizes from `1200px` upwards.

Global Margins Reset and Utility Spacer Classes

Forcing consistent spacing between elements in a design is something most front-end developers, including myself, obsess over. It's a tricky task and the plethora of screen resolutions available doesn't make the job easier.

To help keep both vertical and horizontal spacing between elements under tight control, Bootstrap 4 resets `margin-top` to `0` while keeping a consistent `margin-bottom` value on all elements.

Further, the framework offers an impressive number of utility classes to make it easier for you to adjust margins and paddings at a more granular level across varying screen sizes.

Conclusion

I've introduced three broad features that in my view make Bootstrap really stand out:

- great documentation
- Mega Lego–type architecture
- easier scaling across devices

Did you notice I didn't mention Bootstrap's move from Less to Sass? Or the rewrite of all JavaScript plugins in ES6?

I consider these to be more like indications of Bootstrap staying current and taking advantage of the latest tools, rather than features integral to the framework itself.

Chapter

4

Understanding Bootstrap Modals

Syed Fazle Rahman

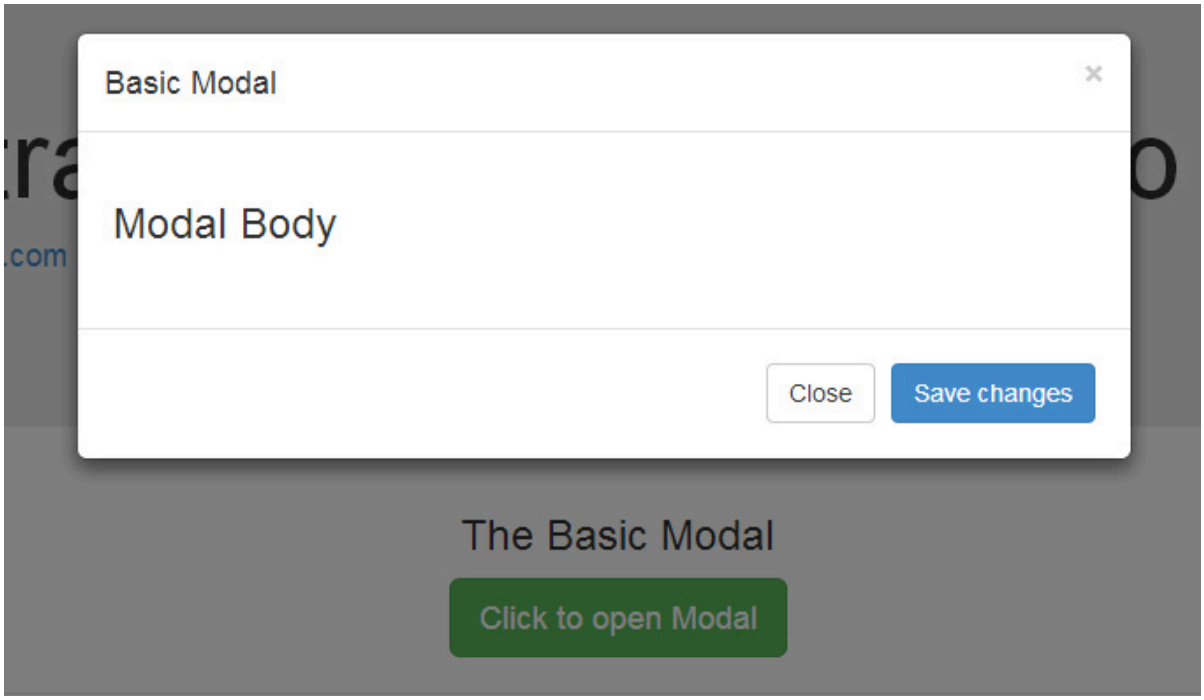
In this tutorial, we'll be talking about one of the most useful jQuery Bootstrap plugins, the Bootstrap Modal. *Bootstrap Modals* offer a lightweight, multi-purpose JavaScript popup that's customizable and responsive. They can be used to display alert popups, videos, and images in a website. Bootstrap-based websites can use Bootstrap modals to showcase, for example, terms and conditions (as part of a signup process), videos (similar to a standard light box), or even social media widgets.

Now let's examine the different parts of Bootstrap Modals, so we can understand them better.

Bootstrap Modals are divided into three primary sections: the header, body, and footer. Each has its own role and hence should be used accordingly. We'll discuss these shortly. The most exciting thing about Bootstrap modals? You don't have to write a single line of JavaScript to use them! All the code and styles are predefined by Bootstrap. All that's required is that you use the proper markup and attributes and they just work.

The Default Modal

The default Bootstrap Modal looks like this:



4-1. Bootstrap Modals: a Default Bootstrap modal

To trigger the modal, you'll need to include a link or a button. The markup for the trigger element might look like this:

```
<a href="#" class="btn btn-lg btn-success" data-toggle="modal"
data-target="#basicModal">
  Click to open Modal
</a>
```

Notice the link element has two custom data attributes: `data-toggle` and `data-target`. The toggle tells Bootstrap what to do and the target tells Bootstrap which element is going to open. So whenever a link like that is clicked, a modal with an ID of "basicModal" will appear.

Now let's see the code required to define the modal itself. Here's the markup:

```
<div class="modal fade" id="basicModal" tabindex="-1" role="dialog"
aria-labelledby="basicModal" aria-hidden="true">
```

```
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h4 class="modal-title" id="myModalLabel">Basic Modal </h4>
        <button type="button" class="close" data-dismiss="modal"
        aria-label="Close">
          <span aria-hidden="true">×</span>
        </button>
      </div>
      <div class="modal-body">
        <h3>Modal Body</h3>
      </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 parent `div` of the modal should have the same ID as used in the trigger element above. In our case, it would be `id="basicModal"`.

Attributes for Accessibility

Custom attributes like `aria-labelledby` and `aria-hidden` in the parent modal element are used for accessibility. It's good practice to make your website accessible to all, so you should include these attributes since they won't negatively affect the standard functionality of the modal.

In the modal's HTML, we can see a wrapper `div` nested inside the parent modal `div`. This `div` has a class of `modal-content` that tells `bootstrap.js` where to look for the contents of the modal. Inside this `div`, we need to place the three sections I mentioned earlier: the header, body, and footer.

The modal header, as the name implies, is used to give the modal a title and some other elements like the "x" close button. This should have a `data-dismiss` attribute that tells Bootstrap to remove the element.

Then we have the modal body, a sibling `div` of the modal header. Consider the body an open canvas to play with. You can add any kind of data inside the body, including a YouTube video embed, an image, or just about anything else.

Lastly, we have the modal footer. This area is by default right aligned. In this area you could place action buttons like "Save", "Close", "Accept", etc., that are associated with the action the modal is displaying.

Now we're done with our first modal! You can check it out on our demo page.

Changing the Modal's Size

Earlier I mentioned that the Bootstrap Modal is responsive and flexible.

The modal comes in two flavors: Large and Small. Add a modifier class `modal-lg` to the `.modal-dialog` `div` for a larger modal or `modal-sm` for a smaller modal.

Activating Bootstrap Modals with jQuery

The modal is a jQuery plugin, so if you want to control the modal using jQuery, you need to call the `.modal()` function on the modal's selector. For Example:

```
$('#basicModal').modal(options);
```

The "options" here would be a JavaScript object that can be passed to customize the behavior. For example:

```
var options = {
  'backdrop' : 'static'
```

```
}
```

Available options include:

- **backdrop**: This can be either `true` or `static`. This defines whether or not you want the user to be able to close the modal by clicking the background.
- **keyboard**: if set to `true` the modal will close via the `ESC` key.
- **show**: used for opening and closing the modal. It can be either `true` or `false`.
- **focus**: puts the focus on the modal when initialized. It can be either true or false and is set to `false` by default.

Bootstrap Modal Events

You can further customize the normal behavior of the Bootstrap Modal by using various events that are triggered while opening and closing the modal. These events have to be bound using jQuery's `.on()` method.

Various events available are:

- **show.bs.modal**: fired just before the modal is open.
- **shown.bs.modal**: fired after the modal is shown.
- **hide.bs.modal**: fired just before the modal is hidden.
- **hidden.bs.modal**: fired after the modal is closed.

You can use one of the above events like this:

```
$('#basicModal').on('shown.bs.modal', function (e) {
  alert('Modal is successfully shown!');
});
```

Conclusion

The modal is one of the best plugins offered by Bootstrap. For a novice designer,

it's one of the best ways to load content inside a popup screen without writing any JavaScript.

Live Code!

Here is a demo with three example Bootstrap modals.

Chapter

5

A Deep Dive into the Bootstrap Form Component

Ilya Bodrov-Krukowski

In this chapter, you'll learn how to style form elements with the Bootstrap form component, and how to take advantage of the grid system to align these elements. Also, we'll see horizontal and inline forms in action, as well as discuss form validation.

I still remember the (not so good) old days when we had to code all the styling for websites manually. There were very few solid CSS solutions, and creating complex UIs used to be a huge pain. It was like an arcane lore that developers were sharing with each other: they told tales on how to make rounded circles, how to highlight focused elements or create a gradient background ... I don't really miss those days. Luckily, there are many new technologies out there that help us to speed the process of styling web applications. One such technology is Bootstrap, and today we will discuss one of its most-used component: *forms*.

There are loads of predefined styles that can be applied to forms, and by using them you can create fancy UIs with little effort.

Getting Started

If you'd like to follow along, create the following boilerplate HTML:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
        rel="stylesheet">
</head>
<body>

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

    <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/
    ↪umd/popper.min.js"></script>
```

```
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.
    ↪js"></script>
</body>
</html>
```

We load the Bootstrap's styles and some scripts that will be required in one of the examples. This is it! All the markup should be placed inside the `container` block that acts as a main element for our page.

We can proceed to the next section and start crafting our first form.

Creating a Simple Form

Suppose we're creating a registration form for our new shiny website. Our form should provide, at the very least, fields for an email address and a password:

```
<div class="container">
    <p class="h1">Register</p>

    <form>
        <label for="email">Email address</label>
        <input type="email" id="email" placeholder="Enter email">

        <label for="password">Password</label>
        <input type="password" id="password" placeholder="Password">
    </form>
</div>
```

Now let's style these fields a bit. First and foremost, the `label` and `input` should be wrapped inside an element with the Bootstrap form component `form-group` class, which is going to add a small top margin. Also, we assign a `form-control` class to the inputs to make them look nicer:

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

```
        <label for="email">Email address</label>
        <input type="email" id="email" placeholder="Enter email"
        class="form-control">
    </div>

    <div class="form-group">
        <label for="password">Password</label>
        <input type="password" id="password" placeholder="Password"
        class="form-control">
    </div>
</form>
```

Note that Bootstrap automatically reboots form elements so that the new styles can be applied easily.

Why don't we also display some help text beneath the email form to inform users that we won't be using their data in any malicious way? In Bootstrap 3, help text is marked with the `help-block` class, but the latest release of the framework uses a class called `form-text`. Let's place it right beneath the `input` (inside the `form-group`). Also, let's make the text look more subtle by applying the Bootstrap form component `text-muted` class:

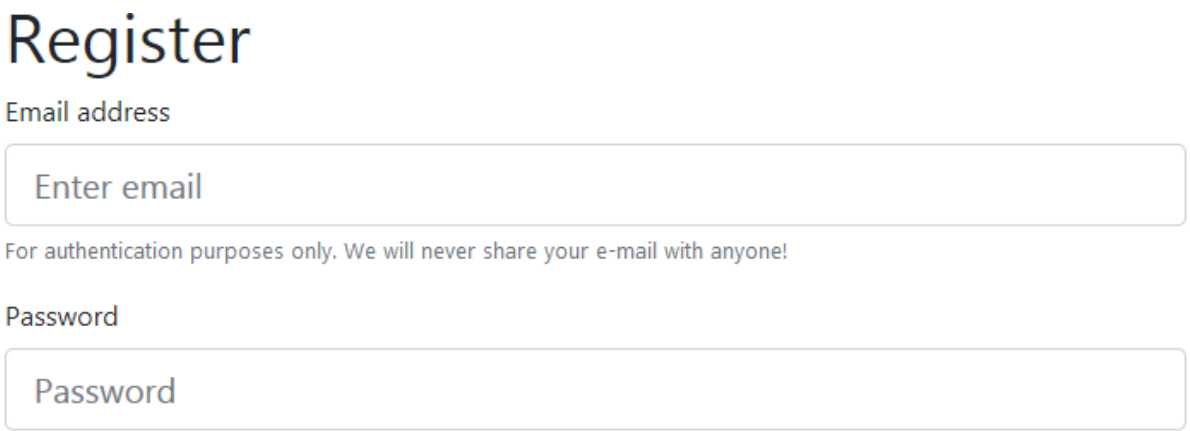
```
<div class="form-group">
  <label for="email">Email address</label>
  <input type="email" class="form-control"
     id="email" aria-describedby="emailHelp" placeholder="Enter email"> <!-- #1 -->
  <small id="emailHelp" class="form-text text-muted"> <!-- #2 -->
    For authentication purposes only. We will never share your email with anyone!
  </small>
</div>
```

Note how we added the `aria-describedby` attribute (point #1) for the input to explain that this field is being described by our help block (#2).

Why don't we also make our two fields a bit larger to emphasize their importance? Inputs can be easily sized with the Bootstrap form component's `form-control-lg` and `form-control-sm` classes:

```
<div class="form-group">
  <label for="email">Email address</label>
  <input type="email" class="form-control form-control-lg"
     id="email" aria-describedby="emailHelp" placeholder="Enter email"> <!-- #1 -->
  <small id="emailHelp" class="form-text text-muted">
    For authentication purposes only. We will never share your email with anyone!
  </small>
</div>

<div class="form-group">
  <label for="password">Password</label>
  <input type="password" class="form-control form-control-lg"
     id="password" placeholder="Password"> <!-- #2 -->
</div>
```

Here's the result:



5-1. Basic Bootstrap form component

The form looks very clear, and the currently selected input gets a nice blue border. Cool!

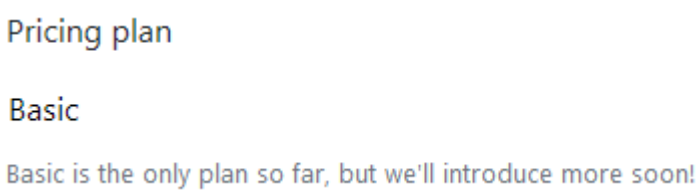
Bootstrap Form Component Read-only Elements

Suppose now we'd like to introduce multiple pricing plans for our users, but at the moment only a "Basic" plan is available. Let's add a Bootstrap form component read-only input styled as plain text. It's as simple as assigning the

`form-control-plaintext` class:

```
<div class="form-group">
  <label for="pricingPlan">Pricing plan</label>
  <input type="text" readonly class="form-control-plaintext"
     id="pricingPlan" value="Basic" aria-describedby="pricingPlanHelp"> <!-- #1 -->
  <small id="pricingPlanHelp" class="form-text text-muted">
    Basic is the only plan so far, but we'll introduce more soon!
  </small>
</div>
```

Here's the result:



5-2. form-control-plaintext class for the Bootstrap form component

Other Types of Input Supported by the Bootstrap Form Component

Bootstrap supports styling for inputs of all types. For example, we may opt for a dropdown to allow users to choose their role:

```
<div class="form-group">
  <label for="role">Your role</label>
  <select class="form-control" id="role">
    <option>Developer</option>
    <option>Designer</option>
    <option>Manager</option>
  </select>
</div>
```

Once again, we're simply wrapping everything inside a container with the Bootstrap form component `form-group` class and assigning the `form-control`

class to the dropdown.

What about the `textarea`? We just follow the same principle:

```
<div class="form-group">
  <label for="comments">Comments (optional)</label>
  <textarea class="form-control" id="comments" rows="3"></textarea>
</div>
```

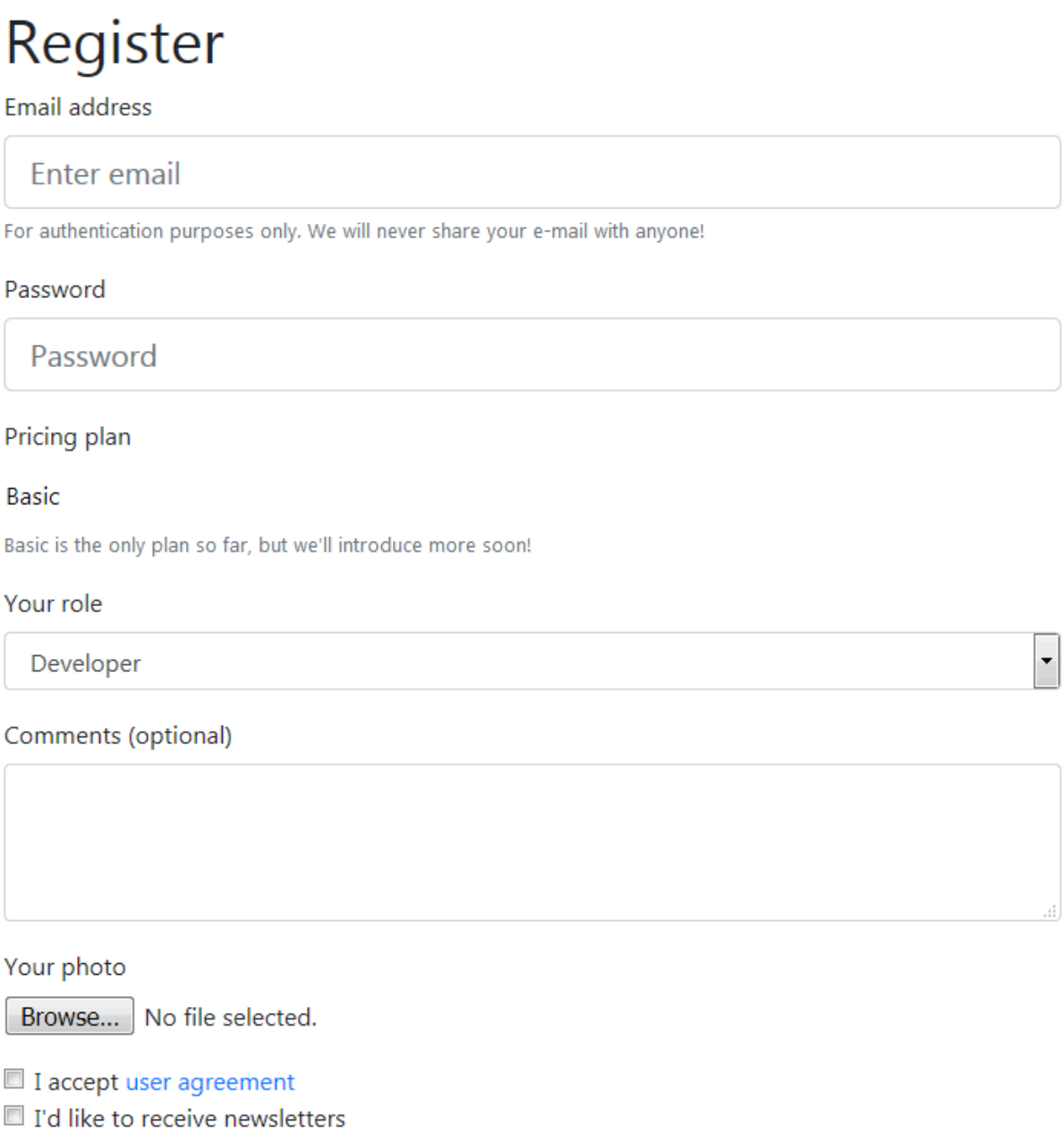
File uploading control? No problem here either:

```
<div class="form-group">
  <label for="photo">Your photo</label>
  <input type="file" class="form-control-file" id="photo">
</div>
```

Checkboxes? Easy! The only thing to remember is that checkboxes and radios should be wrapped inside an element with the `form-check`, not `form-group`, class. This class adds some extra padding to make these elements look nicer. Also, we assign `form-check-input` to the checkbox and `form-check-label` to the label so that they are aligned properly:

```
<div class="form-check">
  <input class="form-check-input" type="checkbox" value="1" id="userAgreement">
  <label class="form-check-label" for="userAgreement">
    I accept <a href="#">user agreement</a>
  </label>
</div>

<div class="form-check">
  <input class="form-check-input" type="checkbox" value="1" id="newsletter">
  <label class="form-check-label" for="newsletter">
    I'd like to receive newsletters
  </label>
</div>
```

Let's reload the page and observe the result:



5-3. How the Bootstrap form component with checkboxes is rendered in the browser

Looking quite nice for a page that has no custom styles at all.

Buttons

Next, we're going to add a *Submit* and a *Back* button to the bottom of the form.

Buttons can be styled with Bootstrap as well: there's a variety of available colors, sizes and states to choose from.

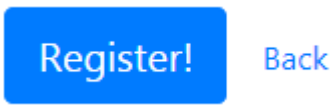
Interestingly, predefined classes don't work just with buttons. For example, a link may act as a button and can be styled as such using the same classes. At the very least, the element should be assigned a class of `btn`:

```
<!-- other form elements go here... -->

<div class="mt-3">
  <button type="submit" class="btn">Register!</button>
  <a href="#" class="btn">Back</a>
</div>
```

`mt-3` sets a small top margin for our buttons. The *Register!* button is obviously the main star, so we're going to mark it as primary and make it larger. The *Back* link doesn't have to attract all the attention, so we'll style it as a button link:

```
<div class="mt-3">
  <button type="submit" class="btn btn-primary btn-lg">Register!</button>
  <a href="#" class="btn btn-link">Back</a>
</div>
```

Now, the primary button visually stands out thanks to the blue background:



5-4. Bootstrap btn-primary and btn-link classes in action

If you don't like this background, the button can also be styled as outlined with the help of `btn-outline-*` classes:

```
<button type="submit" class="btn btn-outline-primary btn-lg">Register!</button>
```

5-5. Bootstrap btn-outline classes

Input Groups

Another cool feature of Bootstrap is input groups. Let's say we want to allow users to choose their unique profile URL looking like `http://SUBDOMAIN.PROFILE.example.com`, where:

- `SUBDOMAIN` is either `my.`, `private.`, or `own.`
- `PROFILE` is defined by the user.

So, the URL might be `http://my.superpage.example.com` or `http://private.dashboard.example.com`. Of course, we can provide a basic text input and say something like "Enter your unique profile name", but the user might not understand what's going on. Instead, we want to offer a visual example of how this profile name is going to be used.

In order to do this, we'll provide additional help to the text input in the following way:

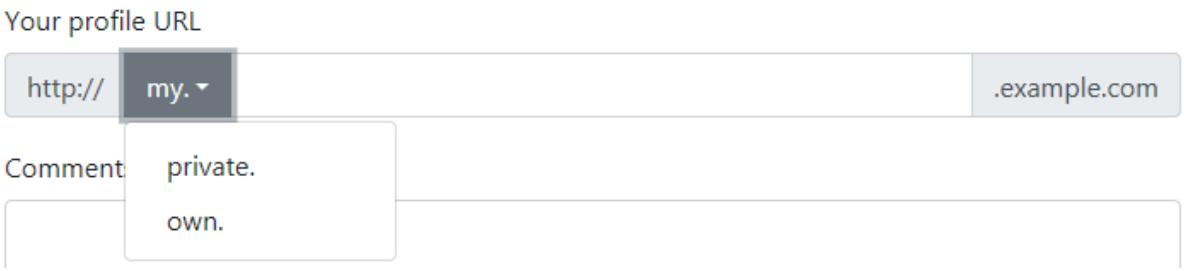
```
<div class="form-group">
  <label for="profileUrl">Your profile URL</label>

  <div class="input-group mb-3">       <!-- #1 -->
    <div class="input-group-prepend">       <!-- #2 -->
      <span class="input-group-text">http://</span> <!-- #3 -->

      <button class="btn btn-outline-secondary dropdown-toggle">  <!-- #4 -->
        type="button" data-toggle="dropdown"
        aria-haspopup="true" aria-expanded="false">my.</button>
      <div class="dropdown-menu">       <!-- #5 -->
        <a class="dropdown-item" href="#">private.</a>
```

```
        <a class="dropdown-item" href="#">own.</a>
      </div>
    </div>

    <input type="text" id="profileUrl" class="form-control">

    <div class="input-group-append">       <!-- #6 -->
      <span class="input-group-text">.example.com</span>
    </div>
  </div>
</div>
```

Let's walk through the code above step by step:

- #1. This is where our input group opens. Each input group contains an input and one or more "add-ons" to be placed before or after this input.
- #2. We're prepending two elements to our text input.
- #3. This is going to display an add-on with the `http://` text.
- #4. Here things become a bit more complex. We're displaying a button with a text "my." (which is our subdomain). When this button is clicked, a dropdown is displayed with two other options: "private." and "own.". Don't forget that the dropdown relies on JavaScript, so make sure you've hooked up all the necessary scripts as explained at the beginning of this article.
- #5. That's the actual dropdown menu, initially hidden. It may be further styled as explained by the docs.
- #6. This `div` with the class of `input-group-append` represents the last add-on that should be appended to the text input.

Whew! As a result, we've achieved the following piece of UI:



5-6. Bootstrap input groups

Looking quite nice and, what's more important, our users will understand right away what this input is for!

This wraps up the first section of this article. You may find the final result on CodePen.

Live Code

See the Pen Sitepoint Bootstrap 4 Simple Form.

Forms With Grid

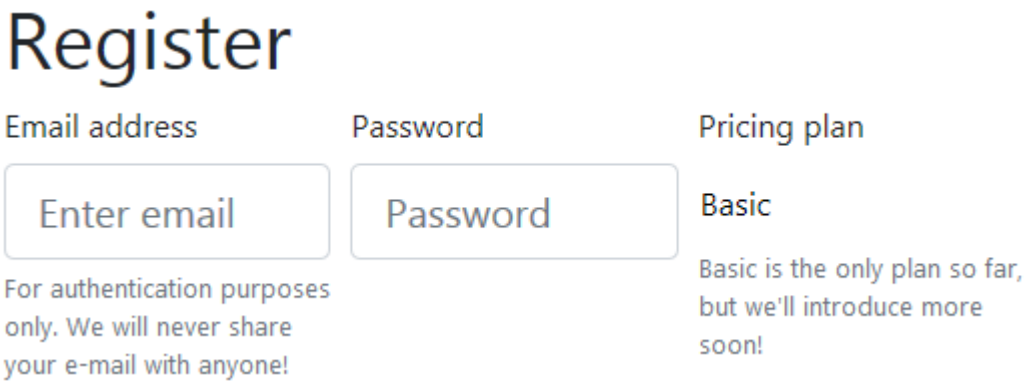
It's All About Rows and Columns

Another quite common task is to make forms more compact by placing multiple inputs on the same row. This can be done using the Bootstrap form component form grid system, which relies on Bootstrap's generic grid.

To place multiple inputs and labels on the same row, our form groups should be wrapped with either the Bootstrap form component's `row` or `form-row` (the only difference is that the latter class adds smaller gutters between the columns). Next, we assign the `col-{viewport}-{columns}` class to the *form groups themselves*. For instance, that's how we can modify the first three groups from the registration form created in the previous section:

```
<div class="form-row"> <!-- #1 -->
  <div class="form-group col-sm-4"> <!-- #2 -->
    <label for="email">Email address</label>
    <input type="email" class="form-control form-control-lg"
       id="email" aria-describedby="emailHelp" placeholder="Enter email">
    <small id="emailHelp" class="form-text text-muted">
      For authentication purposes only. We will never share your email with anyone!
    </small>
  </div>
  <div class="form-group col-sm-4">
    <label for="password">Password</label>
    <input type="password" class="form-control form-control-lg"
       id="password" placeholder="Password">
  </div>

  <div class="form-group col-sm-4">
    <label for="pricingPlan">Pricing plan</label>
    <input type="text" readonly class="form-control-plaintext"
       id="pricingPlan" value="Basic" aria-describedby="pricingPlanHelp">
    <small id="pricingPlanHelp" class="form-text text-muted">
      Basic is the only plan so far, but we'll introduce more soon!
    </small>
  </div>
</div>
```

Take a look at the result:



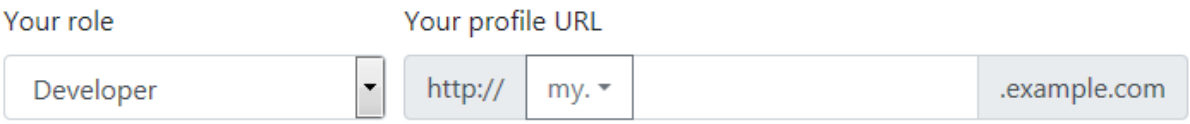
5-7. First row of Bootstrap form component styled using grid classes

For very small viewports, these groups will be stacked on top of each other.

We then proceed to the pricing plan and profile URL. The latter group probably requires quite a lot of space, so let's use an `md` (medium viewport) breakpoint in this case:

```
<div class="form-row">    <!-- #1 -->
  <div class="form-group col-md-4">    <!-- #2 -->
    <label for="role">Your role</label>
    <select class="form-control" id="role">
      <option>Developer</option>
      <option>Designer</option>
      <option>Manager</option>
    </select>
  </div>

  <div class="form-group col-md-8">    <!-- #3 -->
    <label for="profileUrl">Your profile URL</label>
    <div class="input-group mb-3">
      <div class="input-group-prepend">
        <span class="input-group-text">http://</span>
        <button class="btn btn-outline-secondary dropdown-toggle"
          type="button" data-toggle="dropdown"
          aria-haspopup="true" aria-expanded="false">my.</button>
        <div class="dropdown-menu">
          <a class="dropdown-item" href="#">private.</a>
          <a class="dropdown-item" href="#">own.</a>
        </div>
      </div>
      <input type="text" id="profileUrl" class="form-control">
      <div class="input-group-append">
        <span class="input-group-text">.example.com</span>
      </div>
    </div>
  </div>
</div>
```

Our second row:

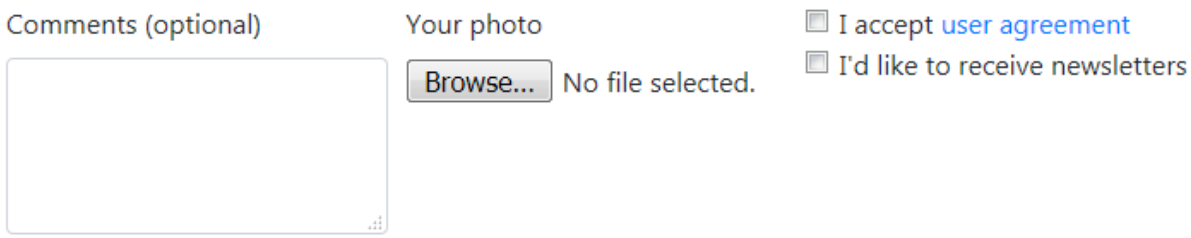


5-8. Second row of Bootstrap form component styled using grid classes

So far, so good. Next, we're going to place the Comments textarea, Photo file control and both checkboxes on the same row for medium and larger viewports:

```
<div class="form-row">  <!-- #1 -->
  <div class="form-group col-md-4">  <!-- #2 -->
    <label for="comments">Comments (optional)</label>
    <textarea class="form-control" id="comments" rows="3"></textarea>
  </div>

  <div class="form-group col-md-4">  <!-- #3 -->
    <label for="photo">Your photo</label>
    <input type="file" class="form-control-file" id="photo">
  </div>

  <div class="col-md-4">  <!-- #4 -->
    <div class="form-check">
      <input class="form-check-input" type="checkbox" value="1" id="userAgreement">
      <label class="form-check-label" for="userAgreement">
        I accept <a href="#">user agreement</a>
      </label>
    </div>

    <div class="form-check">
      <input class="form-check-input" type="checkbox" value="1" id="newsletter">
      <label class="form-check-label" for="newsletter">
        I'd like to receive newsletters
      </label>
    </div>
  </div>
</div>
```

Note that we've wrapped both checkboxes in one column (#4) for a nice layout.

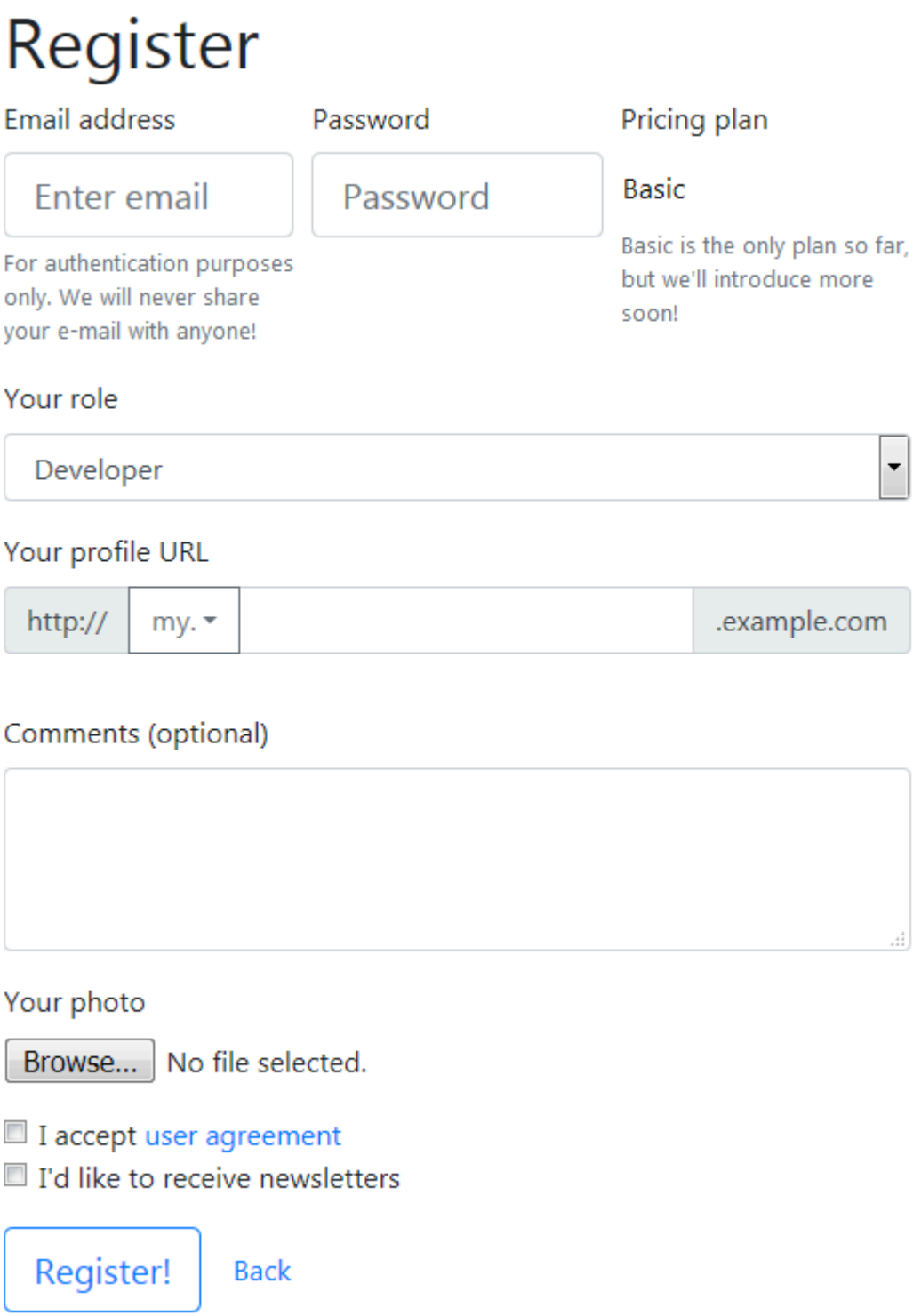


5-9. Third row of Bootstrap form component styled using grid classes

Lastly, let's style the buttons. I think they can occupy a single column:

```
<div class="form-row mt-3">
  <div class="col-xs-12"> <!-- #1 -->
    <button type="submit" class="btn btn-outline-primary btn-lg">Register!</button>
    <a href="#" class="btn btn-link">Back</a>
  </div>
</div>
```

This is how our form looks now:



5-10. Completed Bootstrap form component styled using grid classes

Live Code

See the Pen Sitepoint Bootstrap 4 Form With Rows.

Horizontal Forms

The grid system may also be leveraged to create horizontal forms — that is, forms that have the label and input elements on the same row. To achieve this result, the Bootstrap form component offers the `col-form-label` class to be added to the labels.

Let's create a small demo representing a horizontal form:

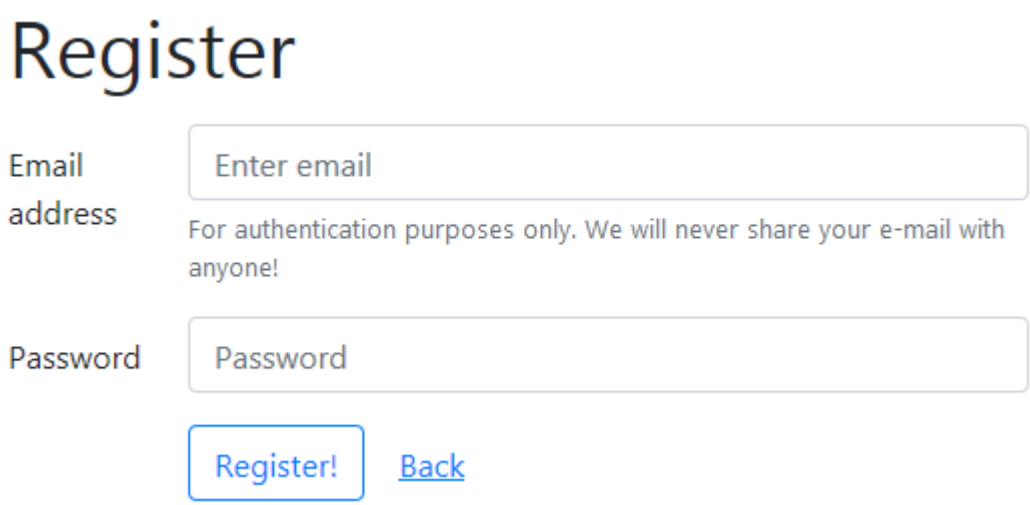
```
<div class="container">
  <p class="h1">Register</p>

  <form class="mt-3">
    <div class="form-group row">  <!-- #1 -->
      <label for="email" class="col-sm-2 col-form-label">Email address</label>
      <!-- #2 -->
      <div class="col-sm-10">  <!-- #3 -->
        <input type="email" class="form-control"
          id="email" aria-describedby="emailHelp" placeholder="Enter email">
        <small id="emailHelp" class="form-text text-muted">
          For authentication purposes only. We will never share your email with
          anyone!
        </small>
      </div>
    </div>

    <div class="form-group row">
      <label for="password" class="col-sm-2 col-form-label">Password</label>
      <div class="col-sm-10">
      <input type="password" class="form-control"
        id="password" placeholder="Password">
      </div>
    </div>
```

```
    <div class="form-group row mt-3">
      <div class="col-10 offset-2"> <!-- #4 -->
        <button type="submit" class="btn btn-outline-primary">Register!</button>
        <a href="#" class="btn btn-link">Back</a>
      </div>
    </div>
  </form>
</div>
```

The main things to note here are:

- #1. Each form group is assigned a `row` class
- #2. Labels are assigned the `col-form-label` class
- #3. Inputs (and accompanying help blocks) are wrapped into separate `div`s
- #4. For buttons we are additionally providing the `offset-2` class, so that they're moved to the right and vertically aligned with the inputs.

This is how the form looks:

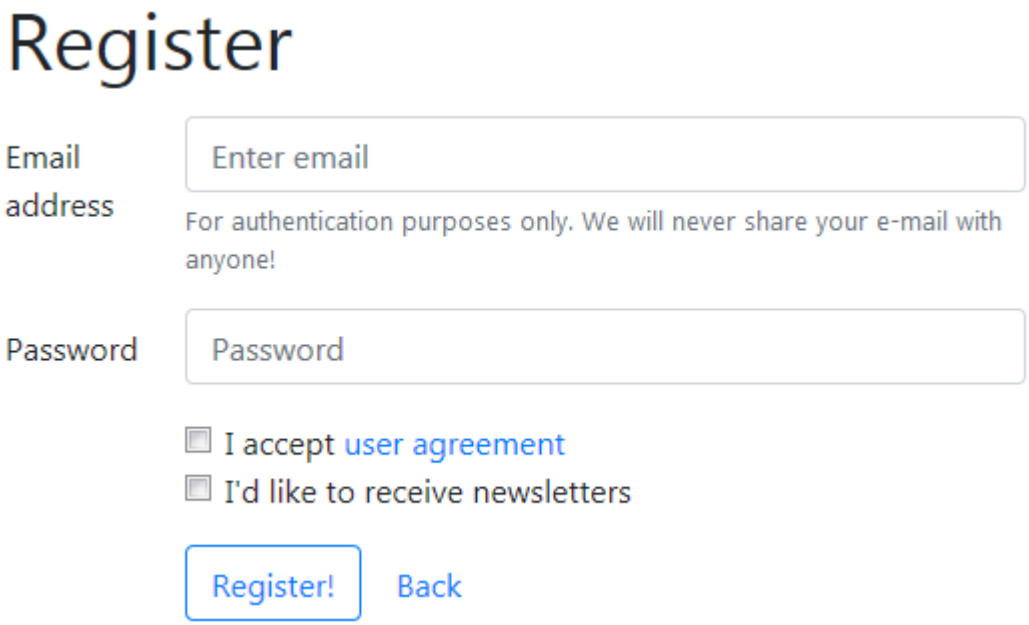


5-11. Bootstrap horizontal form built with the help of Bootstrap grid classes

What about the checkboxes? Well, we can also apply some offset and vertically align them with the buttons and inputs:

```
<div class="form-group row"> <!-- #1 -->
  <div class="col-10 offset-2"> <!-- #2 -->
    <div class="form-check">
      <input class="form-check-input" type="checkbox" value="1" id="userAgreement">
      <label class="form-check-label" for="userAgreement">
        I accept <a href="#">user agreement</a>
      </label>
    </div>

    <div class="form-check">
      <input class="form-check-input" type="checkbox" value="1" id="newsletter">
      <label class="form-check-label" for="newsletter">
        I'd like to receive newsletters
      </label>
    </div>
  </div>
</div>
```

And that's our final version of the form:



5-12. Completed Bootstrap horizontal form with checkboxes built with Bootstrap grid classes

All elements have nice spacing so the UI is not visually cluttered. The source

code for this example can also be found on CodePen.

Live Code

See the Pen SitePoint Bootstrap 4 Horizontal Form.

Inline Forms

Inline forms are also very common these days. Such forms may be employed to display search or sign in features. Let's see them in action by creating a "Subscribe to newsletter" form:

```
<div class="container">
  <p class="h1">Subscribe to our newsletter</p>

  <form class="form-inline"> <!-- #1 -->
    <label class="sr-only" for="email">Name</label> <!-- #2 -->
    <input type="text" class="form-control mb-2 mr-sm-2 form-control-sm"
    id="email" placeholder="Email">  <!-- #3 -->

    <div class="form-check mb-2 mr-sm-2">  <!-- #4 -->
      <input class="form-check-input" type="checkbox" id="partners" value="1">
      <label class="form-check-label" for="partners">
        Receive news from our partners
      </label>
    </div>

    <button type="submit" class="btn btn-primary mb-2 btn-sm">Subscribe</button>
    <!-- #5 -->
  </form>
</div>
```

The main things to note here are:

- #1. The form must be assigned a Bootstrap form component `form-inline` class. Note, however, that the form elements will appear inline only for

viewports larger than 575px. For smaller screens, they'll be stacked.
- #2. We're hiding the label and displaying it only for screen readers.
- #3. The input has right margin for small and larger viewports with the help of the `mr-sm-2` class. We don't require this margin for extra small screens because the elements will be stacked, as explained above. Apart from that, the input is made smaller with the `form-control-sm` class.
- #4. The checkbox also has a right margin.
- #5. The button is made smaller as well using the `btn-sm` class.

Here's the result:

5-13. Bootstrap inline form component

Live Code

See the Pen SitePoint Bootstrap 4 Inline Form.

Form Validation

Another very important thing is validation of users' input. As a rule of thumb, we should never blindly trust the data submitted by users because they can make mistakes and provide incorrect values. This, in turn, might lead to the web application not working correctly, or worse. So we should always perform validations.

The first line of defense is *client-side* validation. Of course, we could rely on the browser's predefined validation rules (for instance, modern browsers check the email's format entered in the field with the corresponding type) but it's not generally recommended by Bootstrap. Instead, we can make use of the

Bootstrap form component's custom validation styles.

To take advantage of these styles, we should disable the browser's default validation mechanism by adding the `novalidate` attribute to the form. Also, we should add a `needs-validation` class:

```
<div class="container">
  <p class="h1">Register</p>

  <form class="needs-validation" novalidate> <!-- #1 -->
    </form>
</div>
```

Next, we simply display all the necessary fields as usual:

```
<form class="needs-validation" novalidate>
  <div class="form-group">
    <label for="email">Email</label>
    <input type="email" class="form-control" id="email"
           placeholder="Email" required> <!-- #1 -->
  </div>

  <div class="form-group">
    <label for="password">Password</label>
    <input type="password" class="form-control" id="password"
       placeholder="Password" required minlength="6"> <!-- #2 -->
  </div>

  <div class="form-group mt-3">
    <button class="btn btn-primary" type="submit">Register!</button>
  </div>
</form>
```

Note that we've provided the `require` attribute for both inputs (#1, #2) as well as `minlength` for the password field (#2). This will make sure that users don't send an empty form.

What's interesting is that each input may also contain two optional Bootstrap form component help blocks: `valid-feedback` and `invalid-feedback`. The first one is displayed when the data is correct, whereas the latter is shown only if users make a mistake. Let's add these blocks now:

```
<div class="form-group">
  <label for="email">Email</label>
  <input type="email" class="form-control" id="email"
    placeholder="Email" required>
  <div class="valid-feedback"> <!-- #1 -->
    Looks good!
  </div>
  <div class="invalid-feedback"> <!-- #2 -->
    The email is required!
  </div>
</div>

<div class="form-group">
  <label for="password">Password</label>
  <input type="password" class="form-control" id="password"
     placeholder="Password" required minlength="6">
  <div class="valid-feedback">
    Great!
  </div>
  <div class="invalid-feedback"> <!-- #3 -->
    The password must contain at least 6 characters!
  </div>
</div>
```

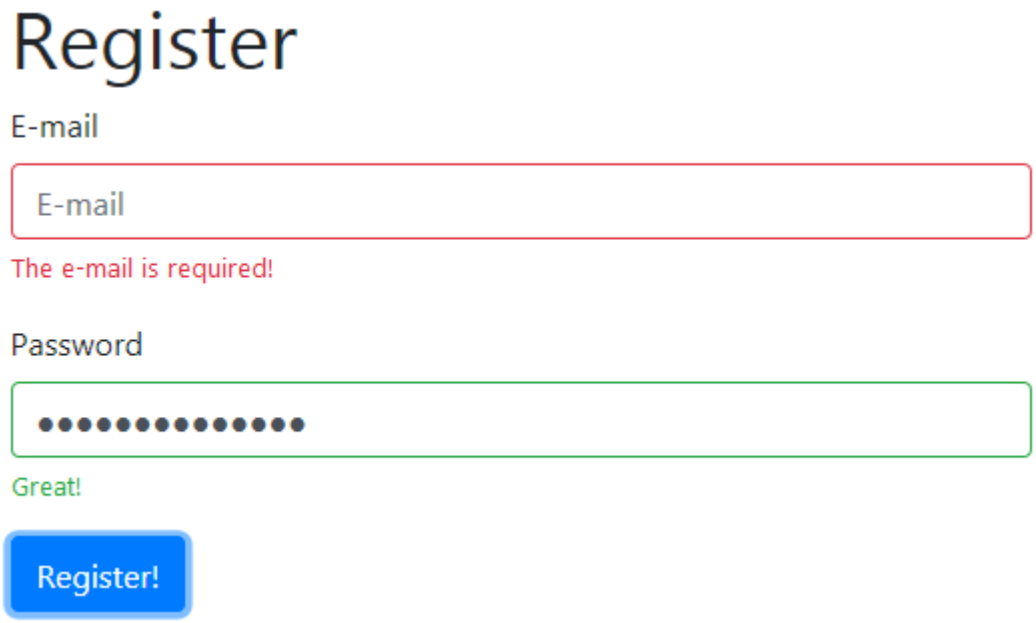
Lastly, we need some JavaScript to perform the actual validation and prevent the form from being submitted if an error is found:

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

<script>
  (function() {
    window.addEventListener('load', function() {
      var forms = document.getElementsByClassName('needs-validation');
```

```
      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>
```

Here, we grab all the forms that require validation and check if the entered data is correct. If it's not, we prevent the submission process. Also, note that we must add a Bootstrap form component `was-validated` class to the form. Each child field is also assigned either an `:invalid` or `:valid` pseudo class. These classes add a green or a red border around the input accordingly and display the proper feedback message.

Here is the result:



5-14. Bootstrap form with validation in action

Live Code

See the Pen SitePoint Bootstrap 4 Form Validation.

This is how the style applied to the `:valid` pseudo class looks in the browser's developer tools:

```
.custom-select.is-valid,     bootstrap.css:2202
.form-control.is-valid,
.was-validated .custom-select:valid, .was-
validated .form-control:valid {
  border-color: #28a745;
}
```

5-15. Bootstrap validation classes in the browser's developer tools

Lastly, note that the Bootstrap form component supports server-side validation as well. All we have to do is mark the inputs with either the `is-valid` or `is-invalid` classes. Feedback blocks also work fine with this type of validation.

Conclusion

In this chapter, we've taken quite a long journey through the features of the Bootstrap form component and discussed various types of forms that come bundled with the latest release of Bootstrap. We've seen how to style various form elements, how to create forms based on the Bootstrap grid system, how to craft horizontal and inline forms. On top of that, we've discussed how the client-side validation rules can be implemented and how to style valid and invalid form inputs.

I really hope that by now you're ready to apply all this knowledge into practice and start building your own fancy forms!

Chapter

6

The Card Component: A Complete Introduction

Ahmed Bouchefra

In this chapter, I'm going to introduce you to the Bootstrap card component and walk you through its many features and uses. By the end of this tutorial, you'll be able to incorporate Bootstrap cards in your web projects for great layouts and page content organization.

Bootstrap is the most popular HTML, CSS, and JavaScript framework for quickly creating responsive and mobile-first web layouts.

Using Bootstrap offers many advantages to developers, including the following:

- you can easily and quickly build a responsive layout by having only some knowledge of HTML and CSS
- the library was built to be mobile-first from the core
- it's compatible with all modern browsers etc.
- it uses Flexbox for its grid system
- it provides tons of features and components.

The latest release of Bootstrap offers many new, modern features — such as the support for Flexbox, and the new card component, which replaces panels, thumbnails and wells from earlier versions of the library.

What Is the Bootstrap Card Component?

Bootstrap introduces a new UI component for creating cards which provides a flexible and stylish container for showing content. Cards come with minimal styling, but you can easily extend them with extra styling options.

This component is built on top of Flexbox, and you can use the Bootstrap spacing utility classes for margins and padding.

Cards can conveniently accommodate different types of content like title, subtitle, body copy, and images etc., and optional header and footer sections. You can also include different sections (card blocks) to construct different types of cards depending on your use.

Including Bootstrap in Your Project

You can very easily start using Bootstrap in your website by either including it from a CDN or downloading it from getbootstrap.com.

For the sake of introducing the new Bootstrap card component, you're going to create a simple HTML page with Bootstrap styling to demonstrate the basic elements of the card component. Go ahead and create the `index.html` file inside your project folder, then add the following content from the docs starter template:

```
<!doctype html>
 <html lang="en">
   <head>
     <!-- Required meta tags -->
     <meta charset="utf-8">
     <meta name="viewport" content="width=device-width, initial-scale=1,
     ↪shrink-to-fit=no">

     <!-- Bootstrap CSS -->
     <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/
     ↪css/bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTN
     ↪h0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">

     <title>Bootstrap Card Example</title>
   </head>
   <body>
       <div class="container">
     </div>

     <!-- Optional JavaScript -->
     <!-- jQuery first, then Popper.js, then Bootstrap JS -->
     <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha
     ↪384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN"
     ↪crossorigin="anonymous"></script>
     <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/
     ↪popper.min.js" integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX
```

```
      39j7fakFPskvXusvfa0b4Q" crossorigin="anonymous"></script>
      <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.
      ↪js" integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar
      ↪5+76PVCmYl" crossorigin="anonymous"></script>
    </body>
  </html>
```

We've included Bootstrap files and their dependencies (jQuery and popper.js) from different CDNs, then created a container division.

Minimal Styling for Cards

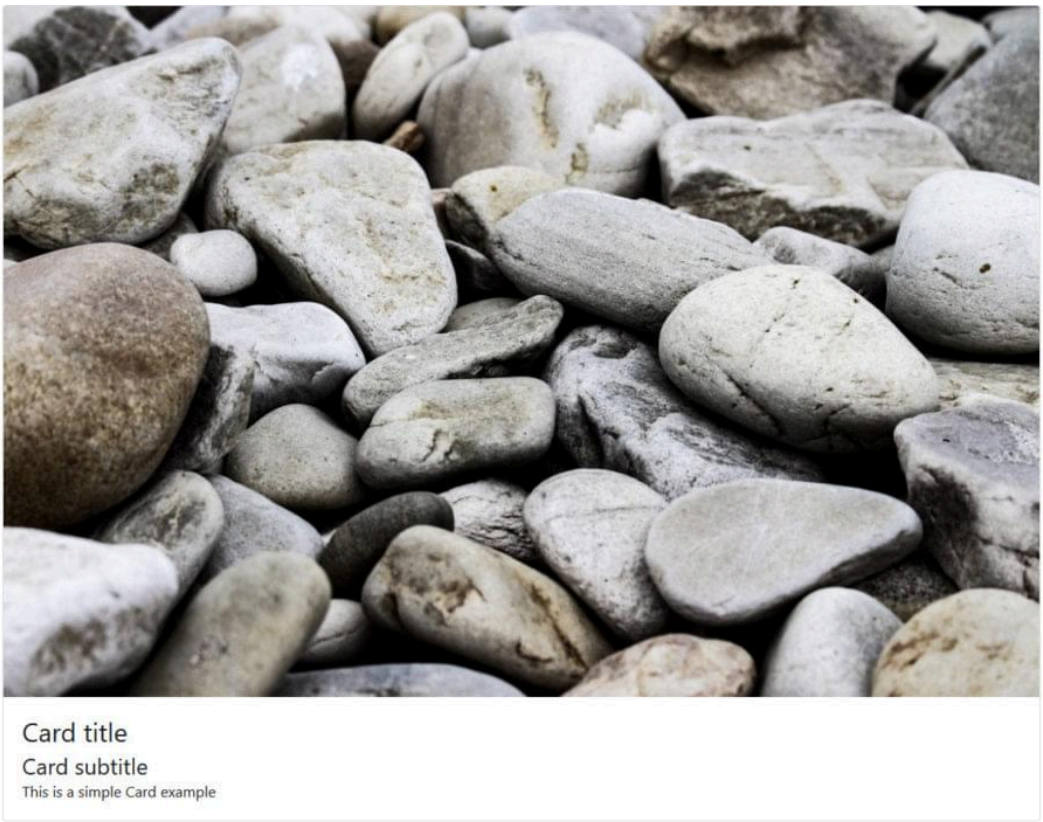
In order to create a basic card, we need to —

- use `.card` with a `div` tag to create the outer container
- add the `.card-body` class to an inner `div` tag to create the card body
- use `.card-title` and `card-subtitle` classes with heading tags for adding the title and the subtitle
- use the `card-text` class with `<p>` tags to add textual content
- use the `card-img-top` class with the `<img>` tag to add an image to the top of the card.

These are the basic elements that constitute a full but basic card layout, as we can see from the following example:

```
<div class="card">
   <img class="card-img-top" src="https://source.unsplash.com/daily"
    alt="Card image top">
   <div class="card-body">
     <h3 class="card-title">Card title</h3>
     <h4 class="card-subtitle">Card subtitle</h4>
     <p class="card-text">This is a simple Card example</p>
   </div>
 </div>
```

This is a screenshot of the result:



6-1. Bootstrap card component: Basic card layout

As can be seen, the card takes the full width of its container `div`.

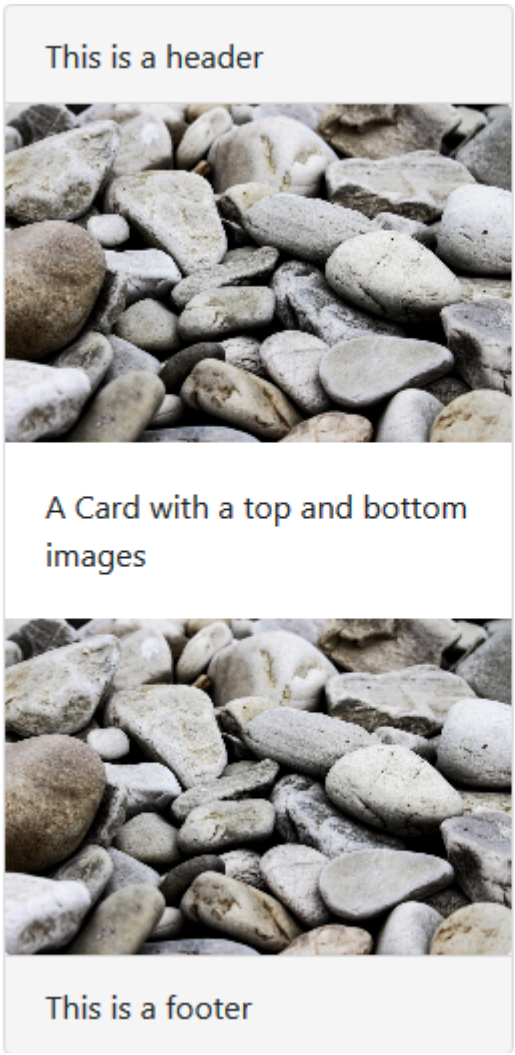
By simply switching the `card-body` class with the `card-img-overlay` class, we can use the image as an overlay:

6-2. Bootstrap card component: Using the image as an overlay

We can also use the class `.card-img-bottom` with the `<img>` tag to add the image at the bottom of the card:

```
<div class="card">
   <div class="card-header">
       This is a header
   </div>
   <img class="card-img-top" src="https://source.unsplash.com/daily"
    alt="Card image top">
   <div class="card-body">
       <p class="card-text">A Card with a top and bottom images</p>

   </div>
   <img class="card-img-bottom" src="https://source.unsplash.com/daily"
    alt="Card image bottom">
```

```
    <div class="card-footer">
            This is a footer
    </div>
</div>
```



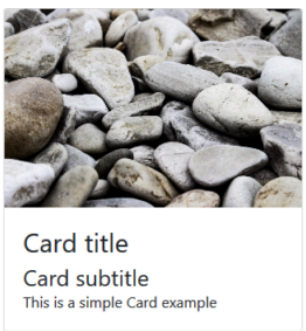
6-3. Bootstrap card component: Adding the image at the bottom of the card

The card top and bottom images are called image caps.

Controlling Bootstrap Card Component Width and Height

By default, cards take all the available width in their parent container, but you can use the `width` and `max-width` classes to control take control of the card's dimensions. So, let's change the previous example by reducing the width of the card:

```
<div class="card" style="width:20rem;">
   <img class="card-img-top" src="https://source.unsplash.com/daily"
    alt="Card image top">
   <div class="card-body">
     <h3 class="card-title">Card title</h3>
     <h4 class="card-subtitle">Card subtitle</h4>
     <p class="card-text">This is a simple Card example</p>
   </div>
 </div>
```



6-4. Bootstrap card component: Reducing card width

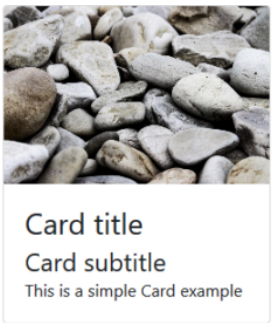
Normally, the height of the card will be adjusted to vertically fit the content of the card, but we can also control it using custom CSS (for example, `style="height: 10rem;"`) or Bootstrap's sizing utilities (for examle, `<div class="card h-200">`).

Please, note that the latest release of Bootstrap has switched to *rem* units

instead of *px* units because *rem* is a scalable measurement unit — so it works better with user settings, which makes text much more accessible. The result is that all elements in the page will scale with the screen size.

Another option for controlling the width of the Bootstrap card component is to use the Bootstrap grid (rows and columns):

```
<div class="row">
     <div class="col-sm-3">
         <div class="card">
             <img class="card-img-top" src="https://source.unsplash.com/daily"
                alt="Card image top">
             <div class="card-body">
                 <h3 class="card-title">Card title</h3>
                 <h4 class="card-subtitle">Card subtitle</h4>
                 <p class="card-text">This is a simple Card example</p>
             </div>
         </div>
     </div>
 </div>
```



6-5. Bootstrap card component: Controlling card width with Bootstrap grid

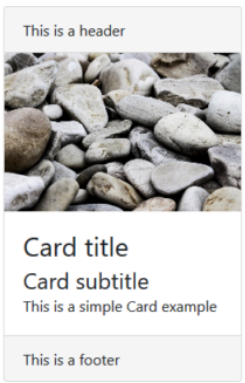
Bootstrap Card Component Header and Footer

A Bootstrap card component can optionally have a header and footer by adding heading (`<h*>`) and `div` tags with `.card-header` and `.card-footer` classes respectively.

Continuing with our example, let's add a header and footer to our Bootstrap card component:

```
<div class="row">
     <div class="col-sm-3">
         <div class="card">
             <div class="card-header">
                 This is a header
             </div>
             <img class="card-img-top" src="https://source.unsplash.com/daily"
                alt="Card image top">
             <div class="card-body">
                 <h3 class="card-title">Card title</h3>
                 <h4 class="card-subtitle">Card subtitle</h4>
                 <p class="card-text">This is a simple Card example</p>
             </div>
             <div class="card-footer">
                     This is a footer
             </div>
         </div>
     </div>
 </div>
```

This is the screenshot of the result:



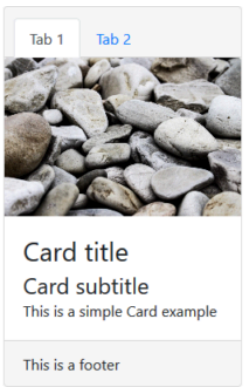
6-6. Bootstrap card component: Adding a header and footer

Adding Navigation

Another nice feature of the Bootstrap card component is the possibility of adding advanced navigation patterns to the header section such as tabs and navigation pills.

So let's change our simple example by adding a tabbed navigation element to the header section of the card with the `.nav-tabs` and `.card-header-tabs` classes on the `<ul>` tag:

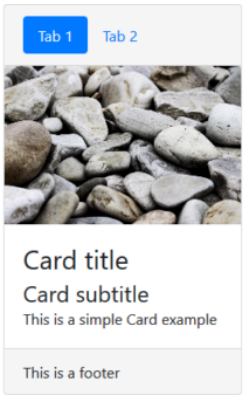
```
<div class="card-header">
   <ul class="nav nav-tabs card-header-tabs">
     <li class="nav-item">
         <a class="nav-link active" href="#">Tab 1</a>
     </li>
     <li class="nav-item">
         <a class="nav-link" href="#">Tab 2</a>
     </li>
   </ul>
 </div>
```



6-7. Bootstrap card component: Adding a tabbed navigation element

In the same way, we can add navigation pills by simply replacing `.nav-tabs` with `nav-pills` and `.card-header-tabs` with `card-header-pill` on the `<ul>` list tag:

```
<div class="card-header">
   <ul class="nav nav-pills card-header-pill">
     <li class="nav-item">
         <a class="nav-link active" href="#">Tab 1</a>
     </li>
     <li class="nav-item">
         <a class="nav-link" href="#">Tab 2</a>
     </li>
   </ul>
 </div>
```



6-8. Bootstrap card component: Adding navigation pills

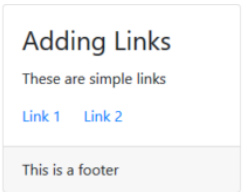
You can find out more information about this in the Bootstrap navigation components documentation.

Adding Links to the Bootstrap Card Component

We can quickly add links inside cards using the `<a>` tag with a `.card-link` class:

```
<div class="card">
 <div class="card-body">
   <h3 class="card-title">Adding Links</h3>
   <p class="card-text">These are simple links</p>
   <a href="#" class="card-link">Link 1</a>
   <a href="#" class="card-link">Link 2</a>
```

```
</div>
<div class="card-footer">
      This is a footer
</div>
</div>
```



6-9. Bootstrap card component: Adding links inside cards

Aligning and Transforming Text in Bootstrap Card Components

We can use different Bootstrap classes (text utilities) to align text within a card component such as `.text-left` (align text to left), `.text-right` (align text to right), `.text-center` (center text), `.text-justify` (justify text) and `.text-nowrap` (prevent text from wrapping).

We can also apply text transformation with `.text-lowercase` (transform text to lowercase), `.text-uppercase` (transform text to uppercase) and `.text-capitalize` (transform the first letter of each word to uppercase).

Customizing Bootstrap Card Component

Background, Foreground and Border Colors

A Bootstrap card component's background and foreground colors can be fully customized using Bootstrap text and background color utilities — `text-primary, text-white, bg-primary` and so on.

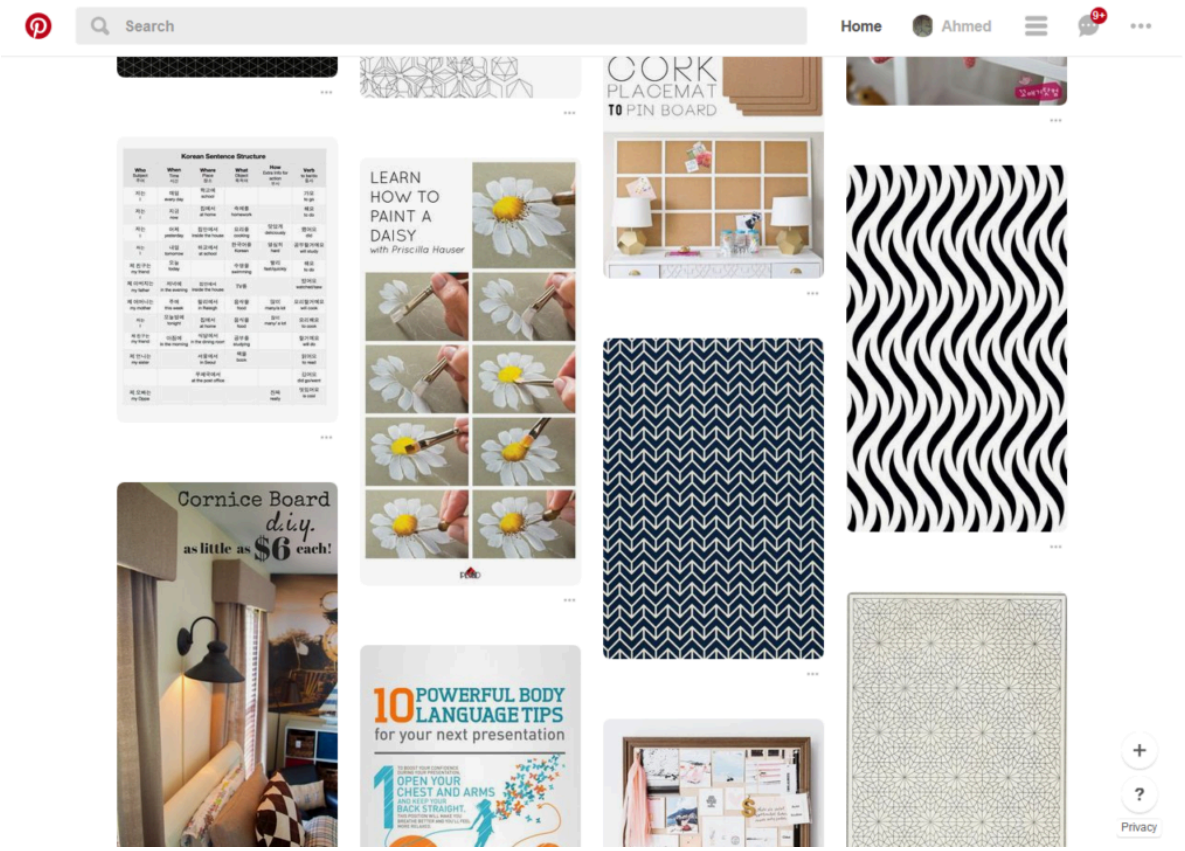
Using the Bootstrap's border utilities (for example, `border-primary`) we can quickly set the border color of a card.

Live Code

This CodePen demo shows all the different kinds of cards we can create with Bootstrap. Feel free to experiment with it on your own.

Creating Advanced Layouts with the Bootstrap Card Component

The card is a modern UI component that's utilized in many websites as a way to display items of mixed and different content types (such as text and images etc.) as a single/grouped entity, but more commonly as a collection of similar items with a different or the same width and height. A popular layout where cards can be used is the Masonry layout, also called the Pinterest-like Layout



6-10. A Pinterest card layout

Cards can be used for creating layouts for image albums, blog posts, ecommerce products etc. Big companies like Google and Facebook are making use of cards for their different web platforms.

Normally, if we want to create complex layouts based on cards, we'd need to have deep knowledge of CSS and HTML. However, thanks to the latest features of Bootstrap, we can quickly build such layouts by wrapping our set of cards inside `div`s with some special classes such as `.card-group`, `.card-deck` and `.card-columns`.

Grouping/Nesting Cards

Groups are used to display multiple Bootstrap card components as single and attached elements with uniform sizing — that is, cards have the same width and

height. This is achieved using the `display: flex;` property.

We can create a group of cards using the `.card-group` class with a wrapper `div` that contains the child cards.

Let's add the following code to create a group with three cards:

```
<div class="card-group">
   <div class="card text-white">
       <img class="card-img-top" src="https://source.unsplash.com/featured/"
        alt="Card image top">
       <div class="card-img-overlay">
           <h3 class="card-title">Card title</h3>
           <h4 class="card-subtitle">Card subtitle</h4>
           <p class="card-text">This is a simple Card example</p>
       </div>
   </div>
   <div class="card text-white">
       <img class="card-img-top" src="https://source.unsplash.com/featured/"
        alt="Card image top">
       <div class="card-img-overlay">
           <h3 class="card-title">Card title</h3>
           <h4 class="card-subtitle">Card subtitle</h4>
           <p class="card-text">This is a simple Card example</p>
       </div>
   </div>
   <div class="card text-white">
       <img class="card-img-top" src="https://source.unsplash.com/featured/"
        alt="Card image top">
       <div class="card-img-overlay">
           <h3 class="card-title">Card title</h3>
           <h4 class="card-subtitle">Card subtitle</h4>
           <p class="card-text">This is a simple Card example</p>
       </div>
   </div>
 </div>
```

This is a screenshot of the result:

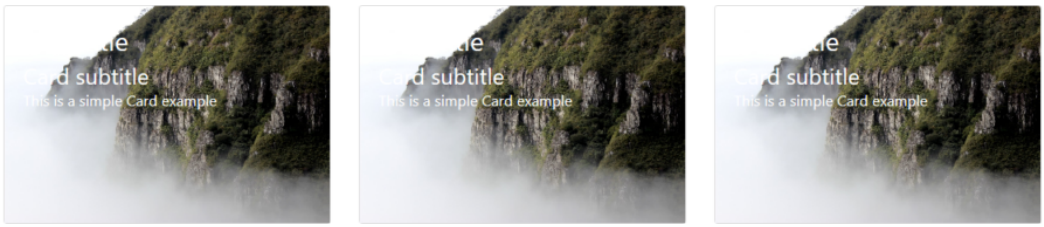
6-11. Bootstrap card component: Creating a group with three cards

As can be seen, the three cards are attached and have the same width and height.

Card Decks

Decks provide you with a layout similar to the previous one (that is, same width and height for each card) but with the one difference that cards are not attached to each other.

We can simply add a `.card-deck` class to a parent `div` element to create a card deck:



6-12. Bootstrap card component: Adding a .card-deck class to a parent div

As we can see, the cards are of the same size with some margin between them.

Card Columns

Just like groups and decks, card columns can be used to lay out a set of cards in a Masonry-like style. We can achieve this popular layout style simply by wrapping our cards in a `div` and adding the `.card-columns` class. That's all we need to do and Bootstrap will take care of the rest without having to use a JavaScript/jQuery

plugin.

Bootstrap uses CSS columns to create the Masonry-like layout where cards are added from top to bottom and left to right. However, if we need a different kind of behavior based on some algorithm, we'll have to resort to a JavaScript library or a jQuery plugin like Masonry.

Live Code

Here's a Masonry layout demo using Bootstrap cards in action.

Conclusion

The Bootstrap card component is a powerful addition to the Bootstrap framework, which allows developers to create modern-style web pages without going deeply into how CSS works. You can add card layouts to represent image galleries, dashboard widgets, to display blog posts or products for an ecommerce website, by just adding a bunch of CSS classes.

As a result of the new features and components, Bootstrap continues to be a powerful CSS framework available to everyone, particularly to developers who need to create their own responsive and modern-style layouts but lack enough time and budget, or the deep knowledge of CSS, necessary to produce custom code.

Chapter

7

How to Build a Responsive Type Scale with Bootstrap

Craig Watson

In this tutorial, we'll be taking an in-depth look at how Bootstrap handles typography and how we can modify the code in a couple of different ways to create a responsive type scale. This is often referred to as "responsive typography", the aim of which is to keep your typography readable on all screen sizes and avoid giant headings on mobiles!

How Bootstrap Sets Up Typography by Default

To understand the way Bootstrap typography works, we need to begin looking into the source `scss` files to explore the setup and default settings.

Note: for the sake of clarity throughout this tutorial, I've commented out styles from the Bootstrap code that are NOT *associated with typography.*

The `html` Element

Let's first look at styles for the root element, found in `_reboot.scss` on line 27:

```
html {
  font-family: sans-serif;
  line-height: 1.15;
  -webkit-text-size-adjust: 100%;
  -ms-text-size-adjust: 100%;
  // -ms-overflow-style: scrollbar;
  // -webkit-tap-highlight-color: rgba(0, 0, 0, 0);
}
```

From the `html` element there's not much to report in terms of setting up a *type scale*. However, it's worth noting the `-*-text-size-adjust: 100%;` declarations. These have been used to prevent some mobile browsers from increasing the font size of their own accord.

The `body` Element

Here are the `body` element styes, as found in `_reboot.scc` on line 57:

```
body {
  // margin: 0; // 1
  font-family: $font-family-base;
  font-size: $font-size-base;
  font-weight: $font-weight-base;
  line-height: $line-height-base;
  color: $body-color;
  text-align: left; // 3
  // background-color: $body-bg; // 2
}
```

Now we can start to see some typographic styes being applied. In terms of type scale, we only need to be concerned with `font-size`. By default, this is set via the `$font-size-base` variable found in `_variables.scss`, and is equal to `1rem`.

The `p` Element

These styles for the `p` element are found in `_reboot.scss` on line 109:

```
p {
  margin-top: 0;
  margin-bottom: $paragraph-margin-bottom;
}
```

Not much to report here: the `p` tag simply inherits its `font-size` and `line-height` from the body as you would expect.

The `h1` through `h6` Elements

These styles are found in `_type.scss` from lines 16 to 21:

```
h1, .h1 { font-size: $h1-font-size; }
h2, .h2 { font-size: $h2-font-size; }
h3, .h3 { font-size: $h3-font-size; }
h4, .h4 { font-size: $h4-font-size; }
```

```
h5, .h5 { font-size: $h5-font-size; }
h6, .h6 { font-size: $h6-font-size; }
```

You can see here the element and the utility class are given the `font-size` through a variable. The corresponding variables are found in `_variables.scss` on lines 246 to 251. Looking at these variables, we can see the default sizes set out to work across all browser and viewport widths:

```
$h1-font-size: $font-size-base * 2.5 !default;
$h2-font-size: $font-size-base * 2 !default;
$h3-font-size: $font-size-base * 1.75 !default;
$h4-font-size: $font-size-base * 1.5 !default;
$h5-font-size: $font-size-base * 1.25 !default;
$h6-font-size: $font-size-base !default;
```

Looking at a type scale for the above, it's clear that an increase of `.25rem` is used until the `h1`, which is given a `.5rem` increase.

These sizes can be overridden in a custom variables file if you're using a Sass compiler, but it still leaves you with one `font-size` for each heading across all browser and viewport widths.

The `.display-1` through `.display-4` Utility Classes

The following code is fond in `_type.scss` from lines 29 to 48:

```
// Type display classes
.display-1 {
  font-size: $display1-size;
  // font-weight: $display1-weight;
  // line-height: $display-line-height;
}
.display-2 {
  font-size: $display2-size;
  // font-weight: $display2-weight;
```

```
  // line-height: $display-line-height;
}
.display-3 {
  font-size: $display3-size;
  // font-weight: $display3-weight;
  // line-height: $display-line-height;
}
.display-4 {
  font-size: $display4-size;
  // font-weight: $display4-weight;
  // line-height: $display-line-height;
}
```

As with the heading elements, the display utility class sizes are defined as variables in the `_variables.scss` file on lines 259 to 262. Again, if you're working with a Sass compiler you can override these in a custom variables file.

That about covers the setup from Bootstrap, and we can now look at ways of making a responsive type scale that's quickly adjustable.

Creating the Responsive Type Scale

It's worth expanding on what I mean by a *Responsive Type Scale*. By default, Bootstrap `font-size` for headings and its `display-*` classes are explicitly set using variables found in `_variables.scss` and rules found in `_type.scss`.

Setting one `font-size` outright for headings across all screen and viewport sizes can quite quickly lead to oversized headings that make for a poor user experience.

You could, of course, create some media queries when it suits to pull down font sizes that look over-sized, but it's at this point that you lose any form of a type scale hierarchy. We *need type hierarchy* to follow the flow of the document.

In comes the *Responsive Type Scale* and a nice Sassy way to implement it into a Bootstrap project! If you don't use Sass or SCSS, you can simply update the pen I

use for examples and extract the compiled CSS.

An Overview of the Responsive Type Scale for Bootstrap

We are going to set up three main things:

- a type scale map for quick changes and experiments
- a function to check if the scale is valid for use
- two mixins to allow us the flexibility of adjusting the font sizes at any given time.

It's worth noting that, should your design not include a type scale that works on a common multiple, using the `mixin` won't work and you'll need to look at the compiled `CSS` in the pen to get the code you need to update the font sizes.

The Responsive Type Scales Map

Create a Sass map of pre-defined typographic scales, `$type-scales`, according to the model found on type-scale.com. The scales in the map can be passed to the Sass mixin that creates the font sizes by using their `key` from the `key: value` pairs.

After the map of scales, two variables are defined: `$heading-type-scale-base` and `$display-type-scale-base`. These variables hold the initial scales that are used from a zero-width viewport or browser and upward. These variables accept a key from the `$type-scales` map or can be passed a unitless value:

```
$type-scales : (
  minor-second: 1.067,
  major-second: 1.125,
  minor-third: 1.200,
  major-third: 1.250,
  perfect-fourth: 1.333,
  augmented-fourth: 1.414,
```

```
  perfect-fifth: 1.500,
  golden-ratio: 1.618
);

$heading-type-scale-base : minor-third;
$display-type-scale-base : minor-third;
```

How to Check the Responsive Type Scales Value

It's important that you aren't restricted to only the values in the map, as that may not be suitable for your design.

For this reason, the function below will check if the value passed to the mixin is one of the values set in the `$type-scales` map or it must be a unitless value to create the type scale:

```
@function check-type-scale-value($scale) {

  // Check $scale against the values in $type-scales.
  @if map-has-key($type-scales, $scale) {

    // If the value of $scale is defined in
    // $type-scales, return the value of $scale.
    @return map-get($type-scales, $scale);

  // If the value of $scale is not defined in the
  // $type-scales map, check if the value is a number
  // and that the number is a unitless value.
  } @else if type-of($scale) == number and unitless($scale) {

    // If the value of $scale is a unitless number,
    // return the number.
    @return $scale;

  // Lastly, should the value passed to $scale be neither
  // found in the $type-scales map nor a unitless number,
  // throw a Sass error to explain the issue.
```

```
  } @else {

    // Throw a Sass error if the $scale value is
    // neither found in the $type-scales map nor
    // a unitless number.
    @error "Sorry, `#{$scale}` is not a unitless number value or a pre-defined key
    ↪in the $type-scales map.";
  }

}
```

Next we will build up the mixins for creating the initial font sizes.

Creating the Heading and Display Font Sizes

I first wrote the `CSS` to achieve a responsive type scale when the alpha version of Bootstrap 4 was available. But since the release of the stable version of the library, I've revamped things to use the default `SCSS` setup, which makes it convenient to include the code in Bootstrap projects.

The first `mixin` is used to create the heading font sizes from `h6` to `h1`:

```
@mixin create-heading-type-scale($scale) {

    // Check the $scale value and store in a variable to be
    // used when calculating the font sizes.
    $the-heading-type-scale: check-type-scale-value($scale);

    // Starting from h6, multiply each previous value by the scale
    // to get the next font size
    $font-size-h6 : $font-size-base;
    $font-size-h5 : $font-size-h6 * $the-heading-type-scale;
    $font-size-h4 : $font-size-h5 * $the-heading-type-scale;
    $font-size-h3 : $font-size-h4 * $the-heading-type-scale;
    $font-size-h2 : $font-size-h3 * $the-heading-type-scale;
    $font-size-h1 : $font-size-h2 * $the-heading-type-scale;
    // $font-size-display-base is made global to allow for accessing the
```

```
    // varibale in the next mixin.
    $font-size-display-base : $font-size-h1 !global;

    // Add the created font sizes to the elements and classes
    h1, .h1 { font-size: $font-size-h1; }
    h2, .h2 { font-size: $font-size-h2; }
    h3, .h3 { font-size: $font-size-h3; }
    h4, .h4 { font-size: $font-size-h4; }
    h5, .h5 { font-size: $font-size-h5; }
    h6, .h6 { font-size: $font-size-h6; }
}
```

Above, the font sizes are first created and stored in variables starting from the `$base-font-size` and multiplying each previous value by the type scale value. The same principle is applied below but we start from the `$font-size-display-base` :

```
@mixin create-display-type-scale($scale) {

    // Store default type scale in a variable for calculations
    $the-display-type-scale: check-type-scale-value($scale);

    // Create variables to reference the previous font size
    $font-size-display-4 : $font-size-display-base + $font-size-base;
    $font-size-display-3 : $font-size-display-4 * $the-display-type-scale;
    $font-size-display-2 : $font-size-display-3 * $the-display-type-scale;
    $font-size-display-1 : $font-size-display-2 * $the-display-type-scale;

    // Add the created font sizes to the elements and classes
    .display-4 { font-size: $font-size-display-4; }
    .display-3 { font-size: $font-size-display-3; }
    .display-2 { font-size: $font-size-display-2; }
    .display-1 { font-size: $font-size-display-1; }
}
```

Drop the Root `font-size`

I like to drop the root `font-size` to `14px` for mobiles and work my way up to

`16px` and then `18px`. Below is the `SCSS` to do so using the breakpoint sizes of `md` and `lg`:

```
html {
    font-size: 14px;
    @media (min-width: 768px) {
        font-size: 16px;
    }
    @media (min-width: 992px) {
        font-size: 18px;
    }
}
```

Wrapping It Up

Once all this work is done, it's all about including your mixins and choosing your desired scales!

```
// Create the heading font sizes
@include create-heading-type-scale($heading-type-scale-base);

// Create the display font sizes
@include create-display-type-scale($display-type-scale-base);

// At the Bootstrap md breakpoint, adjust the heading font sizes.
@media (min-width: 768px) {
    @include create-heading-type-scale(minor-third);
}
```

As you can see, you can include the mixin at any breakpoint and completely change your type scale. Each font size can still be overridden if necessary and all looks good.

The Bootstrap Responsive Type Scale in Action

You can see the Bootstrap responsive type scale in action in this pen:

Live Code

You can see the Bootstrap responsive type scale in action in this pen.

I hope this tutorial has given you an insight into how you can customize Bootstrap to work for you by providing a better understanding of how to achieve a responsive type scale using this popular library.

Chapter

8

A Beginner's Guide to the Latest Bootstrap Utility Classes

Ilya Bodrov-Krukowski

Bootstrap 4 was released as stable version (at last!) and brought many new cool features for us. One of them is a set of upgraded Bootstrap utility classes that we'll focus on in this chapter.

Introducing Bootstrap Utility Classes

Bootstrap utility classes are meant to be applied to various elements on the page to quickly style them in some manner without the need to write custom CSS rules. In many cases, you just provide a class for your element and observe the result right away. If you've never employed utility classes before, fear not: in most cases they are quite simple, yet powerful. They are very convenient because you don't need to reinvent the wheel and can concentrate on more complex tasks when creating a web application.

There are lots of Bootstrap utility classes available that set the element's position, tweak its display and float properties, adjust margins and paddings, color the text and background, and more. A full list of these classes can be found on the getbootstrap.com official website, and in this article I'll cover some interesting Bootstrap utility classes that may greatly simplify your life as a developer. So, shall we start?

Flex

Flex is a new and much-anticipated feature of Bootstrap 4. It allows us to easily manage an element's layout and alignment with a bunch of classes. Note, however, that this feature relies on the `display: flex` property, which may not be supported by older browsers. Interestingly, Bootstrap's grid system now also relies on flexbox.

Flex is quite a big topic on its own and we won't discuss it thoroughly in this article. Let me just quickly show some results that you may achieve with this feature.

First, it's possible to easily control the direction of your elements (that is,

whether they'll be displayed left to right or right to left). For instance:

```
<div class="d-flex flex-row-reverse">
  <div class="p-2">Flex item 1</div>
  <div class="p-2">Flex item 2</div>
  <div class="p-2">Flex item 3</div>
</div>
```

This will render three elements aligned to the right, with the *first* element located on the *right-most* position:

8-1. Bootstrap utility classes: flex-row-reverse

The order of the elements may be further controlled with the classes like `order-N` where `N` is the position's number:

```
<div class="d-flex flex-nowrap">
  <div class="order-3 p-2">First flex item</div>
  <div class="order-2 p-2">Second flex item</div>
  <div class="order-1 p-2">Third flex item</div>
</div>
```

Note that this feature supports responsiveness with the classes like `order-sm-1`.

The elements can be also aligned in a specific manner on the X or Y axis. For example:

```
<div class="d-flex align-items-center bd-highlight mb-3">
  <div>Flex item</div>
</div>
```

This will center the item on the Y axis:

8-2. Bootstrap utility classes: flex-align-items

Float

Float is a new utility that replaces good old "pulls". In Bootstrap 3 you would write:

```
<div class="pull-left">Pull to the left!</div>
```

In Bootstrap 4:

```
<div class="float-left">Float to the left!</div>
```

A cool enhancement is that floats have become responsive too, meaning you don't need to craft your own styles to take care of various breakpoints. For instance, the following element will float to the right only on medium (or larger) screens:

```
<div class="float-md-right"></div>
```

Display

The display property has also been fully reworked in Bootstrap 4. It now supports all display values (from `none` to `flex`) and is fully responsive. The corresponding classes look like `.d-{breakpoint}-{value}`:

- `breakpoint` may have values of `sm`, `md`, `lg`, or `xl`. If this value is skipped,

then `xs` is used by default, which effectively means the display will have the same value on all viewports.

- The `{value}` is any of those supported by the `display` property.

This means that you may hide an element for chosen viewports like this:

```
<div class="d-lg-none">Hide on screens wider than lg</div>
```

What's more, this utility works with print layouts too. For instance, an element may be hidden for printing:

```
<div class="d-print-none">Screen Only (Hide on print only)</div>
```

Sizing

With Bootstrap 4 you may also control the sizing of your elements with ease. Suppose we want to display a `div` that occupies 25% of the parent's width:

```
<div class="w-25">Width 25%</div>
```

What about the height? That's possible too:

```
<div class="h-25">Height 25%</div>
```

Want to provide `max-height` or `max-width`? Utilize either `mh-100` or `mw-100` classes. As you see, nothing complex!

Spacing

The Spacing feature presents a wide range of small but handy utilities that can be used to quickly set margins and paddings for any element. Lazy developers

like me will surely love this addition!

The corresponding classes look like `{property}{sides}-{size}`:

- `{property}` is either `m` (margin) or `p` (padding).
- `{sides}` can have values like `t` (top) or `l` (left). If it has a blank value, the sizing will be applied to all four sides of the element.
- `{size}` is an integer from `0` to `5`, or `auto`. This integer specifies a multiplier passed to the formula that calculates the resulting sizing `($spacer * MULTIPLIER)`. The default value of `$spacer` is `1 rem`. For example `1` corresponds to the `0.25` multiplier, `2` corresponds to the `0.5` multiplier, `3` to the value of `$spacer` alone, `4` corresponds to the `1.5` multiplier, and finally `5` corresponds to the `3` multiplier.

The following code means that the element will have left and right margins with the value of `1rem` (`x` means "X axis"):

```
<div class="mx-3"></div>
```

8-3. Bootstrap utility classes: spacing

Text

Text utilities were available in Bootstrap 3, but in the new version they've become even more powerful and convenient.

Alignment

As before, you can specify the alignment of text and, for example, make it centered:

```
<p class="text-center">Text is centered!</p>
```

But, what's more, these utility classes are also responsive now, which means we can make this text centered only on extra-large viewports:

```
<p class="text-xl-center">Text is centered only on very large screens!</p>
```

Transform

Transform is a new, small feature that may change the text to lowercase, uppercase or capitalize it. For instance:

```
<p class="text-capitalize">capiTaliZed</p>
```

This will display our text as "CapiTaliZed" (only the first "c" letter is uppercased):

CapiTaliZed

8-4. Bootstrap utility classes: text-transform

Font Weights

Changing a font's weight is now as simple as it can be:

```
<p class="font-weight-bold">Bold text.</p>
<p class="font-weight-light">Light weight text.</p>
```

Bold text.

Light weight text.

8-5. Bootstrap utility classes: font-weight

Colors

You can now make your website brighter with a collection of Bootstrap utility classes for color! This feature was already available in the previous version of Bootstrap, but now it provides some new classes and options. For instance, we may now combine the color and background in the following way:

```
<p class="text-danger bg-dark">Text</p>
```

This will display red text on a black background:

8-6. Bootstrap utility classes for text and background colors

All in all, classes that change the text's color itself look like `text-{color}`. Background can be changed with classes like `bg-{color}`.

There is also support for background gradients, but it must be enabled by explicitly setting the `$enable-gradients` Sass variable to `true`.

Borders

This is a new and quite cool feature of Bootstrap 4 that allows you to quickly style borders of any element (and even each border individually!).

For example, the following code will add a grey border for the element:

```
<div class="border"></div>
```

8-7. Bootstrap utility classes: border

If, for some reason, you're interested only in the top border, then write this:

```
<div class="border-top"></div>
```


8-8. Bootstrap utility classes: border-top

The classes look like `border-{side}`, where `side` can have the following values:

- `left`
- `right`
- `bottom`
- `top`
- no value (in this case border will be set for all sides)

We can be even more specific by utilizing so-called subtractive border classes. These classes look pretty much the same as the ones listed above, but must end with a `-0` postfix: `border-{side}-0`. So, if we say:

```
<div class="border-top-0"></div>
```

our element will have all borders except for the top one. You can further combine these classes and, for instance, display only the left and bottom borders:

```
<div class="border-top-0 border-right-0"></div>
```


8-9. Bootstrap utility classes to hide specific borders

Quite neat, huh?

Color 'Em All!

What's more, the border color can also be changed with the help of Bootstrap utility classes. The available colors correspond to the ones listed on the colors page.

So, for instance, we may say the following to make the element's text appear in red:

```
<div class="border border-danger"></div>
```

8-10. Bootstrap utility classes to color borders

Tweak the Radius

Lastly, the radius of any element can be modified as well with the help of the following classes:

- `rounded`

- `rounded-top`
- `rounded-left`
- `rounded-right`
- `rounded-bottom`
- `rounded-circle`
- `rounded-0`

`rounded-circle` sets `border-radius` of an element to `50%`, whereas `rounded-0` cancels all the rounding. Other classes make the corresponding border slightly rounded by setting the value to `0.25rem`:

8-11. Bootstrap utility classes for the border-radius CSS property

Embeds

Embeds is yet another new Bootstrap utility classes feature that might greatly simplify things for you. As the name implies, it allows you to create responsive embeds that have specific *ratio* scaling on various devices.

What does that mean? Suppose you'd like to display a YouTube video on your website. You can do it like this:

```
<div class="embed-responsive embed-responsive-16by9">
  <iframe class="embed-responsive-item" src="SOME_URL_HERE"></iframe>
</div>
```

This will make sure that the video's ratio is always 16:9 and it will scale properly! There are other ratios available:

- `embed-responsive-21by9`
- `embed-responsive-16by9`
- `embed-responsive-4by3`
- `embed-responsive-1by1`

Close Icon

Last but not the least is this tiny utility that does what it says: displays a close icon. One thing to remember is to specify some text for screen readers in the following way:

```
<button type="button" class="close" aria-label="Close">
  <span aria-hidden="true">&times;</span>
</button>
```

It looks like this:

8-12. Bootstrap utility classes for the close icon

Conclusion

In this chapter, we've discussed Bootstrap utility classes that simplify styling of various elements on the web page. We've covered utilities like flex, display, border, and seen some examples of their usage. Of course, there's more to these classes, so be sure to browse the official documentation, where you'll find even more useful examples.

Chapter

9

3 Tips for Speeding Up Your Bootstrap Website

Maria Antonietta Perna

Website optimization is a cluster of techniques, both server-side and client-side, which aim at speeding up websites' loading time and rendering in the browser for a better user experience.

Today, visitors are used to the immediacy and reactivity typical of native apps. They expect a web page to load within 1000ms. If it takes much longer than this, they're likely to leave the site.

Search engines have certainly caught up with this trend, and in many ways they've contributed to it. In fact, Google has been using page speed as a ranking factor for websites in desktop searches for a while now. Google has announced that, from July 2018, mobile searches will also be super important to achieve top positions in its result list.

Bootstrap has often been criticized for adding unnecessary bloat to websites, so if you use this popular front-end UI library in your project, make sure you pay extra attention to page weight and page speed.

In this chapter, I'll go through three **front-end optimization** steps you can take to ensure your Bootstrap-based website is fast-rendering and well optimized.

Only Download the Bootstrap Package You need

If you decide to work with the precompiled download package of Bootstrap, you should really have a serious think about which parts of the library you really need.

The download folder contains both the complete CSS library (`bootstrap.css` and `bootstrap.min.css`) and JavaScript components library with all its dependencies except for jQuery (`bootstrap.bundle.js` and `bootstrap.bundle.min.js`), as well as a number of standalone CSS files containing the code necessary for specific parts of this popular UI kit.

If you just need a nice CSS reset for your project, simply use

`bootstrap-reboot.min.css`. If you only need a flexible and easy-to-use grid system, then go for `bootstrap-grid.min.css`. There's no need to download the entire framework. If, on the other hand, you know you're going to use everything in the library, at least make sure you include the minified version.

Similarly with the JavaScript code. If you know you're not going to have dropdowns, popovers and tooltips in your project, then use `bootstrap.min.js` rather than `bootstrap.bundle.min.js` because you won't need to include Popper.js.

Opt for the Source rather than the Precompiled Download Package

As much as the latest release of Bootstrap lets you select parts of it to include in your project, the precompiled files might still contain stuff that you might not actually need.

Browsers still have to download and process unused code, which could impact on website performance, especially on slow network connections.

A better idea would be to download the Bootstrap source code, because:

- you'll be able to include exactly the components you need in your project
- customizing any part of the library becomes cleaner and more efficient, with no need to override styles over and over
- the stylesheet that ends up in production is usually leaner.

Make Use of Proven Client-side Optimization Techniques

Apart from the points made above, optimizing a website built on top of

Bootstrap for performance still has to incorporate front-end performance techniques just like any other website.

Below are just some of the critical factors you could pay attention to for effective front-end optimization of your website.

Write Lean CSS and JavaScript

Each character in your code adds up to the final weight of the web page. Writing clean and concise CSS and JavaScript code while keeping it readable is not always easy. However, it should be something to strive for in every project.

Good CSS practices include getting rid of unused selectors, duplicate code, and overly-nested rules. It's good to keep your code well organized at the beginning of a project. For example, using style guides can really benefit your development process and the quality of your code.

Also, there are great tools out there to help you clean up you code. A linter like CSS Lint and JSLint can check your document for syntax errors, inefficient coding patterns, unused code, etc.

Minify and Concatenate CSS and JavaScript Code

An important optimization step is to limit the number of HTTP requests a website has to make to render its content. Each round trip to the server and back to fetch resources takes time, thereby negatively impacting on the user experience.

Minifying (that is, removing comments and white space from your document) and **concatenating** CSS and JavaScript files have now become a consolidated practice that aims to keep file size small and decrease the number of HTTP requests.

If you'd like to delve deeper, How to Optimize CSS and JS for Faster Sites by Gary

Stevens is a great read.

Watch Out for Your Images File Size

The weightiest part of a web page is often represented by image files, but also audio and video files play their part. Optimizing visual assets is therefore crucial to website performance.

Doing so involves two aspects:

- Making sure you use the right image format for the job at hand.
- Squeezing excess bytes from your assets before uploading them for production. There are great tools out there that can help you. Check out online tools like TinyPNG for raster images (PNG, JPG, etc.) and Jake Archibald's SSVGOMG for SVG optimization. Also, consider tools you can install locally like your favorite task runner (Grunt, Gulp, etc.).

Conclusion

A fast-rendering website is a core factor in determining great user experience on websites. This becomes even more crucial when it comes to evaluating web user experience on mobile devices.

In this chapter, I've listed a number of techniques that play a role in optimizing a Bootstrap website for performance from the point of view of front-end development.

Chapter

10

Customizing Bootstrap jQuery Plugins

Maria Antonietta Perna

Bootstrap comes packed with tons of ready-to-use jQuery-powered plugins. The beauty of Bootstrap plugins is that you can slot them into your project, write the appropriate markup using `data-` attributes, and the JavaScript kicks in automatically. No need to write any JavaScript code.

There may be times when you'd like to add your own customizations to Bootstrap's plugins to fit your project's requirements.

If this is you, read on. I'm going to show you one way in which you can do just that.

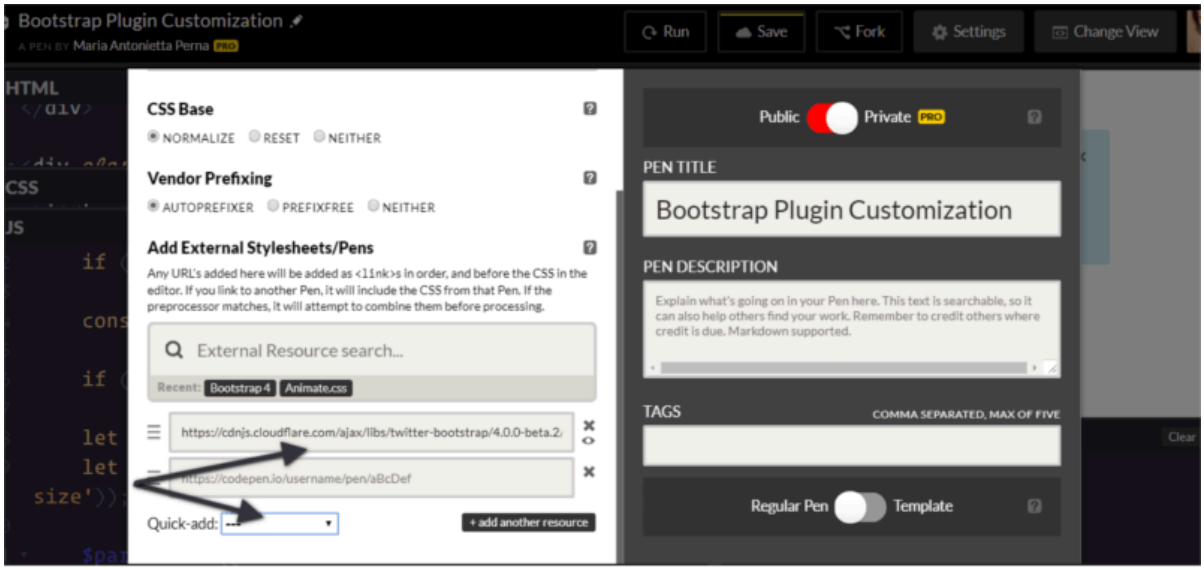
How to Customize the Appearance of Bootstrap Plugins

You can customize Bootstrap plugins' look and feel by working with CSS. On the other hand, if you'd like to modify the plugin's *functionality*, you'll need to write some JavaScript code.

Customizing Bootstrap CSS should be done in your own custom file, whether you're working with the Sass source code or with the compiled version.

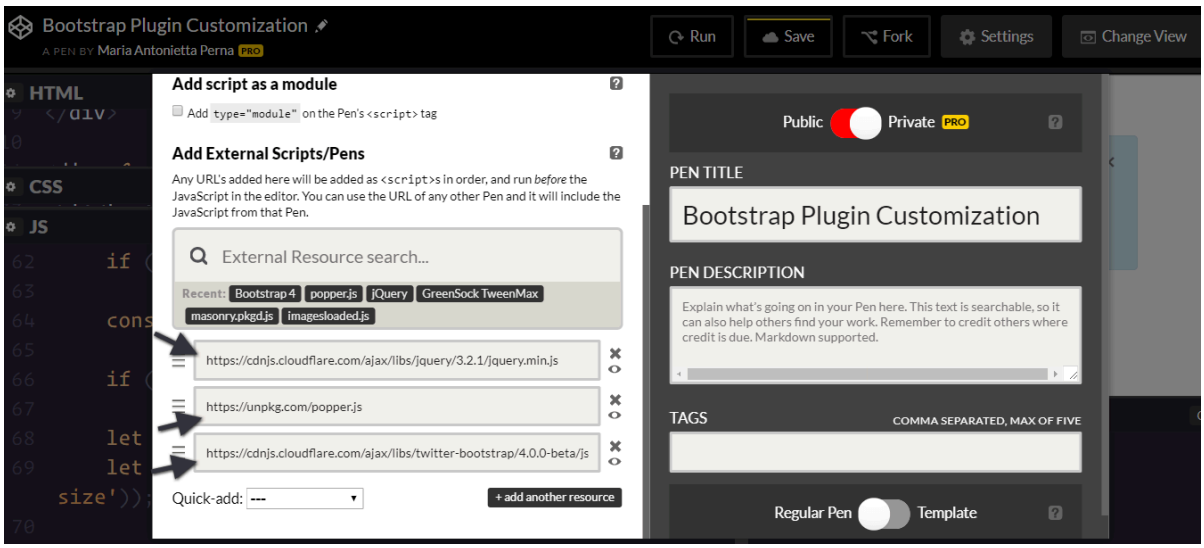
To keep things simple and CodePen-friendly, pull the precompiled Bootstrap package from the CDN.

This includes both the stylesheet ...



10-1. Bootstrap Plugin CSS file pulled from a Codepen demo.

... and the JavaScript file. Also, don't forget to include jQuery and, if you plan to use dropdowns, Popper.js too:



10-2. Bootstrap library and its dependencies pulled from a CodePen demo.

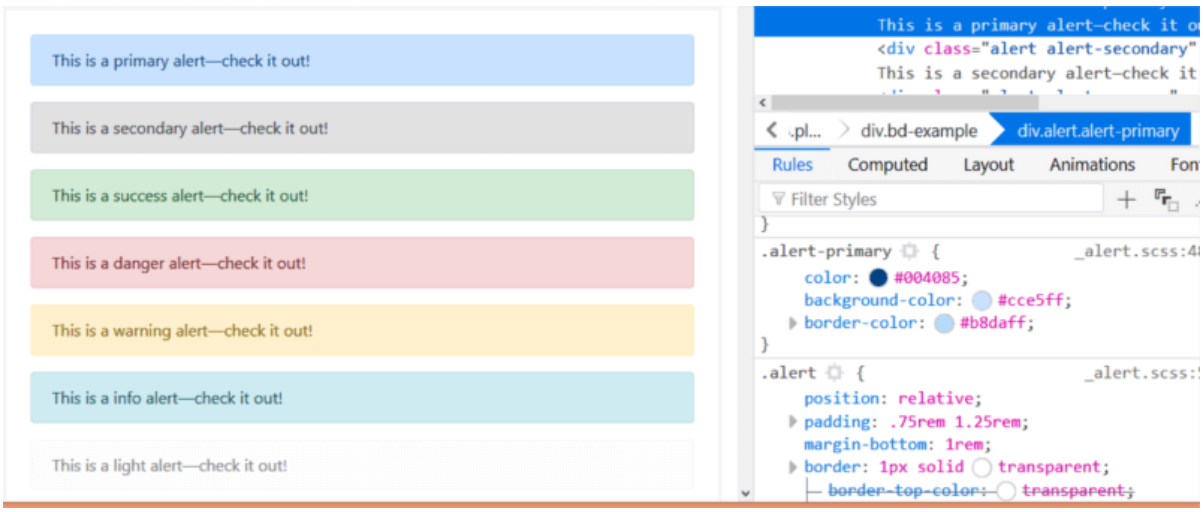
To make it even easier to modify CSS code, the developers behind Bootstrap have included a list of CSS variables for colors and breakpoints inside the `:root` element at the top of the stylesheet.

For example, the variable for the green success color is `--success: #28a745;`

To change this into a different shade of green, replace the variable's color value in your own stylesheet and add this code to your chosen selector:

```
selector {
  background-color: var(--success);
}
```

To change the look and feel of a specific plugin, the easiest way is to find the Bootstrap styles using your browser's devtools and overriding them with your own styles.



10-3. Customizing Bootstrap CSS with the browser web tools.

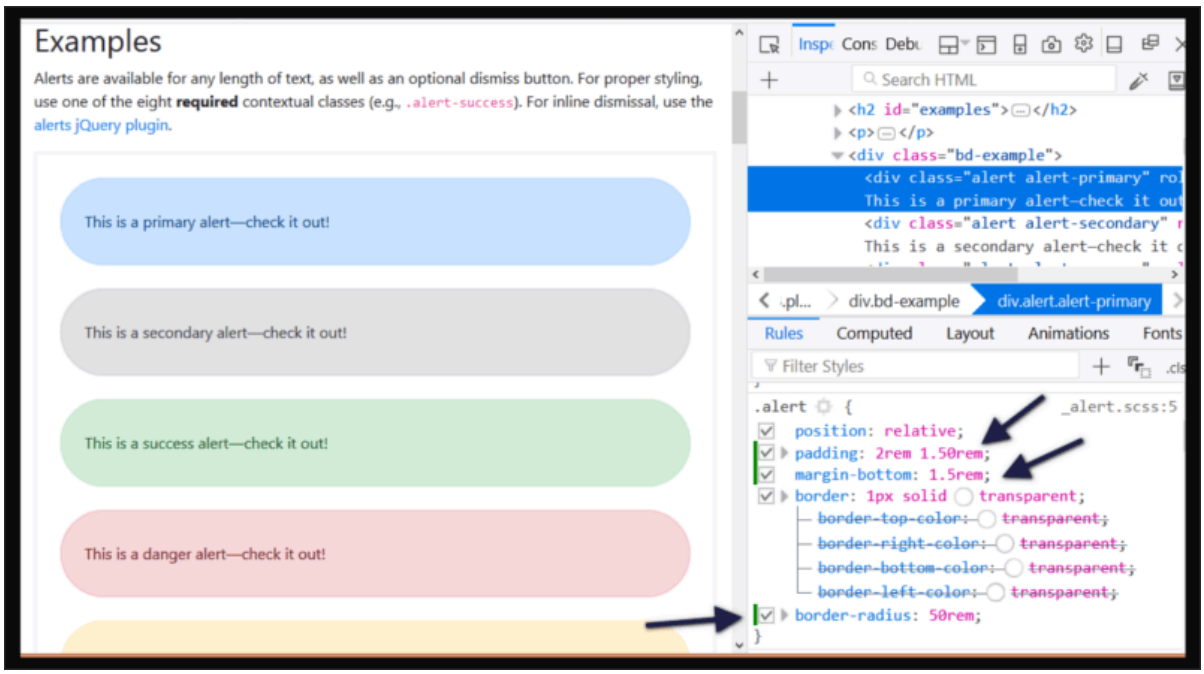
Let's focus on the simple alert component in this article.

For instance, try changing the alert's original values for the `padding`, `margin-bottom`, and `border-radius` in your custom CSS document:

```
.alert {
  padding: 2rem 1.50rem;
  margin-bottom: 1.5rem;
```

```
  border-radius: 50rem;
}
```

Now, all elements with the `.alert` class will look like this:



10-4. Customization of Bootstrap alert component's CSS.

Next, let's move on to extending the Bootstrap alert's functionality.

Customizing a Bootstrap Plugin's Functionality

Modifying Bootstrap plugins' functionality often involves working both with the CSS and JavaScript document.

Most dynamic effects are produced by adding and removing CSS classes, and overall the functionality is quite simple.

In this chapter, I'm going to add some extra functionality to the Alert component.

Alerts display a piece of information and users can remove them from the page

at the click of a button.

I'm going to add *zoom in* and *zoom out* buttons to the alert, which enable users to make the alert bigger or smaller respectively. I'm also going to include a *high contrast* button to allow users to view the alert contents in high contrast mode, i.e., inverting the background color and the text color.

Let's get to work!

The Alert's Structure

To follow along, start by adding the markup for the alert:

```
<div class="alert alert-info alert-zoom-in-out">
  <a href="#" class="close" data-dismiss="alert" aria-label="close">×</a>
  <a href="alert" class="close zoomIn" data-zoomIn="alert"
    aria-label="zoom in">+</a>
  <a href="alert" class="close zoomOut" data-zoomOut="alert"
    aria-label="zoom out">-</a>
  <a href="alert" class="close high-contrast" data-highContrast="alert"
    aria-label="high contrast mode"><i class="fa fa-adjust"></i></a>
  <p><strong>New Bootstrap Alert</strong></p>
  <p>Zoom in and out, or enable contrast mode.</p>
</div>
```

Above is a simple alert box with three links: the first one is a native Bootstrap *dismiss alert* link. This serves as template for the custom three links you're going to create. The - link lets users shrink the size and contents of the alert box, while the + link does the exact opposite — that is, enlarge the alert box and the text inside it.

The contrast link enables high contrast mode on the alert.

Notice the `data-zoomIn`, `data-zoomOut`, and `data-highContrast` on each custom link's markup. These are our hooks for the click event: a button with those

custom data types will be responsible for triggering the actions on the alert box.

Modifying the CSS

The next step is to create your own custom CSS file for the alert plugin and add the following code:

```
.alert-zoom-in-out {
  font-size: 16px;
  height: auto;
  top: 20px;
  color: rgb(12, 84, 96);
  background-color: rgb(209, 236, 241);
}

.high-contrast-on,
.high-contrast-on a,
.high-contrast-on i {
  background-color: rgb(12, 84, 96);
  color: rgb(209, 236, 241);
}
```

`alert-zoom-in-out` is the class I've chosen for the alert box. Because the functionality you're building includes modifying the size and color combination of the alert box, you need to set those for the `alert-zoom-in-out` class.

The `high-contrast-on` class on the alert box will be added and removed dynamically using JavaScript. When added, `color` and `background-color` values are inverted. When removed, the alert box will get back to its default appearance.

Adding Zoom-in Functionality

There's usually more than one way of achieving the same result in programming. My approach owes a lot to Benjamin Jakobus and Jason Marah. For this simple customization, I've chosen to take the public `close()` method on the Bootstrap `alert.js` file as my reference point.

Start by wrapping the code inside an immediately invoked function expression (IIFE), which avoids polluting the global scope with the plugin's code. Also, add the function to the jQuery object:

```
(function ($) {
  'use strict';
})(jQuery);
```

The Bootstrap JavaScript docs state that each plugin exposes its raw constructor on a `Constructor` property, which you can access in your code to add custom functionality:

```
(function ($) {
  'use strict';

  const Alert = $.fn.alert.Constructor;

})(jQuery);
```

At this point, the constant `Alert` contains a reference to the alert plugin's constructor. This lets you access the plugin's prototype so you can now add a custom method to it:

```
Alert.prototype.zoomIn = function(e) {
  // zoom-in code here
};
```

Now you're all set to add your custom method. Here's Bootstrap `close()` method, which will work as template for `zoomIn()`:

```
close(element) {
  element = element || this._element

  const rootElement = this._getRootElement(element)
  const customEvent = this._triggerCloseEvent(rootElement)
```

```
  if (customEvent.isDefaultPrevented()) {
    return
  }

  this._removeElement(rootElement)
}
```

Below is the code for the `zoomIn()` method:

```
// prevent default action on the link
if (e) e.preventDefault();

// cache the element which has been clicked
const $this = $(this);

// find the parent of the element that has
// been clicked, which is the Alert
const $parent = $this.closest('.alert');

// trigger the custom event on the element
$this.trigger(e = $.Event('zoomIn.bs.alert'));

// grab the alert's current width and font-size
let $parentWidth = parseInt($parent.css('width'));
let $parentFontSize = parseInt($parent.css('font-size'));

// allow devs to run their own custom functionality
if (e.isDefaultPrevented()) return;

// increment alert's width by 20px on each click
// and the font-size by 2px
$parent.css({
  width: $parentWidth += 20,
  fontSize: $parentFontSize += 2
});
```

The code above is self-explanatory. The bit that could be somewhat confusing is `e.isDefaultPrevented()`. I found a great explanation oo this Stack Overflow thread.

The way the code is supposed to work is that each time you click on the **+** link, the alert box and its text content get bigger.

The code for the zoom out functionality is the same as the code above, but instead of increasing the pixel values of `width` and `font-size`, it makes it smaller:

```
$parent.css({
  width: $parentWidth -= 20,
  fontSize: $parentFontSize -= 2
});
```

Adding High Contrast Functionality

The `highContrast()` function follows the same pattern as `zoomIn()` and `zoomOut()`, but instead of using inline CSS styles to dynamically modify the alert's appearance, it adds and removes the `high-contrast-on` class:

```
Alert.prototype.highContrast = function(e) {
  if (e) e.preventDefault();

  const $this = $(this);

  const $parent = $this.closest('.alert');

  $this.trigger(e =
  $.Event('highContrast.bs.alert'));

  if (e.isDefaultPrevented()) return;

  $parent.toggleClass('high-contrast-on');
};
```

Setting Up the Event Listeners

The last piece of the puzzle consists of hooking up the actions to the DOM element with a specified `data` attribute. The Bootstrap `alert.js` file gives some indications as to how you could do this.

In the Data API implementation section, you'll find:

```
$(document).on(
  Event.CLICK_DATA_API,
  Selector.DISMISS,
  Alert._handleDismiss(new Alert())
)
```

The document listens to a click event on any element with a `data-dismiss` attribute and its value. In fact, `Selector.DISMISS` is defined at the top of the file, as well as the EVENT_KEY and DATA_API_KEY:

```
const Selector = {
  DISMISS : '[data-dismiss="alert"]'
}

const DATA_KEY       = 'bs.alert'
const EVENT_KEY      = `.${DATA_KEY}`
const DATA_API_KEY   = '.data-api'
```

Therefore, you can hook up the `zoomIn()`, `zoomOut()`, and `highContrast()` methods to the DOM elements with `data-zoomIn`, `data-zoomOut`, and `data-highContrast` attributes as follows:

```
// hook up zoomIn()
$(document).on('click.bs.alert.data-api', '[data-zoomIn="alert"]',
↪Alert.prototype.zoomIn);

// hook up zoomOut()
```

```
$(document).on('click.bs.alert.data-api', '[data-zoomOut="alert"]',
↪Alert.prototype.zoomOut);

// hook up highContrast()
$(document).on('click.bs.alert.data-api', '[data-highContrast="alert"]',
↪Alert.prototype.highContrast);
```

And that's it!

Live Code

See the Pen Customizing Bootstrap Alert.

Conclusion

As you can see, Bootstrap has made it convenient to add customizations to all its modules, including its JavaScript-powered components.

Chapter

11

8 Tips for Improving Bootstrap Accessibility

Rhiana Heath

A few years ago, I wrote about my experiences on developing a Bootstrap version 3 project to be fully accessible for people with disabilities. This focussed mostly on Bootstrap accessibility in terms of front-end design. (It didn't cover accessibility in terms of screen readers, as that's a whole other story.)

While I could see that the developers behind Bootstrap were making an effort, there were a few areas where this popular UI library fell short. I could also see that there were issues raised on the project that showed they were actively improving — which is fantastic, considering how approximately 3.6% of websites use Bootstrap.

Recently, Bootstrap version 4 was released, so let's take a look and see if any of the issues I had in the past have improved.

What We're Looking For with Design Accessibility

There are a few things to consider when designing a website with accessibility in mind. I believe these improve the user experience for everyone and will likely cover a lot of points you would consider anyway.

Layout

One way to achieve accessibility is by having a clean, easy-to-use layout that looks good on all devices, as well as looking good at a high zoom level. Up to 200% is a good guide.

Bonus points: having front-end code that matches the layout is also good for users who access the Web with a screen reader or by using a keyboard instead of a mouse.

This allows people to use your website easily irrespective of how they're viewing it.

Font

An easy-to-read font is also a big plus for accessibility, as there are some disabilities which make it difficult to read adorned or cursive fonts.

11-1. Cursive fonts can be difficult to read

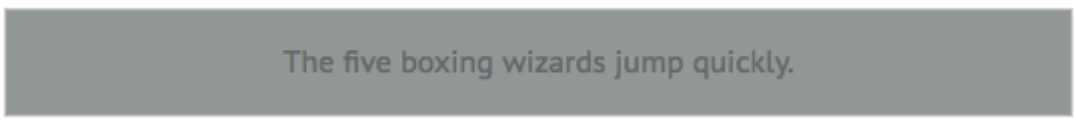
As much as people are against comic sans, for example, it's actually one of the easiest fonts to read if you have dyslexia, which is as high as 15% of the population.

11-2. Seriously in comic sans

Sans-serif fonts like the widely used Roboto and Open Sans are easy to read in general, so picking one from that family is a good choice.

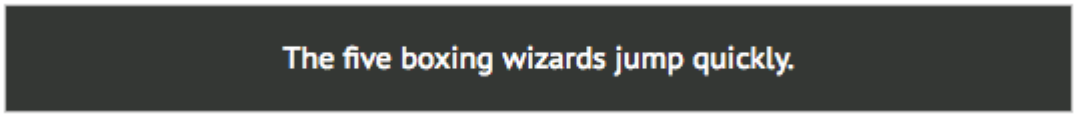
Color

Color choice is also a big consideration. If your colors are too similar contrast-wise, they can make text difficult to read.



11-3. Low contrast text

So it's important to pick colors that stand out. You can test this by using a color contrast checker tool.



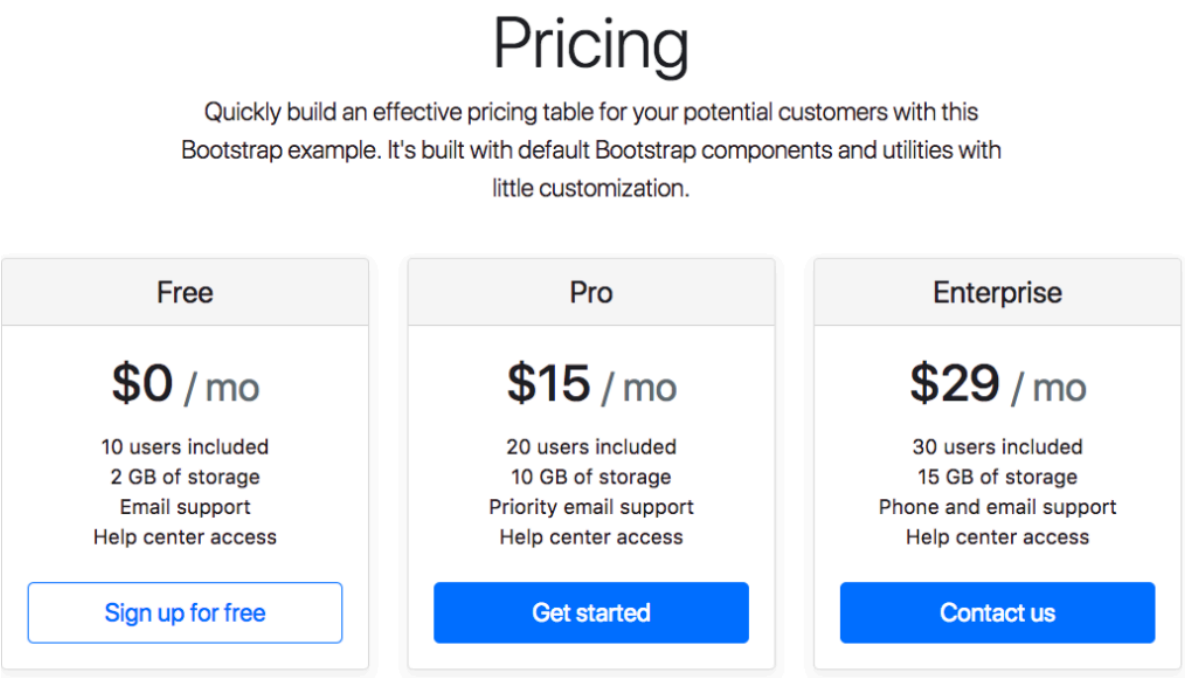
11-4. High contrast text

Let's Start Testing Bootstrap Accessibility

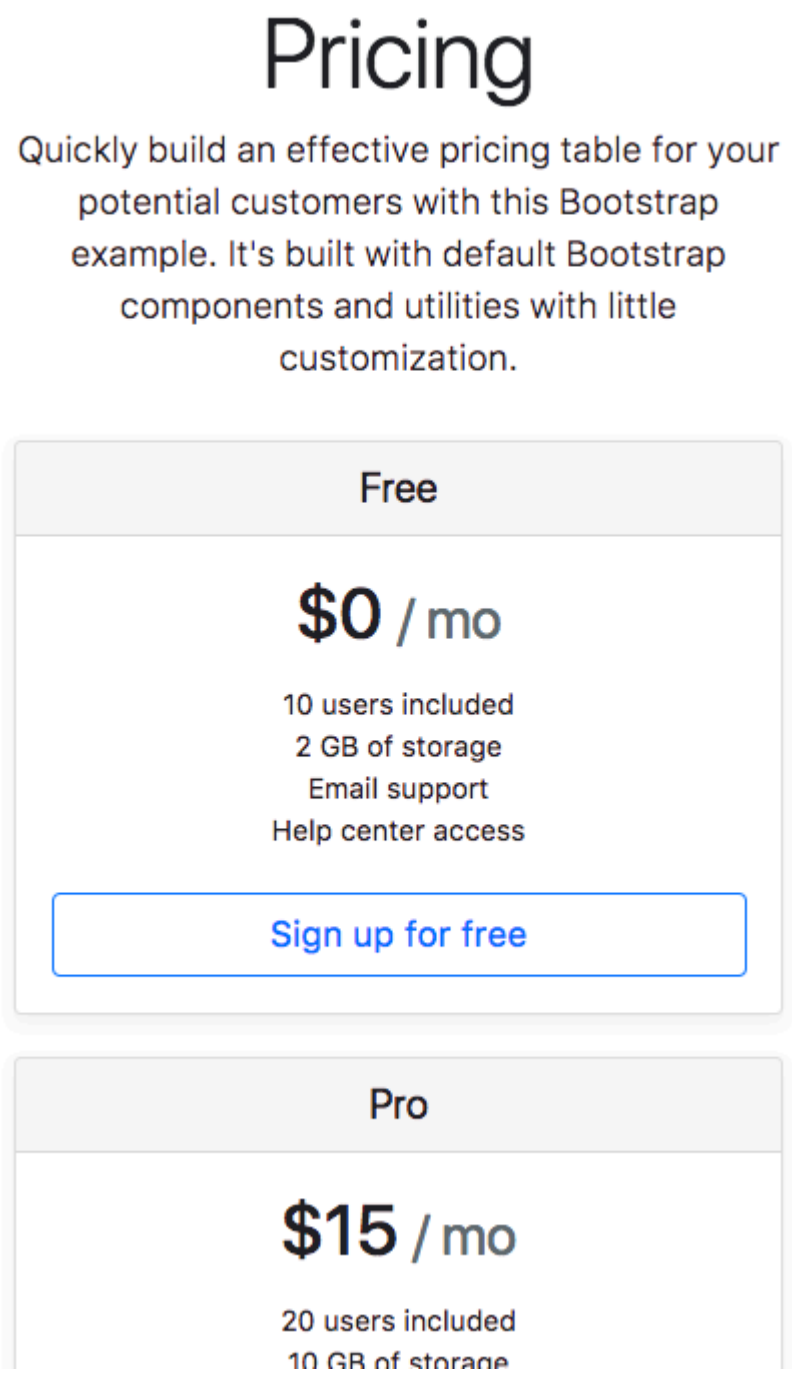
Let's take a look at how Bootstrap stacks up against these criteria.

Layout

While your website might need a custom layout, there are quite a few example layouts that Bootstrap provides, as well as a grid system. Quite often people use Bootstrap just for the responsive grid system, as it's very robust — showing content in columns on desktop and collapsing gracefully onto tablet and mobile views.



11-5. Desktop view

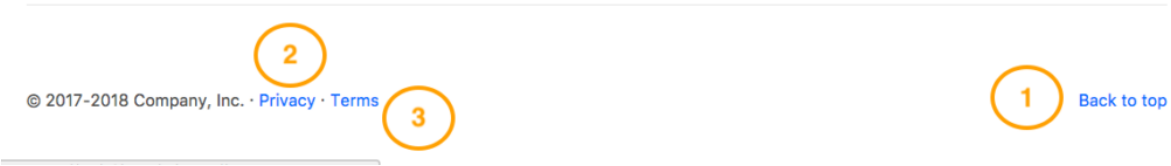


11-6. Mobile View

Bonus points: the layout part where it can get a bit tricky to implement is getting the code to match the layout, so when you're navigating with the keyboard (pressing the tab key) as opposed to the mouse, it follows a logical order and you

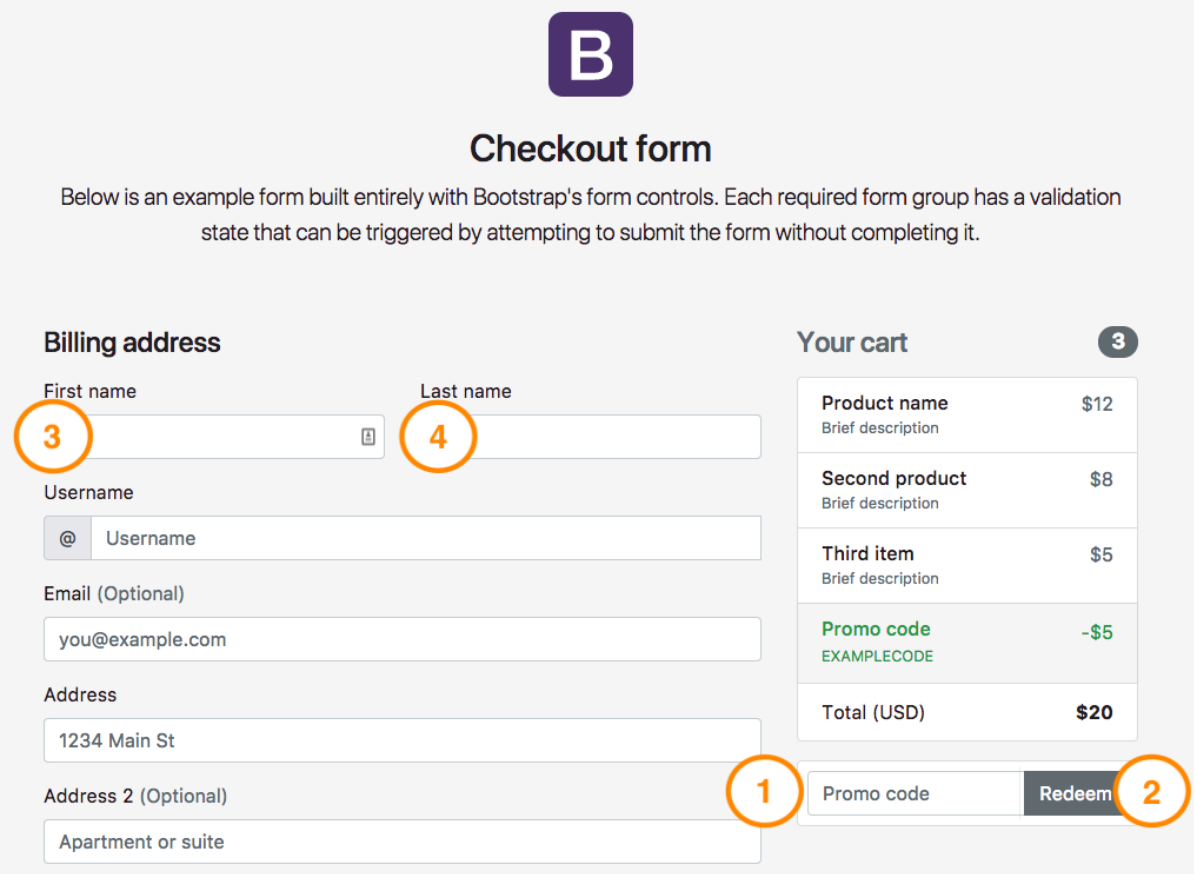
can access all areas.

In the example below, there's a *Back to top* link, but after that there are two more navigation links, so if you select it by keyboard your view goes up to the top, but if you press tab again, you go back down to the bottom links.



11-7. Bootstrap accessibility: Footer link order

Another example is this form where the keyboard focus goes to the sidebar before the main form, which might be a bit confusing.



11-8. Form link order

While these examples are a little annoying, it's an improvement over last time when I tested Bootstrap 3's keyboard focus and it was nearly impossible to use their website with a keyboard. Another thing they need to add in order to improve Bootstrap accessibility is replicating their `hover` mouse styling to match their `focus` keyboard styling. For example:

```
a:hover,
a:focus {
  text-decoration: underline;
}
```

At the moment, when navigating by keyboard, there are no visual cues to show you where you are on the page. Adding this in for focusable elements such as

links, inputs, and buttons would greatly improve the Bootstrap accessibility rating.

Font

By default, Bootstrap comes with Helvetica Neue which is a sans serif font stack.

11-9. Bootstrap accessibility: Helvetica Neue

Sans Serif is a very friendly and easy to read font, so I think this is an accessible choice. They've also set it up so it's compatible with a number of browsers and mobile devices, so it looks consistent no matter what you're using. The font sizes are also set up using `em` units (as opposed to `px`) so will scale well on smaller devices and on zooming.

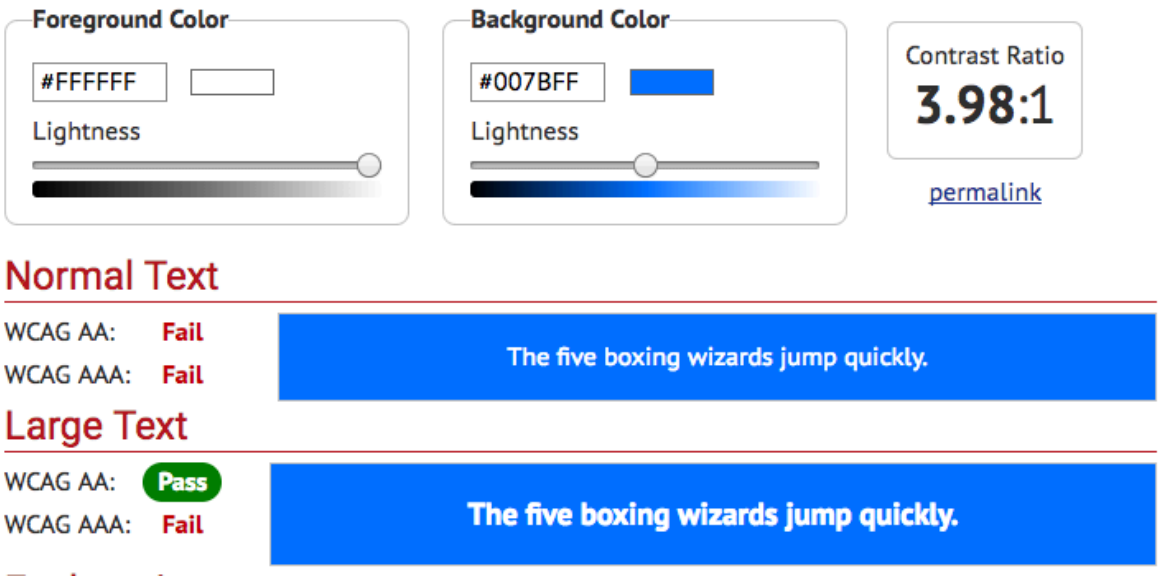
Color

The default colors that Bootstrap comes with are not accessible as per the developers' own Bootstrap accessibility documentation.

> *Most colors that currently make up Bootstrap's default palette — used throughout the framework for things such as button variations, alert variations, form validation indicators — lead to insufficient color contrast.*

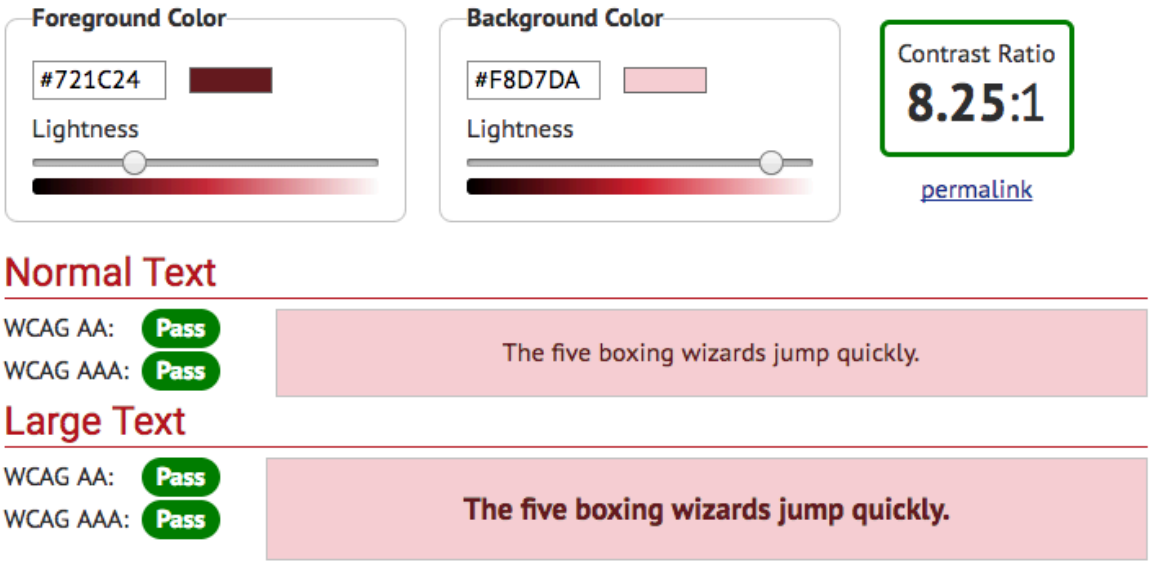
They have gotten better from version 3 to version 4, but it's still important to test any color combination you have on your website to make sure it has a high enough contrast. Webaim has a good color contrast checker, otherwise you can get a browser plugin to help check.

For example, here's the outcome when you test their default blue color. It has a rating of 3.98:1 (up from 3.63:1 from last version). Which is okay if the font size is larger than 18px, but otherwise it's not high enough.



11-10. Default blue color contrast

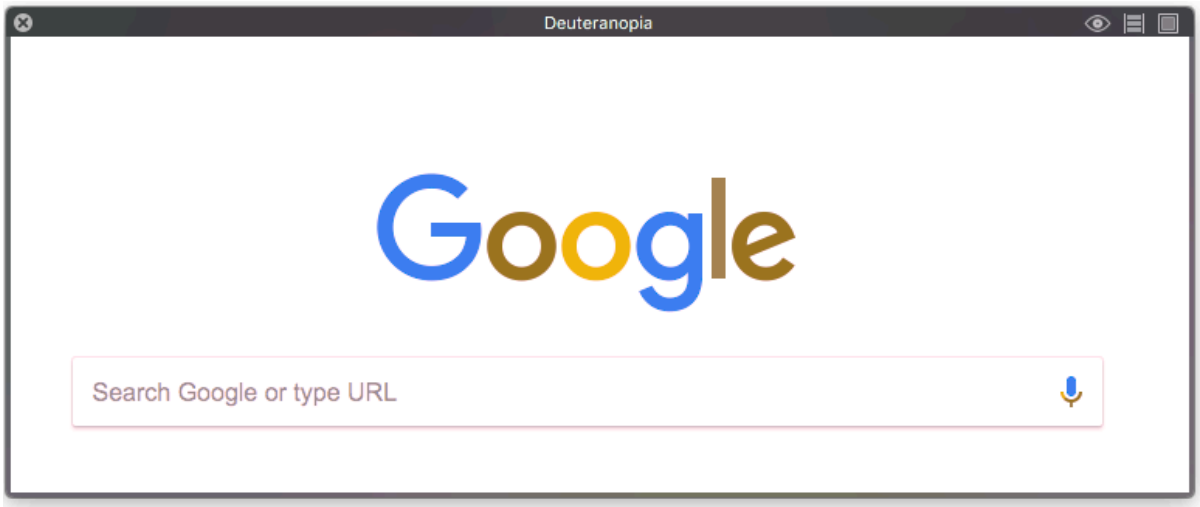
Another example is their alert box coloring. It has a very high rating of 8.25:1, which easily passes (up from 4.53:1 last version, which only just passed).



11-11. Default alert color contrast

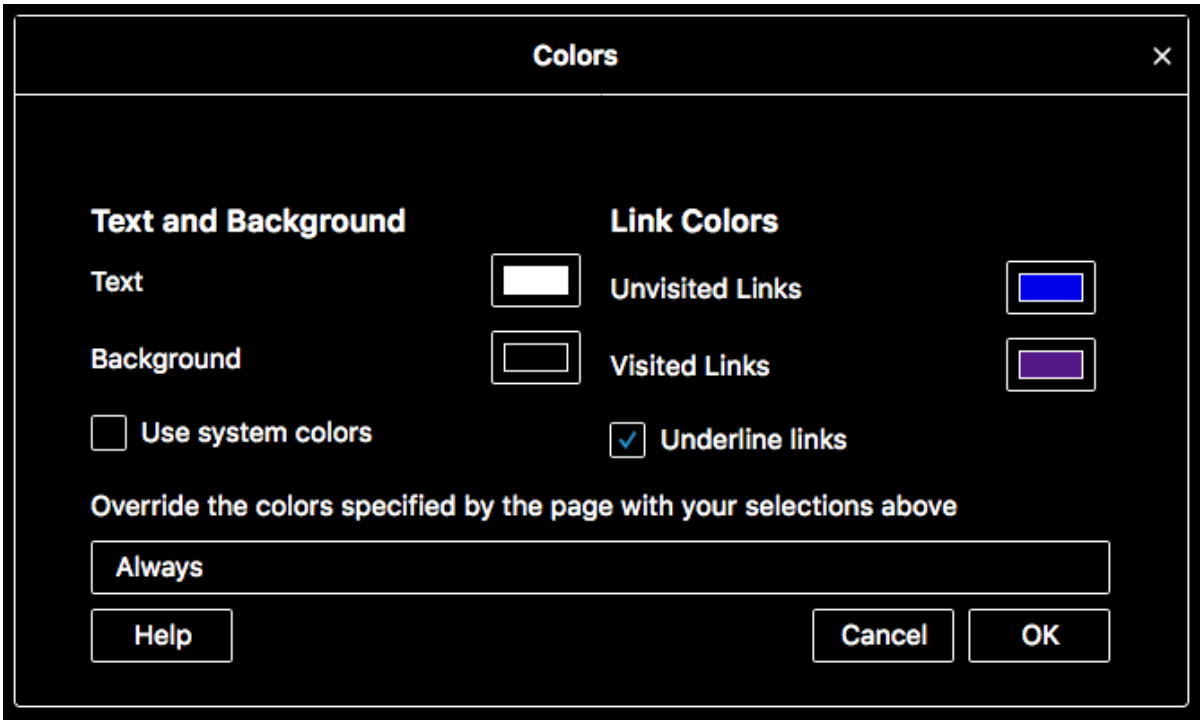
The other alerts had a similar rating — which is great, as they aren't normally customized, unlike the default blue which would be replaced with the website brand color. Just make sure whatever color scheme you do override it with passes the color contrast check.

Bonus points: Running your colors through a color blindness check is also a great thing to do for accessibility. Approximately 8% of the male population has red green color blindness so will have trouble with some color combinations. There are a few tools that show your site with a color blind filter, if you're unsure.



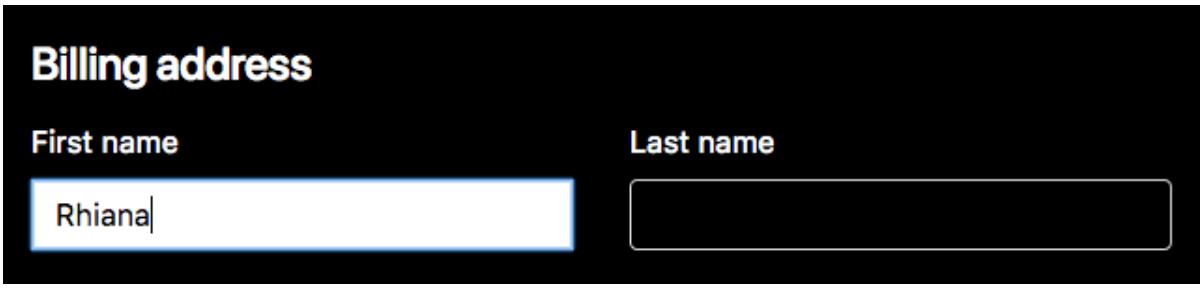
11-12. Using Sim Daltonism

Double extra points: Checking out your website with high contrast color settings is also good for accessibility. Some people with low vision customize their browser settings for high contrast to help them see more clearly. For example, if you head into Firefox's preferences settings, you can set the browser's background and text color:



11-13. Firefox color preferences for high contrast

Last time when I tested Bootstrap version 3 this way, the parts that suffered were the inputs and buttons. The text on both was completely obscured when this setting was turned on, which made the site unusable. This time it faired a little better in some parts.



11-14. High contrast good example

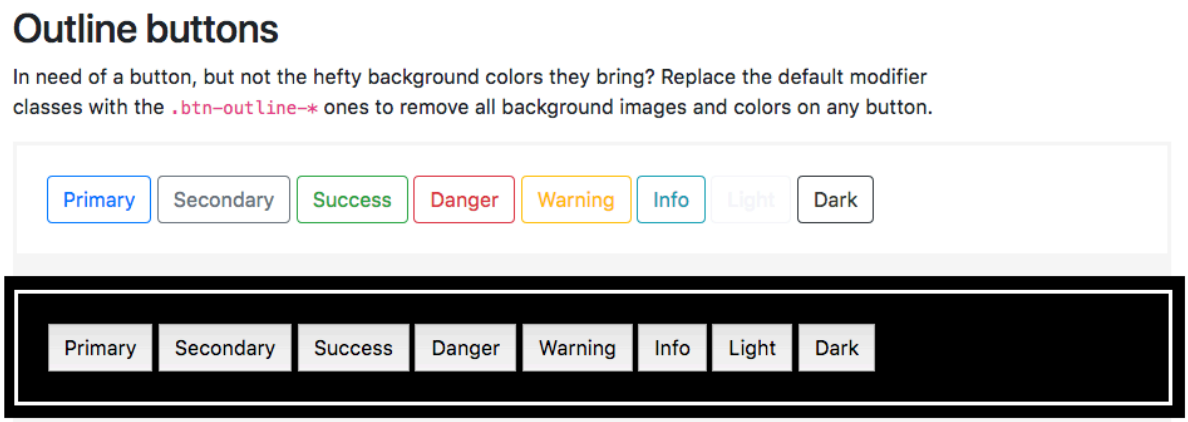
However, in other examples, the background stays black and you can't see the text you type into the input or that on the button:

11-15. High contrast poor example

Thankfully, this is a fairly easy fix: the inputs just need their background color to be transparent rather than white.

```
input {
  background-color: transparent !important;
}
```

The outlined buttons (and link buttons), rather than the block buttons, fair much better as well.



11-16. Outlined buttons high contrast

So Is Bootstrap Accessible?

Out of the box, the short answer is no, not quite. The developers have added in a lot of Bootstrap accessibility features with their documentation, and it's clear they value it, which is a great step. However, that isn't the end of the story.

Designers and developers also need to have accessibility in mind when working on a website. With some care and modification, a framework like Bootstrap can become accessible like any website, but it's up to website developers to take the next steps to ensure their sites can be used by as many people as possible.

Chapter

12

Front-end Frameworks: Custom vs Ready-to-use Solutions

Ivaylo Gerchev

The dilemma over "custom versus ready-to-use" is pretty common in the Web world. Whether we talk about CSS, JavaScript, PHP, or a framework in any other programming language, the question of whether to use a prebuilt codebase versus using your own code arises often. This is especially important for front-end frameworks, because they're directly responsible for the look and feel of a website.

Front-end frameworks like Bootstrap and Foundation have changed the way people build websites. These tools make it a lot easier, even for non-programmers, to create complete websites with minimal effort and time investment. But such "automated" website building can have some serious drawbacks.

So the question remains: is the easier solution the better one?

Although most people tend to choose the tool that gives them the fastest and the easiest results, this, as we'll see later, is not always the best possible option. Also, it seems that many people are more or less confused when they need to choose between custom and ready-to-use variants. That's why, in the next part of this article, we're going to examine the problem in more detail by exploring the advantages and downsides of each choice, which I hope will help you make the right decision when you're faced with this question.

Advantages of Ready-to-use Front-end Frameworks

The reason most people prefer ready-to-use front-end frameworks is that they offer great advantages. Let's look at the most important of them.

- **You don't have to be a programmer**. Decent knowledge of HTML and CSS is enough, in most cases, to build a good-looking and functioning website. All the prior programming has already been done by skilled developers.
- As already mentioned, a framework solution seriously **minimizes the time and**

efforts involved in the website building process.

- The **plug-and-play functionality**. We need only to type some markup to get a fully functional piece of code (a well-cooked HTML, CSS, and JavaScript soup) without worrying about styling, dynamic behavior, and so on.
- When we download a framework's package, we can be sure that we have **stable and well-tested code**. This means that your website will look and behave properly on all browsers that the framework supports.
- We get **regular updates with bug fixes and new features** for the framework.
- Because of the popularity that ready-to-use front-end frameworks have, you can rely on **help from the community** in the form of articles, tutorials (for the framework learning process), and framework add-ons and extensions (for the website building process).

As you can see, there are so many advantages to using a framework. But if you rethink it well, you'll see that most of these refer to the time and efforts you save during the learning and/or building process. These advantages are not strictly related to the quality of the final product. In short, they concern *you* and *your time and efforts*, but not necessarily *your product*.

This is the exact opposite to our users' and clients' interests. They care about how good your product is, and not how much time and effort you've invested to learn the framework and build the website.

So, the easy is good, but …

Downsides of Ready-to-use Front-end Frameworks

Everything comes with a price. Despite the many advantages that a ready-to-use framework offers, there are also some serious drawbacks. For an average user, or for one with modest requirements, the downsides listed below can seem insignificant, but for one looking for the best possible quality, these downsides are of highest importance.

- Although the plug-and-play functionality sounds great, before you take advantage of using it you'll need to **invest time to learn** how to use it.
- Because "ready-to-use" often means "one-for-all", such a framework attempts to solve every single problem for every single situation that a front-end developer could encounter, which leads to **a lot of unnecessary code**.
- As it's made for mass consumption, you can be almost certain that **you'll need to tweak a ready-to-use framework to meet your requirements**, which will take **additional time**.
- If you don't make any customizations **your website will look like all the rest**, which can damage your product's image or at the very least do nothing to improve it.
- In many cases, the framework you've chosen won't have all the components you need, which can lead to an **additional third party integration step** (possibly in the form of bloated jQuery plugins or similar).
- You have **no control over the code**. If the development team decides to remove some component from the framework, you're forced to accept that change. If you don't want it, you have to use the a modified or older version of their product.

For the above reasons, you're unable to achieve some of the most important goals in website building such as uniqueness, high performance, strong compatibility and usability, and a compact and unified codebase and appearance. This is very important, because in web development, especially for mobile, every small improvement can cause great impact. So, "being average" is not a smart decision.

Advantages of Custom Solutions

If you're looking for the best results, then a *custom framework* can provide some strong advantages.

- Once built, a custom solution **will save you time and effort in the future** because it was built precisely to your long-term needs.
- You or your team **will not need to learn how to use or customize it** because

you'll know your framework inside out (although new team members will need to learn it).
- It's **optimized to satisfy only your needs, not everyone's**. This way, it's far easier to achieve high performance.
- You include **only what you need and in the way you need it**. No unnecessary stuff, no bloated code.
- You have **full control over the code and its design implementation**. You can be sure that some really great feature won't be removed or deprecated in future releases unless it aligns with your agenda.
- **Complete modularity**. The flexibility of your framework depends only on you. You can use only those parts of your framework that you need for a particular project.
- **A unified code base**. You can minimize the need for third-party components, which means less mix-and-match work.
- **Uniqueness of your website is 100% guaranteed**. Default themes in most front-end frameworks are pretty much the same, so you still need to create your own. But for your framework you can create unique themes from the very beginning.
- Despite the initial extra work and maintenance, the process will teach you a lot, and thus will improve your skills as a developer.

These advantages may also impact in a positive way your time and efforts, but in this case they're addressed much more to the quality of the final product, thus giving you the ability to achieve the best performance, functionality, and appearance.

Downsides of Custom Solutions

The most significant downsides to a custom-made framework are the time and effort needed when building, testing, maintaining, and improving it. But the downsides can be seen as benefits. Let's see what I mean.

- It takes **more time and effort**. This is true for the early stages, but in the long run this work will save you time and effort.

- **You need to test and maintain the code**. But only for the browsers you're targeting.
- The **bug fixes, updates, and new features are built by you**. This takes time, but you update only what you need, and implement only those features that are required for your projects. Nothing superfluous.

The Third Solution

I don't know to what extent people realize it, but besides the custom and ready-to-use options there is actually a third solution, which I call a *semi-custom* solution. We use a semi-custom solution any time we take a ready-to-use framework such as Bootstrap and perform hard customizations on it.

In fact, a ready-to-use framework is such only if we use it as is, or with minor cosmetic touches (soft customization). But when we do that the net result is a recognizable website with pretty much the same look and feel as many others using the same framework. On the other hand, if you need to customize deeply a particular framework before using it, then we can't say that the framework is truly a ready-to-use solution.

Is "Reinventing the Wheel" a Real Problem?

In forums, blogs, and similar resources, one of the main arguments against custom-made front-end frameworks centers on one question: why reinvent the wheel?

At first glance, such an argument sounds reasonable, but honestly, I can't agree with such thinking. So, let's consider the problem a bit more deeply.

If people hadn't reinvented the wheel over and over again throughout history, today we'd still be using primitive carts. Reinventing the wheel can be a real problem only if you replicate the wheel in the exact same manner — that is, being a copycat. But if you make it more useful, valuable, and better suited to your needs, then there's no problem.

If you take it even more deeply, you'll see that the whole evolution process is based on the "reinventing the wheel" paradigm. Every new phase is a replication of the previous one, but with improvements (sometimes significant, sometimes not).

Let's now consider the word "*develop*". This word means "bring to a more advanced or effective state", "to cause to grow or expand", "to elaborate or expand in detail". In other words, we're not creating something from nothing; we're improving what we already have. So, for a web developer this means writing better and more effective code, and expanding that code in detail — that is, adding additional functionality on top of the existing one.

As you can see, reinventing the wheel is not something you need to avoid. It's a natural process and it occurs very often. So don't be afraid to reinvent the wheel.

One more thing I want to point out is that making your own custom framework has nothing to do with Yet Another Framework Syndrome (YAFS). Building a custom framework *can't* "lead to confusion and frustration", and *you don't need* "to manually evaluate 300 different options in order to select something maintainable." You make the framework for yourself, not for the mass. You know exactly what you need, and implement it in the exact manner you want and consider best practice. So, there's no confusion, no frustration, and no front-end choice paralysis.

Finally, here's a kōan to meditate on: why are people "reinventing the wheel" by having their own children, when they can instead adopt the children of others?! :)

How Can You Make the Right Choice?

To make the right decision, you need to ask the right question. The right question isn't "What's the proper decision?" The right question is "What's the proper decision *for me*?"

The above means that, before you make the right decision, you need a clear

vision about your needs, capabilities, and resources.

As you may have already guessed, I personally prefer a custom solution any time when I'm capable of building it. But this is the proper decision for me (of course, not in every situation). To find out whether it's a proper decision for you too, you need to answer the following questions:

- Am I capable of creating it?
- Do I have enough free/extra time to do it?
- Is it reasonable to make it?

(Note: You may be completely capable of doing it and you may have plenty of free time, but if it doesn't make sense, you should not. If some small adjustments of an existing framework will do the job, then the time and effort to build a completely new product may not be justified.)

If the answers of the above questions are all positive, then the custom solution is definitely the proper decision **for you**.

I hope this article has solved the dilemma "custom versus ready-to-use", and from now on you'll be able to make the right decision easier and with more confidence.