from beginner to pro in 2021

Building AMAZING LAYOUTS (book 2)

Bootstrap 5

layouts in depth

(version 4 also included)

By Ajdin Imsirovic

Bootstrap layouts in depth

Building Amazing Layouts (Book 2)

Ajdin Imsirovic

This book is for sale at http://leanpub.com/bootstrap-layouts-in-depth

This version was published on 2021-04-27

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2020 - 2021 Ajdin Imsirovic

Contents

Chapter 1: Bootstrap 4 containers and contextual colors

We'll begin this chapter by quickly revisiting some basics. After that we'll discuss containers and contextual colors in Bootstrap 4.

The structure of every HTML page

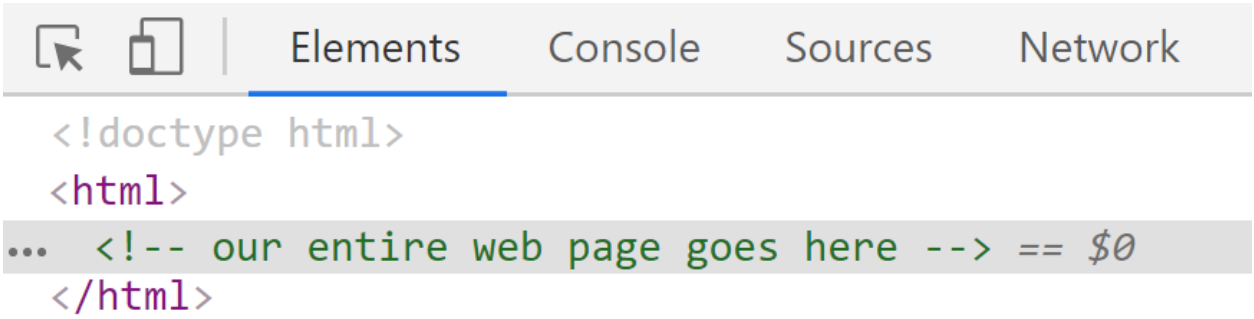
At the very root level of every HTML page, there are just 2 elements:

- the `doctype` declaration
- the `html` tag

In code, it looks like this:

```
<!doctype html>
<html>
    <!-- our entire web page goes here -->
</html>
```

The line that's wrapped inside the `<html>` tag is an HTML comment. An HTML comment is a piece of code that the browser will simply skip when it renders a web page on the screen. However, if you look inside the source code, you'll still see the comment:



An HTML comment in devtools

Let's now replace this comment with some additional tags. These tags are also required for a properly-structured HTML page:

```
<html>
    <head>
        <!-- meta information goes here -->
    </head>
    <body>
        <!-- body information goes here -->
    </body>
</html>
```

The purpose of the `<head>` tag that we see above, is to hold some meta information. What is that meta information? Among other things, the meta information contains links to outside resources - such as the Bootstrap 4 stylesheet.

Adding Bootstrap 4 link meta information

To link to the Bootstrap 4 stylesheet (or to any stylesheet for that matter), we need to use the HTML `<link>` tag.

Every `link` tag has at least a couple of HTML attributes, most notably the `rel` and the `href` attributes:

```
<link rel="stylesheet" href="https://some-web-address-which-has-the-css-file">
```

The `link` HTML tag is an example of a self-closing HTML tag. This means that there's no closing `</link>` tag.

There are a few other HTML attributes that sometimes go into the link tag. We'll see them soon, when we copy-paste the Bootstrap link tag from the official Boostrap docs[1].

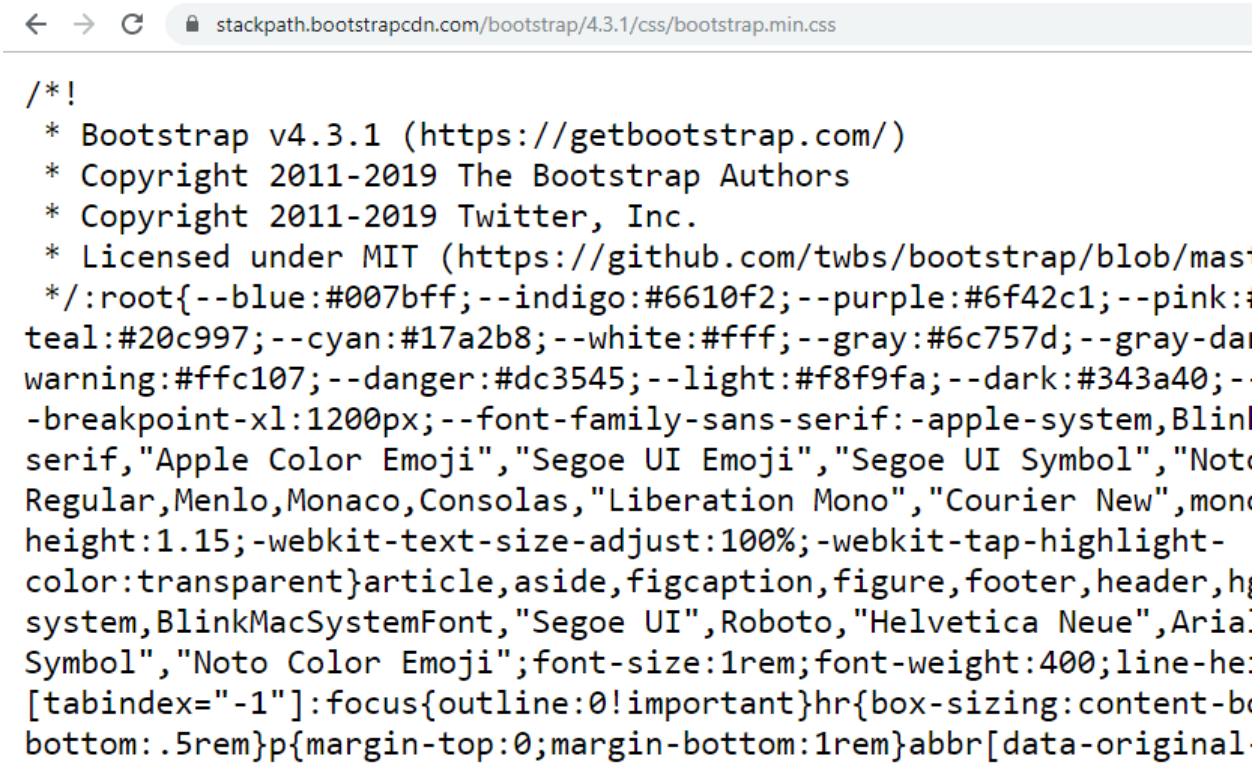
This is the copied code:

```
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">
```

As it can be seen above, besides the `rel` attribute, which describes the type of the resource the `link` element links to, we also see the `href` attribute, which ***points to a minified version of all the Bootstrap framework's styles***.

[1] https://getbootstrap.com/docs/4.3/getting-started/introduction/#css

If we directly open the link in the `href` attribute, we'll see this:



Minified Bootstrap code taht the href attribute of the link tag points to

Alright, so when we open the minified Bootstrap 4 code in the `link tag's` href attribute, it shows in the browser as a bunch of CSS code with spaces removed.

Basically, minified CSS code is just plain old CSS with whitespace removed. This is done so that the CSS file is made smaller, and thus faster to be downloaded from the server to a browser. This improves user experience and keeps server costs down (on websites with lots of traffic).

Here's our updated page:

```
1  <!doctype html>
2  <html>
3      <head>
4          <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.\
5  3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/\
6  iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">
7      </head>
8      <body>
9
10     </body>
11 </html>
```

We could be adding some more meta information inside the `head` tag, but this is it for now.

We'll be adding more complete meta information in the next chapter of this book.

At this point however, let's focus on building the first layout.

We'll start off by just adding some background color.

Adding background color to an HTML element in Bootstrap 4

Let's update our `<body>` tag so that it looks like this:

```
<body class="bg-warning">
    Our layout goes here.
</body>
```

Here's a screenshot of the updated page in the browser:

The result of the .bg-warning CSS class applied on the body element

The CSS class of `bg-warning` is an example of Bootstrap's so-called ***contextual colors***.

There are other colors in Bootstrap, besides the ***warning contextual color***. For example, in the default Bootstrap styles, the red color is the ***danger contextual color***.

Here's the full list:

- primary, e.g `bg-primary`,
- secondary, e.g. `bg-secondary`,
- success, e.g. `bg-success`,
- danger, e.g `bg-danger`,
- warning, e.g `bg-warning`,
- info, e.g `bg-info`,
- light, e.g `bg-light`,
- dark, e.g `bg-dark`

Contextual colors are used throughout Bootstrap. You can find them used in text classes too, or to style borders.

For example, to give a nice red border to a div, we simply do this:

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

Note that above, we're using ***two CSS classes*** to style that border. To understand why this is happening, let's look at the CSS declaration for the `border` class first:

```
.border {
    border: 1px solid #dee2e6!important;
}
```

The hex color of `#dee2e6` is a light shade of gray, so if we just used the `border` class on our div, we'd get a light, gray-colored border around the div.

However, when we additionally use the `border-danger` class on that div, we're adding the following styles:

```
.border-danger {
    border-color: #dc3545!important;
}
```

That's why we need two classes for contextual-colored borders in Bootstrap 4.

If you'd rather like to use, for example, some red-colored text inside a div, you could write code like this:

```
<div class="text-danger">
    ...
</div>
```

The `text-danger` class has the following CSS:

```
.text-danger {
    color: #dc3545!important;
}
```

Ok, so that's it for contextual colors in Bootstrap 4. We've learned about contextual colors, and we've seen how to use them on backgrounds, borders, and text.

Before we wrap up this section, let's just mention a handy CSS class that helps us make our text huge: the `display-*` class. Note that the `*` here is actually a wildcard: it stands for numbers 1 through 4, so the actual classes are:

- `display-1` (the largest huge text)
- `display-2`
- `display-3`
- `display-4` (this is the smallest large text of all four classes)

Now we can update our example layout like this:

```
<!doctype html>
<html>
    <head>
        <link
        rel="stylesheet"
        href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.c\
ss"
        integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT\
2MZw1T"
        crossorigin="anonymous">
    </head>
    <body>
        <div class="bg-warning border border-primary display-1">
            The first div
        </div>
        <div class="bg-info border border-dark display-4">
            The second div
        </div>
    </body>
</html>
```

Two divs with contextual color classes and display text classes

Now that we're aware of how we can easily change background colors, border colors, text colors, and add very large text in Bootstrap, let's learn about another important concept: **containers**.

Containers in Bootstrap 4

In Bootstrap, there's this concept of a container.

It comes in 2 flavors: the first one is `container`, which has a fixed maximum width of 1170 pixels, and the second one is `container-fluid`, which spans the entire available width of the browser window.

By adding the value of `container` to an HTML tag's `class` attribute, its pre-defined CSS code affects how that HTML tag looks on the screen.

Let's update our first layout to see this in action.

```
<!doctype html>
<html>
    <head>
        <link
        rel="stylesheet"
        href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.c\
ss"
        integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT\
2MZw1T"
        crossorigin="anonymous">
    </head>
    <body>
        <div class="container-fluid">
            <div class="bg-warning border border-primary display-1">
                The first div
            </div>
        </div>
        <div class="container">
            <div class="bg-info border border-dark display-4">
                The second div
```

```
            </div>
        </div>
    </body>
</html>
```

Here's the screenshot of the updated layout:

Two divs with contextual color classes and display text classes, part 2

You can see this layout live on codingexercises.com[2].

Completing our first Bootstrap 4 layout

To complete our first Bootstrap 4 layout, all we have to do is add a few more divs with some contextual classes thrown in.

Let's update our first layout to see this in action. Here's the code:

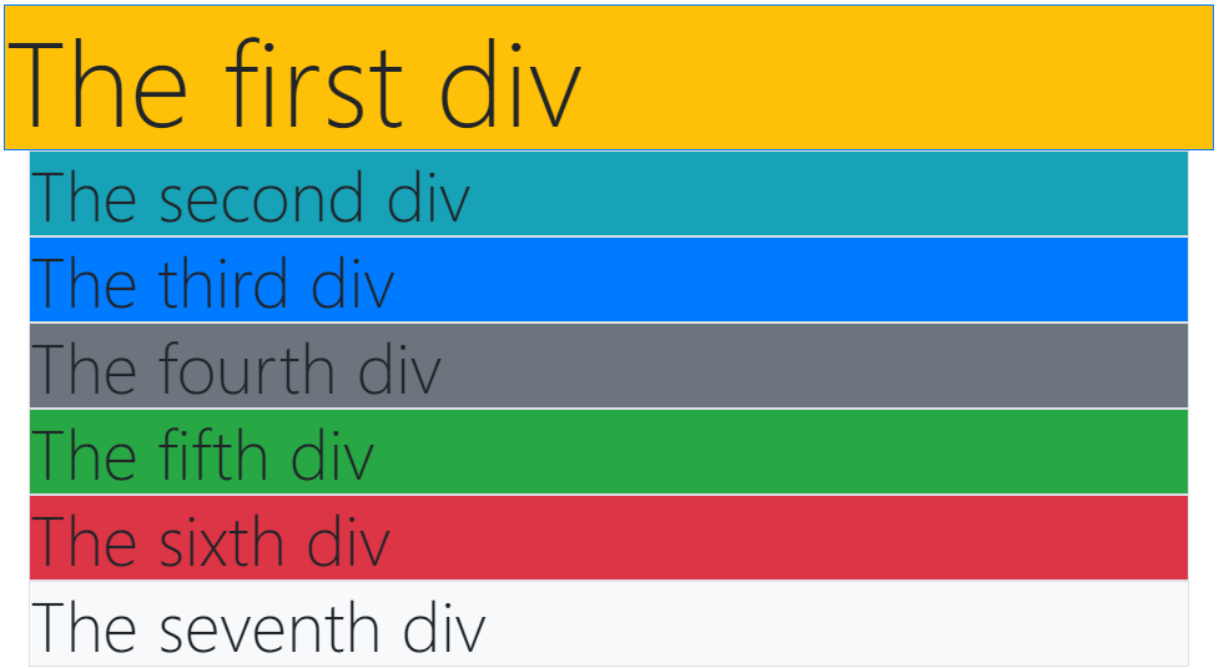
```
<!doctype html>
<html>
    <head>
        <link
        rel="stylesheet"
        href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.c\
ss"
        integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT\
2MZw1T"
        crossorigin="anonymous">
    </head>
    <body>
        <div class="container-fluid">
            <div class="bg-warning border border-primary display-1">
                The first div
            </div>
        </div>
```

[2] https://www.codingexercises.com/codelabs/2019-09-28-building-bootstrap-layouts-part-1/

```
18          <div class="container display-4">
19              <div class="bg-info border">
20                  The second div
21              </div>
22              <div class="bg-primary border">
23                  The third div
24              </div>
25              <div class="bg-secondary border">
26                  The fourth div
27              </div>
28              <div class="bg-success border">
29                  The fifth div
30              </div>
31              <div class="bg-danger border">
32                  The sixth div
33              </div>
34              <div class="bg-light border">
35                  The seventh div
36              </div>
37          </div>
38      </body>
39  </html>
```

To see this update to our layout, visit the layout development's part 2 on codingexercises.com[3]. Here's the screenshot of the updated layout:

[3] https://www.codingexercises.com/codelabs/2019-09-28-building-bootstrap-layouts-part-2/



Using Bootstrap 4 containers and contextual classes on divs

The difference between `container` and `container-fluid`

To really appreciate the difference between `container` and `container-fluid` CSS classes, we need to mention another important feature of Bootstrap: its responsive **breakpoints**.

In the first book of this book series, we covered breakpoints as one of the important foundations of responsive web design.

TL; DR: a breakpoint is a specific resolution at which the layout ***responds to a change in screen width***.

In other words, a breakpoint is defined by the CSS `@media` declaration - also known as **a media query**.

For example, let's say we have a media query that looks like this:

```
@media(min-width: 576px){
    /* ...some CSS code... */
}
```

...then we can say that such a CSS layout has a breakpoint at 576 pixel width.

In Bootstrap, there actually ***is*** a breakpoint of 576 pixels. It's the ***small breakpoint*** ("small" as in "small screen").

All together, there are actually 5 responsive breakpoints in Bootstrap 4[4].

These breakpoints are explained in detail at the above link, but here we can just give a quick overview, and list them out:

1. `xs` breakpoint (no media query, since this breakpoint is the default)
2. `sm` breakpoint for small screens (mobile)
3. `md` breakpoint for tablets
4. `lg` breakpoint for smaller desktops
5. `xl` breakpoint for larger desktops.

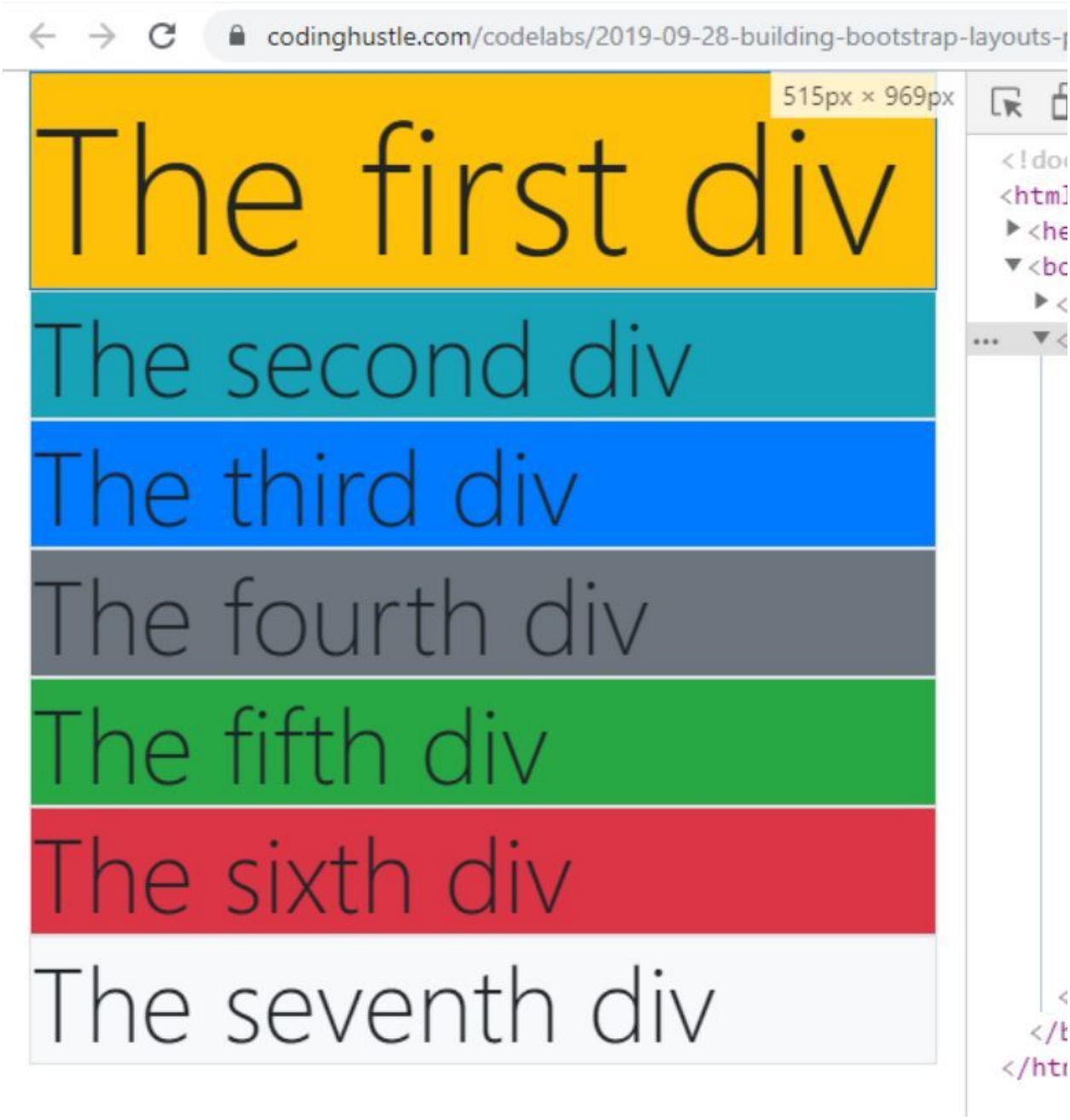
Now that we know about breakpoints in Bootstrap, we can understand the difference between the `container-fluid` and the `container` class in Bootstrap 4, ***through the lens of these breakpoints***.

The `container-fluid` and `container` class at `xs` resolutions

As we've seen above, at screen resolutions with widths under 576 pixels, there's no difference between the `container` and `container-fluid` CSS class.

They both behave exactly the same, and stretch the full width of the screen, as seen in the following screenshot.

[4] https://getbootstrap.com/docs/4.1/layout/overview/#responsive-breakpoints



Container and container fluid at screen widths of under 576 pixels

Note the Chrome developer tools screen measure in pixels in the above screenshot: it reads "515px X 969px". Obviously, at a random width of under 576 pixels - 515 pixels to be precise - the first div and all the other divs have exactly the same widths.

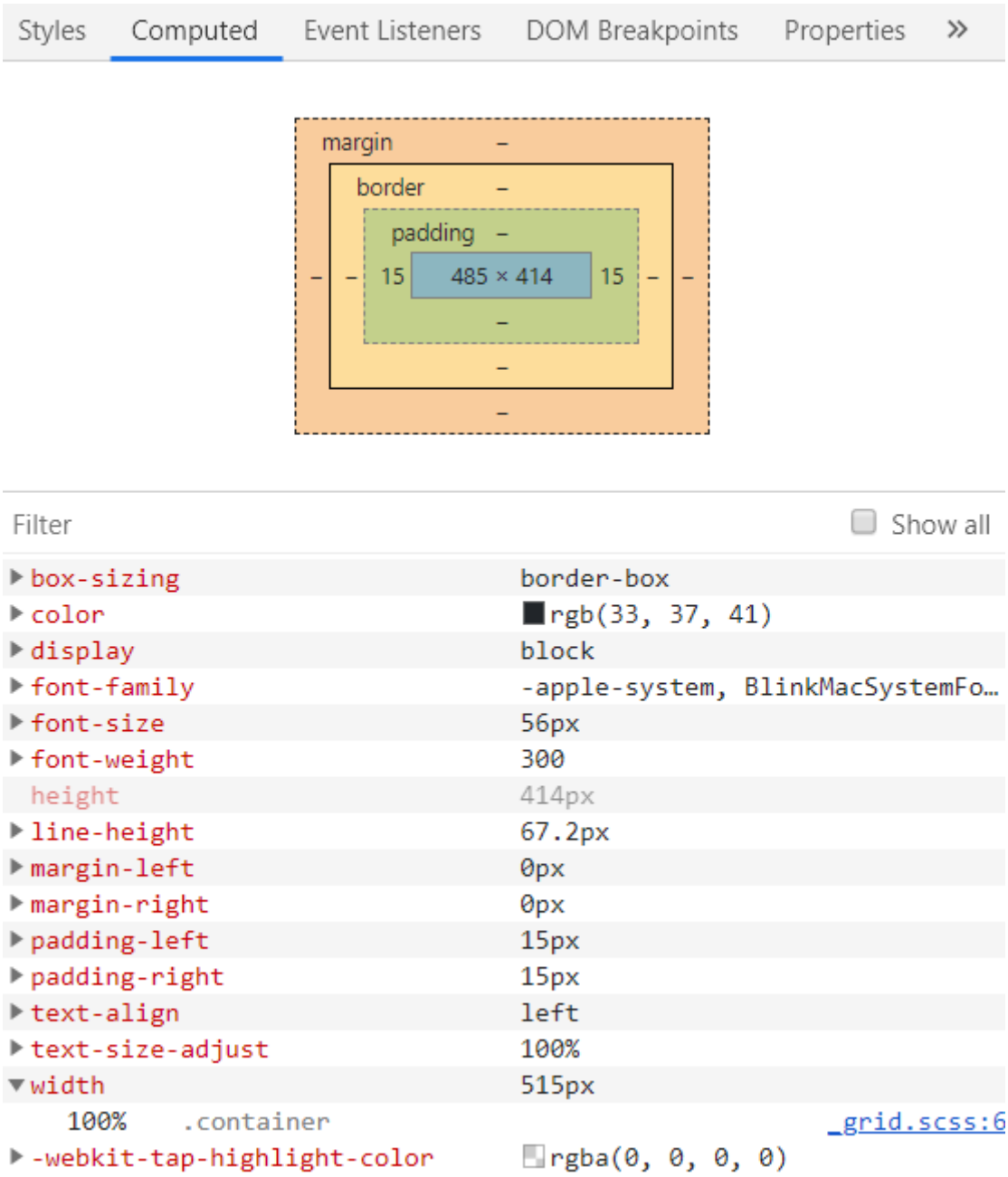
The screenshot is from the example layout we built in this chapter, so we know that the first div has the Bootstrap 4 `container-fluid` class applied to it, while all the others have the `container` class applied - however, at this resolution, they both look exactly the same.

Why is that the case?

In Chrome developer tools, let's look at computed styles for the div with the class of `container-fluid`:

Computed width CSS property for the container-fluid div

Let's now look at computed styles for the div with the class of `container`:



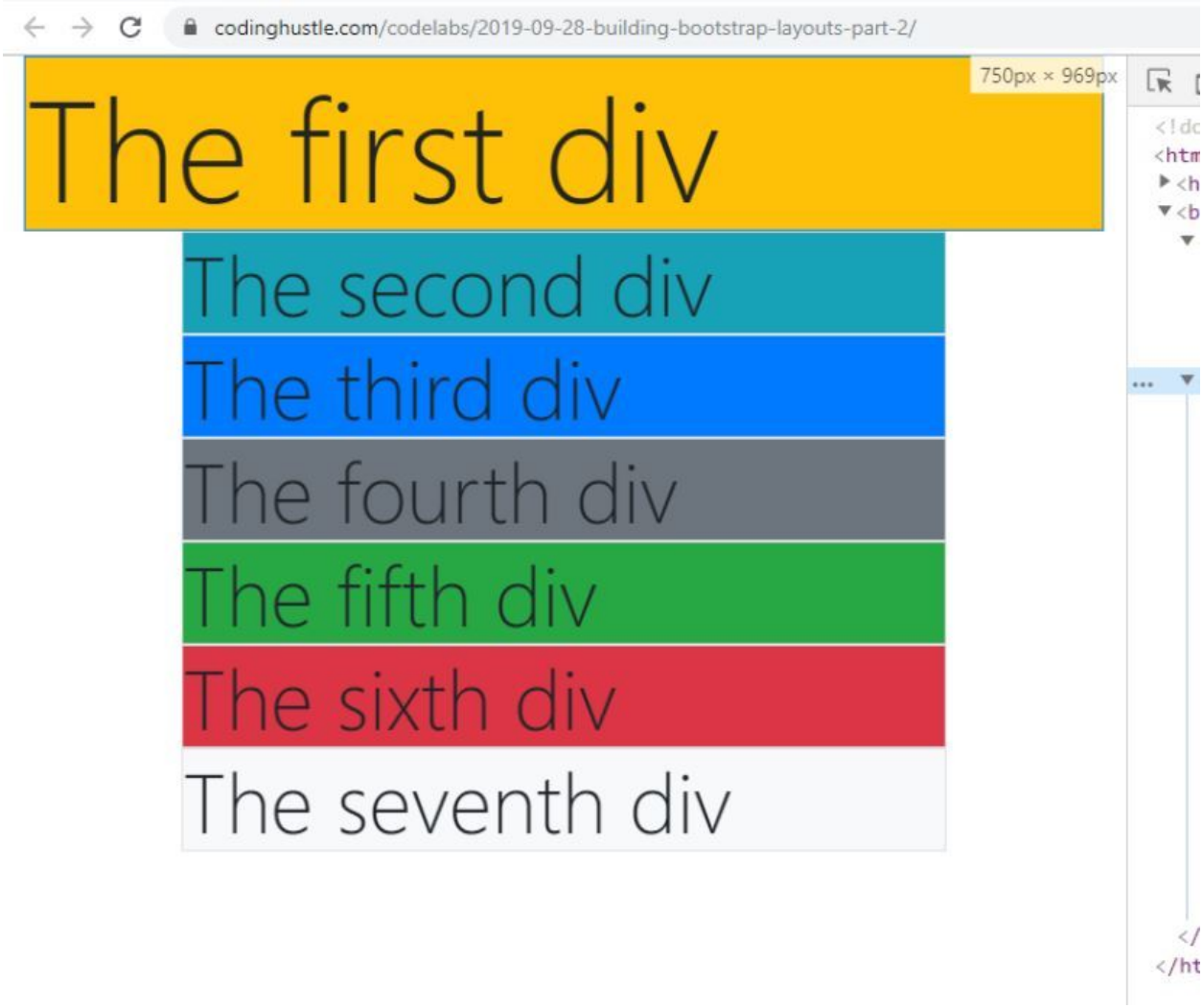
Computed width CSS property for the container div

Sidenote: What are ***computed properties*** in devtools? Computed properties in Chrome developer tools are just the properties that the browser calculates based on all the CSS styles that apply on a certain HTML element. Put differently, computed properties show us the result of browser's CSS specificity calculations on a given element.

The `container-fluid` and `container` class at `sm` resolutions

While we saw that both the `container-fluid` and the `container` CSS classes end up with their width set to 100% of the parent div, meaning their widths are exactly the same at `xs` resolutions, things change as soon as we hit any width larger than 576 pixels.

Let's see a screenshot of such a scenario, at a random 740 pixels, which are still inside the `sm` resolution.



Compare container and container-fluid at 750 pixels width

This screenshot shows us the main distinction between the `container` and `container-fluid` CSS classes: it's how their widths work on all the Bootstrap resolutions, besides the `xs` resolution.

In other words, while the `container-fluid` CSS class will always be 100% wide in Bootstrap 4, the `container` class will have lots of breathing space on its left and right sides on all screen sizes except `xs` - that is, on `sm`, `md`, `lg`, and `xl` resolutions.

Conclusion

In this chapter, we've seen how to get started building layouts in Bootstrap 4, using container classes, contextual color classes, and some textual utility classes.

The layout that we've built might not be anything to write home about, but in my humble opinion, it's the best way to start building some excellent layouts in the Bootstrap 4 framework.

We've also learned exactly why there are two kinds of container classes in Bootstrap.

In the next article in this series, we'll expand on this good foundation further, by building a simple layout using Bootstrap's column grid.

Chapter 2: Bootstrap 4 rows and responsive columns

In this chapter, we'll continue with a detailed introduction to the Bootstrap 4 framework. In the previous chapte, we've built a rudimentary layout using only the `container` and `container-fluid` classes, some contextual color classes, and `display-*` classes.

We'll build this chapter's layout with the `row` and column classes.

The `row` Bootstrap 4 CSS class

Here's a quote straight from the Bootstrap docs: *Rows are wrappers for columns.*

There's no better way to put it: rows are a way to keep columns in check.

We've already covered a brief history of the column grids in the first book in this book series.

If you are unfamiliar with the concept of columns, make sure to read Chapter 13 of that book; besides the history, we all see how to build our own simple column grid.

Bootstrap's column grid is not that much different than the one we saw how to build in the above-referenced article.

Let's see how it works.

Bootstrap 4 column grid

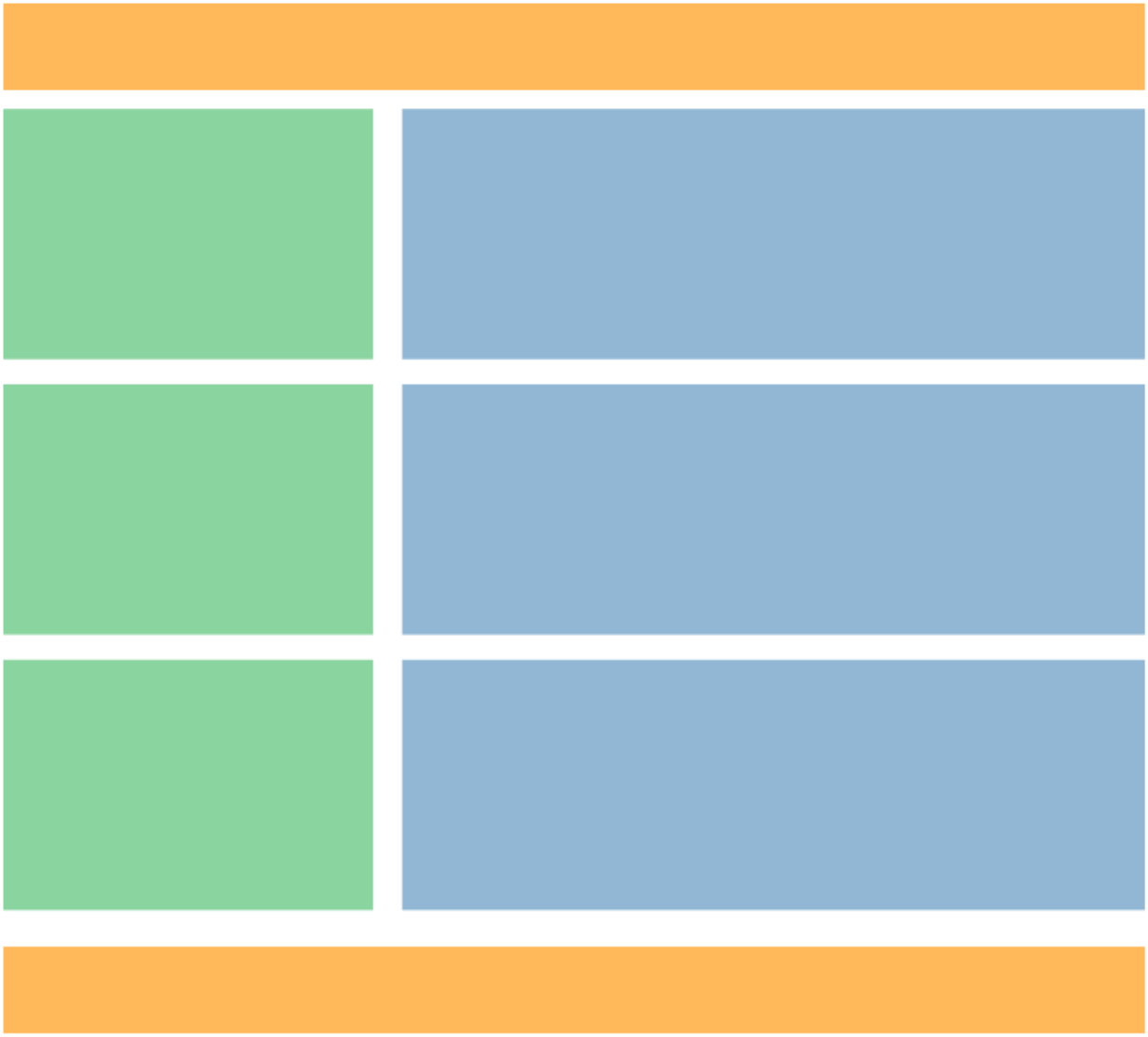
Similar to our own column grid we built in Book 1, in Bootstrap 4, all the columns also have to add up to 12 columns in one row.

For example, we could have three columns in a row, in various combinations:

- `col-4`, `col-4`, `col-4`, or
- `col-2`, `col-2`, `col-8`, or
- `col-3`, `col-6`, `col-3`,
- etc.

As we can see, having 12 columns makes our column grid very flexible. We can easily achieve layouts with halves, quarters, thirds, and so on.

It now time to see this in practice. We'll build a very simple layout, with the following structure:



A quick mockup of what we will build

As we can see in the above diagram, we're going to have a header and a footer.

In between these two, we'll have a repeating pattern, which will have an image spanning 4 columns, and some text spanning 8 columns.

Let's build this layout.

Adding the HTML structure to our layout

Let's begin by adding the container-fluid to the header and footer, and the container class to the middle section of our layout:

```
<header class="container-fluid bg-warning">...</header>
<div class="container">
    <!-- the column grid will be placed here -->
</div>
<footer class="container-fluid bg-warning">...</footer>
```

In the above code, we're using all the Bootstrap 4 CSS classes we've learned about in *Chapter 1, Bootstrap 4 containers and contextual colors*.

Let's now add the middle section, using Bootstrap's column grid.

Adding Bootstrap's column grid

Let's update the code so far by adding the row class to group our columns:

```
<header class="container-fluid bg-warning">...</header>
<div class="container">
    <div class="row">the first image and text goes here</div>
    <div class="row">the second image and text goes here</div>
    <div class="row">the third image and text goes here</div>
</div>
<footer class="container-fluid bg-warning">...</footer>
```

We've just added three wrapping `row` classes to group the divs.

Why are we adding exactly three rows?

Because each `row` is supposed to wrap a total of 12 columns. As we'll see next, these 12 columns can be spread over several HTML elements:

```
<header class="container-fluid bg-warning display-4">
    The header goes here
</header>
<div class="container">
    <div class="row">
        <div class="col-sm-4">
            <img class="img-fluid"
            src="https://images.unsplash.com/photo-1542291026-7eec264c27ff?ixlib=rb-\
1.2.1&auto=format&fit=crop&w=1950&q=80">
        </div>
        <div class="col-sm-8">
            Here goes the text
            for the first image
```

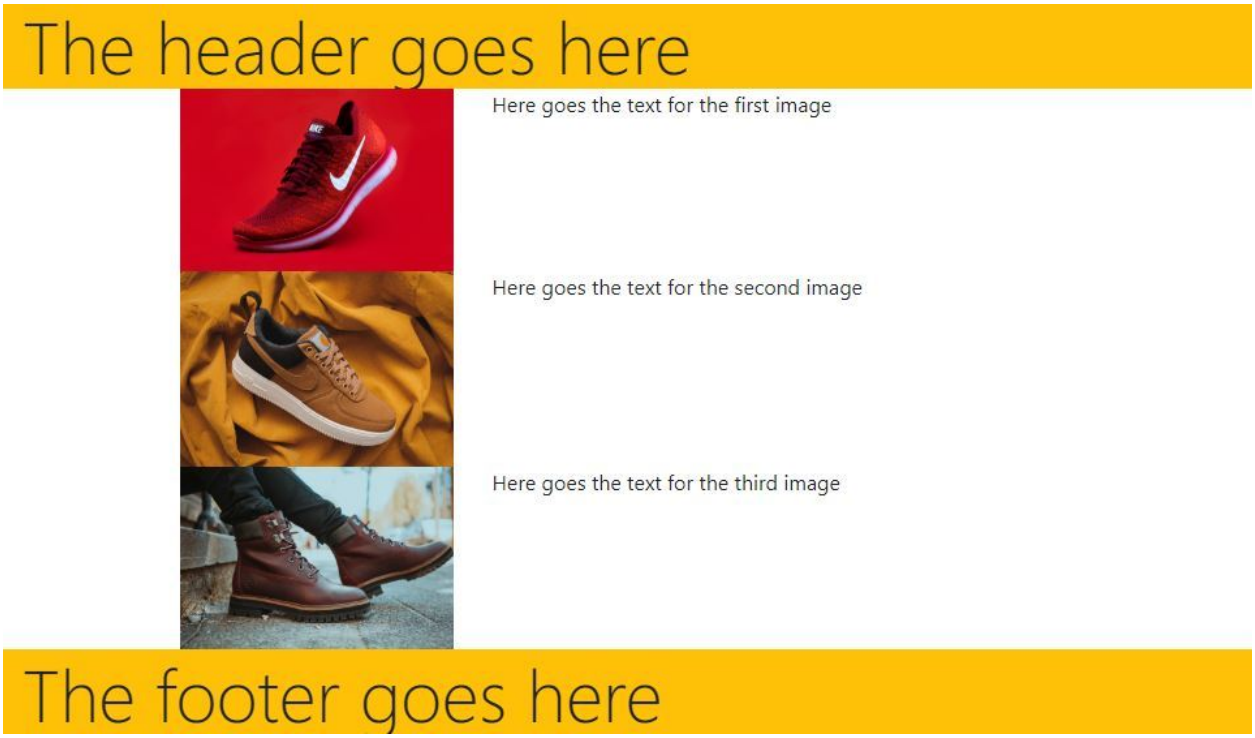
```
        </div>
    </div>
        <div class="row">
        <div class="col-sm-4">
            <img class="img-fluid"
            src="https://images.unsplash.com/photo-1549298916-b41d501d3772?ixlib=rb-\
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=1300&q=80">
        </div>
        <div class="col-sm-8">
            Here goes the text
            for the second image
        </div>
    </div>
        <div class="row">
        <div class="col-sm-4">
            <img class="img-fluid"
            src="https://images.unsplash.com/photo-1542834281-0e5abcbdc5b4?ixlib=rb-\
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=1350&q=80">
        </div>
        <div class="col-sm-8">
            Here goes the text
            for the third image
        </div>
    </div>
</div>
<footer class="container-fluid bg-warning display-4">
    The footer goes here
</footer>
```

We're hotlinking the above images directly from Unsplash.com. Hotlinking is the practice of linking to images directly from another domain, i.e you're not storing images on your server.

With all of these updates made to our code, we can now see the result in this codelab[5].

Let's see a screenshot of our layout at this stage:

[5] https://www.codingexercises.com/codelabs/2019-09-29-building-bootstrap-layouts-article-2-pt-1



This Chapter's layout screenshot

Let's now see this layout with the Bootstrap 4 grid overlay.

This chapter layout screenshot with column grid overlay

We can see above that we're indeed fitting the image into four columns out of the 12 columns in each row.

We can also see that the rest of the space is taken up by the few words that we added to the `col-sm-8` divs.

In the previous chapter, we've learned about some classes to use to give our divs some borders.

We will use them here, next.

Adding borders to our rows

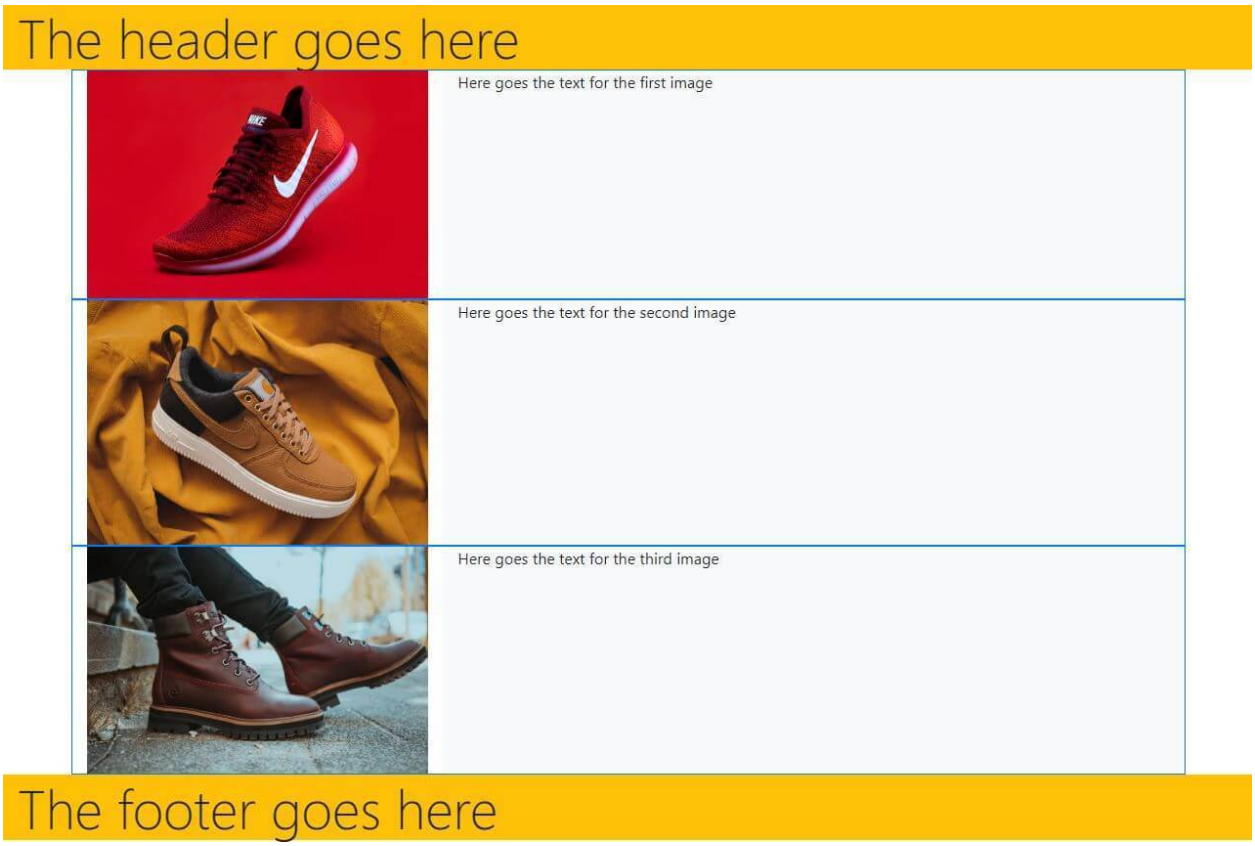
Let's update our code to this:

```
<header class="container-fluid bg-warning display-4">
    The header goes here
</header>
<div class="container">
    <div class="row border border-primary">
        <div class="col-sm-4">
            <img class="img-fluid"
            src="https://images.unsplash.com/photo-1542291026-7eec264c27ff?ixlib=rb-\
1.2.1&auto=format&fit=crop&w=1950&q=80">
        </div>
        <div class="col-sm-8 bg-light">
            Here goes the text
            for the first image
        </div>
    </div>
    <div class="row border border-primary">
        <div class="col-sm-4">
            <img class="img-fluid"
            src="https://images.unsplash.com/photo-1549298916-b41d501d3772?ixlib=rb-\
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=1300&q=80">
        </div>
        <div class="col-sm-8 bg-light">
            Here goes the text
            for the second image
        </div>
    </div>
    <div class="row border border-primary">
        <div class="col-sm-4">
            <img class="img-fluid"
            src="https://images.unsplash.com/photo-1542834281-0e5abcbdc5b4?ixlib=rb-\
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=1350&q=80">
        </div>
        <div class="col-sm-8 bg-light">
            Here goes the text
            for the third image
        </div>
    </div>
</div>
<footer class="container-fluid bg-warning display-4">
    The footer goes here
</footer>
```

We've added borders around each row in the layout above. We've also added the `bg-light`

background to each of the `col-sm-8` divs that contain the "description" for each image.

Now that we've updated our layout, it will look like this:



Applying borders on each row in our layout

The next improvement to our layout could be adding some margins and paddings.

Luckily, Bootstrap has a bunch of so-called *utility* classes, including margin and padding classes.

Margin and padding utility classes in Bootstrap 4

There are 5 default margin and padding classes in Bootstrap 4:

- `m-0` and `p-0`
- `m-1` and `p-1`
- `m-2` and `p-2`
- `m-3` and `p-3`
- `m-4` and `p-4`
- `m-5` and `p-5`

As you might have guessed, both the `m-0` and `p-0` set the margin and padding to zero, respectively.

Then with each rising number, the value of the margin or padding increases, giving us the largest possible default margin and padding in Bootstrap with `m-5` and `p-5` classes, respectively.

We can also add margins and paddings to each individual side (top, right, bottom, left). For example:

- `mt-0` sets the top margin to 0
- `pr-1` sets the right padding's "intensity" to 1
- `mb-2` sets the bottom margin's "intensity" to 2
- etc, all the way up to 5

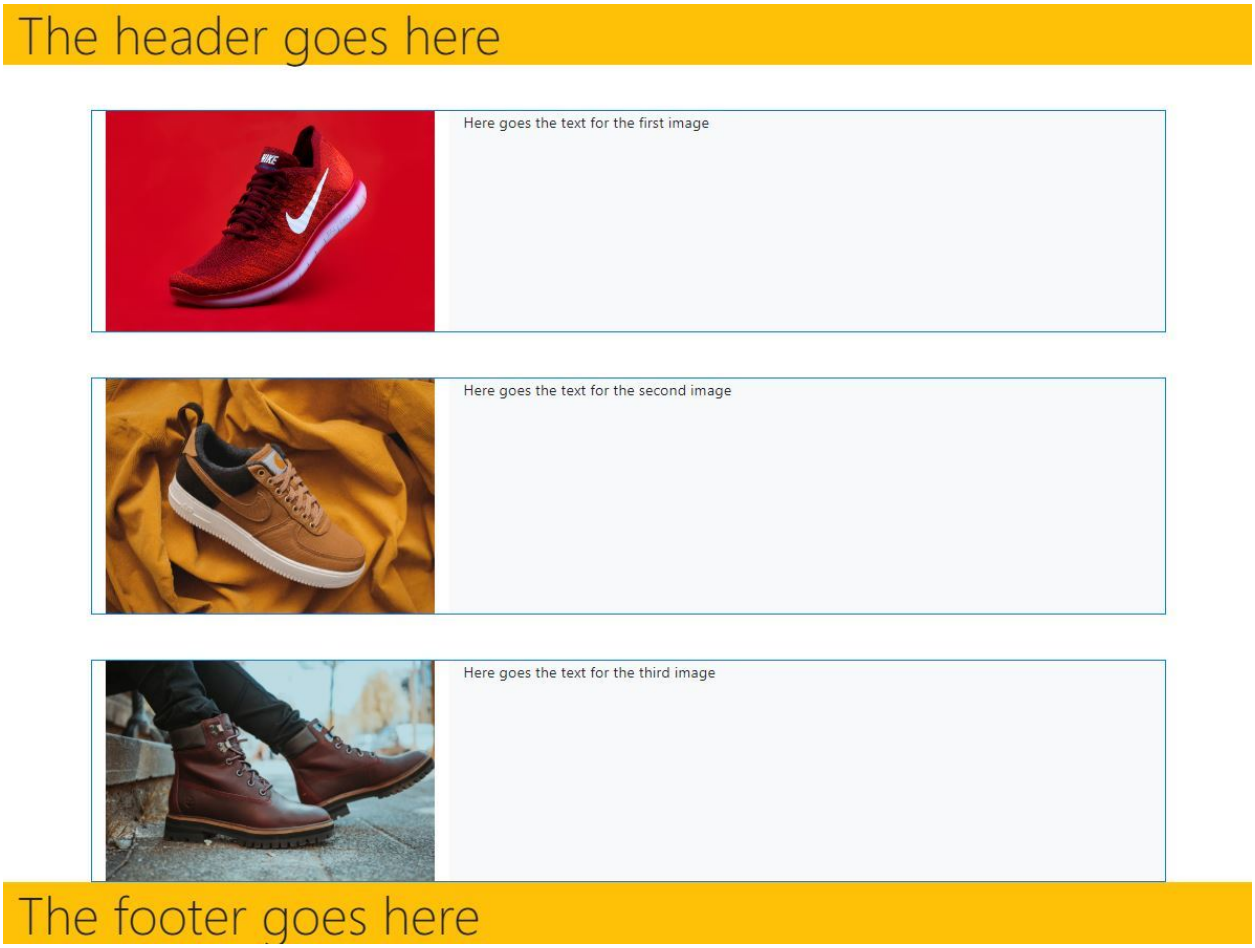
Now that we know how these utility classes work, lets add them to our layout.

Here's the complete code for this chapter's layout, including some margin utility classes:

```
<header class="container-fluid bg-warning display-4">
    The header goes here
</header>
<div class="container">
    <div class="row border border-primary mt-5">
        <div class="col-sm-4">
            <img class="img-fluid"
            src="https://images.unsplash.com/photo-1542291026-7eec264c27ff?ixlib=rb-\
1.2.1&auto=format&fit=crop&w=1950&q=80">
        </div>
        <div class="col-sm-8 bg-light">
            Here goes the text
            for the first image
        </div>
    </div>
        <div class="row border border-primary mt-5">
        <div class="col-sm-4">
            <img class="img-fluid"
            src="https://images.unsplash.com/photo-1549298916-b41d501d3772?ixlib=rb-\
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=1300&q=80">
        </div>
        <div class="col-sm-8 bg-light">
            Here goes the text
            for the second image
        </div>
    </div>
        <div class="row border border-primary mt-5">
        <div class="col-sm-4">
```

```
            <img class="img-fluid"
            src="https://images.unsplash.com/photo-1542834281-0e5abcbdc5b4?ixlib=rb-\
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=1350&q=80">
        </div>
        <div class="col-sm-8 bg-light">
            Here goes the text
            for the third image
        </div>
    </div>
</div>
<footer class="container-fluid bg-warning display-4">
    The footer goes here
</footer>
```

Here is the completed layout's screenshot.



Applying margin-utility classes to our layout

Feel free to check out the live example[6] as well.

Conclusion

In this chapter, we've learned about Bootstrap's column grid and about the `row` class. We've built a fully responsive layout. We've also learned about margin and padding utility classes in Bootstrap.

We've also implemented some stuff we've learned before, namely:

- `container-fluid` and `container` classess
- contextual background classes
- contextual border classes
- typographic `display-*` classes

In the next chapter, we'll bring our layouts to a whole new level by learning about components in Bootstrap.

[6] https://www.codingexercises.com/codelabs/2019-09-29-building-bootstrap-layouts-article-2-pt-3/

Chapter 3: Bootstrap 4 components

In the first two chapters of this book, we've learned about containers, rows, column grids, and utility classes.

Bootstrap also comes with its own CSS reset, called *Reboot*. We've covered CSS resets in the first book in this book series.

We've got enough basics down to really start using all that Bootstrap 4 has to offer.

Introducing components in Bootstrap

Components in Bootstrap are the result of the fast evolution of layouts on the web.

In his book *Don't make me think*, Steve Krug writes about how users come to expect certain norms and patterns in design. For example, you wouldn't expect a steering wheel on a laundry machine, just like you wouldn't expect a lid covering up the top of a dashboard in a car.

The argument that the author makes there is that you shouldn't build layouts that will leave your users head-scratching.

While these examples might seem a bit silly, they point to the fact that as soon as the novelty wears off, website designers generally start using and re-using some patterns in design more often than others, and after enough time has passed, those patterns become the norm.

In web design, the essence of this norm can be seen in Bootstrap components. There are not many of them: the official documentation for Bootstrap version 4.3 lists 24 components altogether, alphabetically.

Let's divide these components in a better way, by splitting them into two broad groups: ***primary*** layout components, and ***secondary*** layout components.

Note that these two terms are just something that I came up with, there's no concept of primary and secondary components in the official docs.

Primary and secondary layout components

Primary layout components include components that fit the following criteria: They represent an entire chunk of a layout.

For example, ***Bootstrap's navbar*** is a primary layout component: it is a full solution for the main navigation on a web layout.

On the other hand, ***Bootstrap's button*** is a secondary layout component: it cannot be used for an entire section of a website layout.

Here is the full list of Bootstrap 4 components that I consider *primary layout components*:

- Breadcrumb
- Card
- Carousel
- Jumbotron
- Navbar
- Pagination

All the other components, in my opinion, can be thought of as *secondary layout components*. These can be further subgrouped into:

- inputs (buttons, button groups, dropdowns, forms, input group)
- notifications (alerts, badges, modals, popovers, progress, spinners, toasts)
- miscellaneous (collapse, list group, media object, navs, scrollspy, spinners)

Dividing our components like this significantly decreases complexity. We no longer have to deal with 24 components without any hierarchy. Instead, we can say that there are 6 primary components; all the others are related to forms (inputs), notifications, and some more specific (miscellaneous) components.

Now that we've listed them, it's time to use these components!

In this chapter, we'll build a layout using only *primary layout components*.

Building a Bootstrap 4 layout with primary layout components only

Let's start by navigating to the Navbar section of Bootstrap's documentation[7].

We'll copy the code found in the very first example, under the Supported content[8] heading.

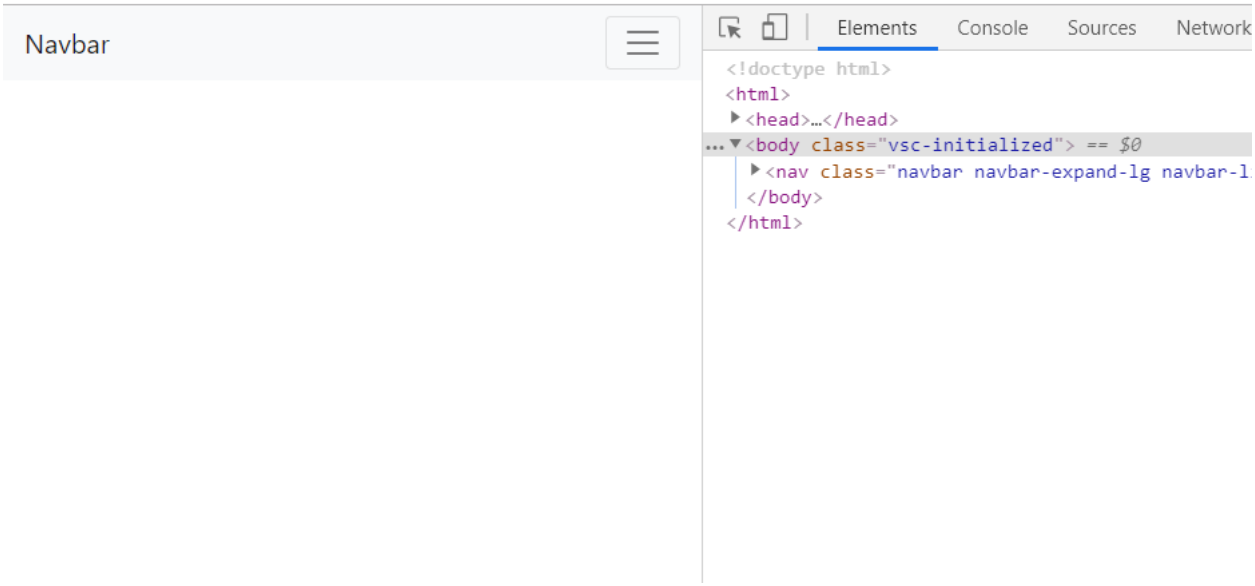
Here is the code:

[7] https://getbootstrap.com/docs/4.3/components/navbar
[8] https://getbootstrap.com/docs/4.3/components/navbar/#supported-content

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <a class="navbar-brand" href="#">Navbar</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>

  <div class="collapse navbar-collapse" id="navbarSupportedContent">
    <ul class="navbar-nav mr-auto">
      <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="#">Link</a>
      </li>
      <li class="nav-item dropdown">
        <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="butto\
n" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
          Dropdown
        </a>
        <div class="dropdown-menu" aria-labelledby="navbarDropdown">
          <a class="dropdown-item" href="#">Action</a>
          <a class="dropdown-item" href="#">Another action</a>
          <div class="dropdown-divider"></div>
          <a class="dropdown-item" href="#">Something else here</a>
        </div>
      </li>
      <li class="nav-item">
        <a class="nav-link disabled" href="#" tabindex="-1" aria-disabled="true">Dis\
abled</a>
      </li>
    </ul>
    <form class="form-inline my-2 my-lg-0">
      <input class="form-control mr-sm-2" type="search" placeholder="Search" aria-la\
bel="Search">
      <button class="btn btn-outline-success my-2 my-sm-0" type="submit">Search</but\
ton>
    </form>
  </div>
</nav>
```

The above code at this point can be found as a live example in this codelab[9].

On narrow resolutions, the navbar even adds a toggle button (sometimes called "the hamburger"), with all the navigation items hidden behind it, to use the narrow space better:



Bootstrap navbar on a narrow resolution

If you click the menu button ("the hamburger"), it should slide down and show all the hidden navigation items.

However, if you did it now, nothing would happen.

Why is that?

It's because Bootstrap 4 uses the jQuery JavaScript library to make interactivity work.

We've just seen an example of what happens when there's no jQuery added to our Bootstrap-based layout: the navbar toggle button doesn't toggle.

The fix is easy enough: we just need to add a `<script>` tag that points to a link to Bootstrap's jQuery.

Adding jQuery to Bootstrap 4 layout

In any website layout, the best place to add a `<script>` tag is usually just above the closing `</body>` tag.

So let's hop on over to Bootstrap's official docs[10] and copy the code for adding jQuery:

[9] https://www.codingexercises.com/codelabs/2019-09-30-building-bootstrap-layouts-article-3-pt-1/
[10] https://getbootstrap.com/docs/4.3/getting-started/introduction/#js

```
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
    integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6ji\
zo"
    crossorigin="anonymous"></script>
<script
    src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js" \
integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"
    crossorigin="anonymous"></script>
<script
    src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"
    integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07j\
RM"
    crossorigin="anonymous"></script>
```

As we can see above, we are actually adding three different JavaScript files. The order is important: first we add the jQuery library, the file name being `jquery-3.3.1.slim.min.js`, then we add `popper.min.js` (which helps with the tooltip component), and we add `bootstrap.min.js`. Since Bootstrap relies on jQuery, it's important that *jQuery gets imported first.*

Our entire website layout for this article now looks like this:

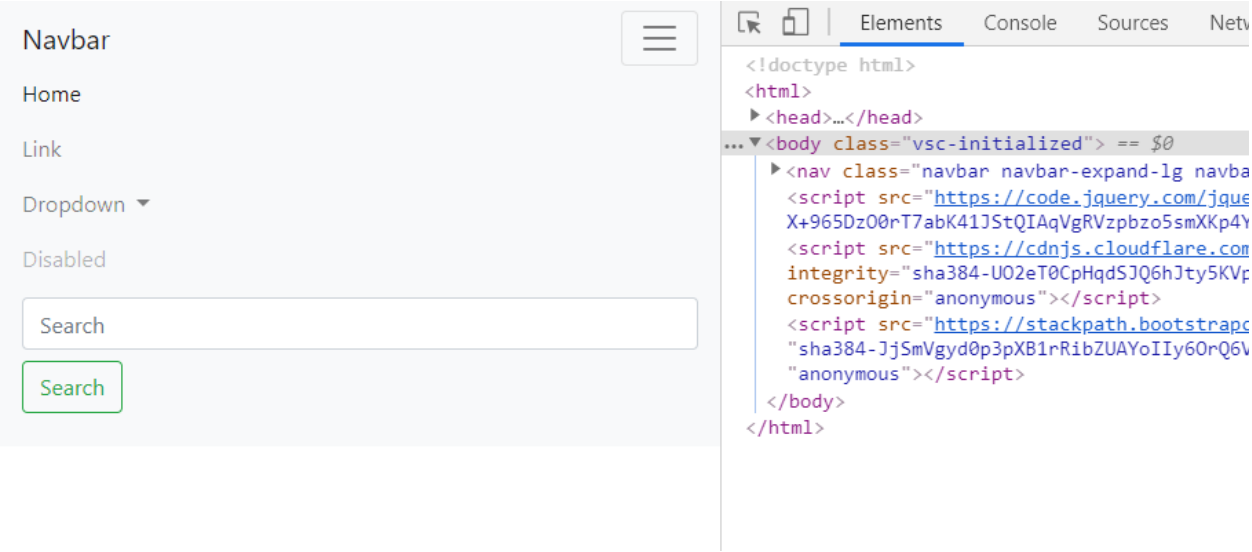
```
<!doctype html>
<html>

<head>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/\
css/bootstrap.min.css"
        integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT\
2MZw1T" crossorigin="anonymous">
</head>

<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <a class="navbar-brand" href="#">Navbar</a>
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-tar\
get="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="\
false" aria-label="Toggle navigation">
          <span class="navbar-toggler-icon"></span>
        </button>

        <div class="collapse navbar-collapse" id="navbarSupportedContent">
          <ul class="navbar-nav mr-auto">
            <li class="nav-item active">
```

```
23              <a class="nav-link" href="#">Home <span class="sr-only">(current)</spa\
24 n></a>
25             </li>
26             <li class="nav-item">
27               <a class="nav-link" href="#">Link</a>
28             </li>
29             <li class="nav-item dropdown">
30               <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role=\
31 "button" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
32                 Dropdown
33               </a>
34               <div class="dropdown-menu" aria-labelledby="navbarDropdown">
35                 <a class="dropdown-item" href="#">Action</a>
36                 <a class="dropdown-item" href="#">Another action</a>
37                 <div class="dropdown-divider"></div>
38                 <a class="dropdown-item" href="#">Something else here</a>
39               </div>
40             </li>
41             <li class="nav-item">
42               <a class="nav-link disabled" href="#" tabindex="-1" aria-disabled="tru\
43 e">Disabled</a>
44             </li>
45           </ul>
46           <form class="form-inline my-2 my-lg-0">
47             <input class="form-control mr-sm-2" type="search" placeholder="Search" a\
48 ria-label="Search">
49             <button class="btn btn-outline-success my-2 my-sm-0" type="submit">Searc\
50 h</button>
51           </form>
52         </div>
53       </nav>
54
55       <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
56       integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6\
57 jizo"
58       crossorigin="anonymous"></script>
59   <script
60       src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js\
61 " integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1\
62 "
63       crossorigin="anonymous"></script>
64   <script
65       src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"
```

```
66        integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B0\
67  7jRM"
68        crossorigin="anonymous"></script>
69  </body>
70
71  </html>
```

With these updates in place, let's see the example layout in action[11].

As expected, the navbar now works, and the button can be toggled, as seen below:



Bootstrap navbar on a narrow resolution with the toggle button turned on

Great! We've successfully added the navbar component to our layout.

Next, let's add some breadcrumbs right under the navbar.

Adding the breadcrumb component

Let's re-visit Bootstrap docs and copy an example of the breadcrumb component[12]:

[11] https://www.codingexercises.com/codelabs/2019-09-30-building-bootstrap-layouts-article-3-pt-2/
[12] https://getbootstrap.com/docs/4.3/components/breadcrumb/

```
<nav aria-label="breadcrumb">
  <ol class="breadcrumb">
    <li class="breadcrumb-item"><a href="#">Home</a></li>
    <li class="breadcrumb-item"><a href="#">Library</a></li>
    <li class="breadcrumb-item active" aria-current="page">Data</li>
  </ol>
</nav>
```

Now we'll just paste it in right under the navbar's closing `</nav>` tag.

The resulting layout[13] is actually starting to look like a real webpage!

Next, let's add a carousel.

Adding the carousel component

Carousel is just a slideshow component, with items sliding from right to left, by default.

Let's copy-paste an example[14] to see this slideshow in action.

```
<div id="carouselExampleIndicators" class="carousel slide" data-ride="carousel">
  <ol class="carousel-indicators">
    <li data-target="#carouselExampleIndicators" data-slide-to="0" class="active"></\
li>
    <li data-target="#carouselExampleIndicators" data-slide-to="1"></li>
    <li data-target="#carouselExampleIndicators" data-slide-to="2"></li>
  </ol>
  <div class="carousel-inner">
    <div class="carousel-item active">
      <img src="..." class="d-block w-100" alt="...">
    </div>
    <div class="carousel-item">
      <img src="..." class="d-block w-100" alt="...">
    </div>
    <div class="carousel-item">
      <img src="..." class="d-block w-100" alt="...">
    </div>
  </div>
  <a class="carousel-control-prev" href="#carouselExampleIndicators" role="button" d\
ata-slide="prev">
    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
```

[13] https://www.codingexercises.com/codelabs/2019-09-30-building-bootstrap-layouts-article-3-pt-3/
[14] https://getbootstrap.com/docs/4.3/components/carousel/#with-indicators

```
22      <span class="sr-only">Previous</span>
23    </a>
24    <a class="carousel-control-next" href="#carouselExampleIndicators" role="button" d\
25  ata-slide="next">
26      <span class="carousel-control-next-icon" aria-hidden="true"></span>
27      <span class="sr-only">Next</span>
28    </a>
29  </div>
```

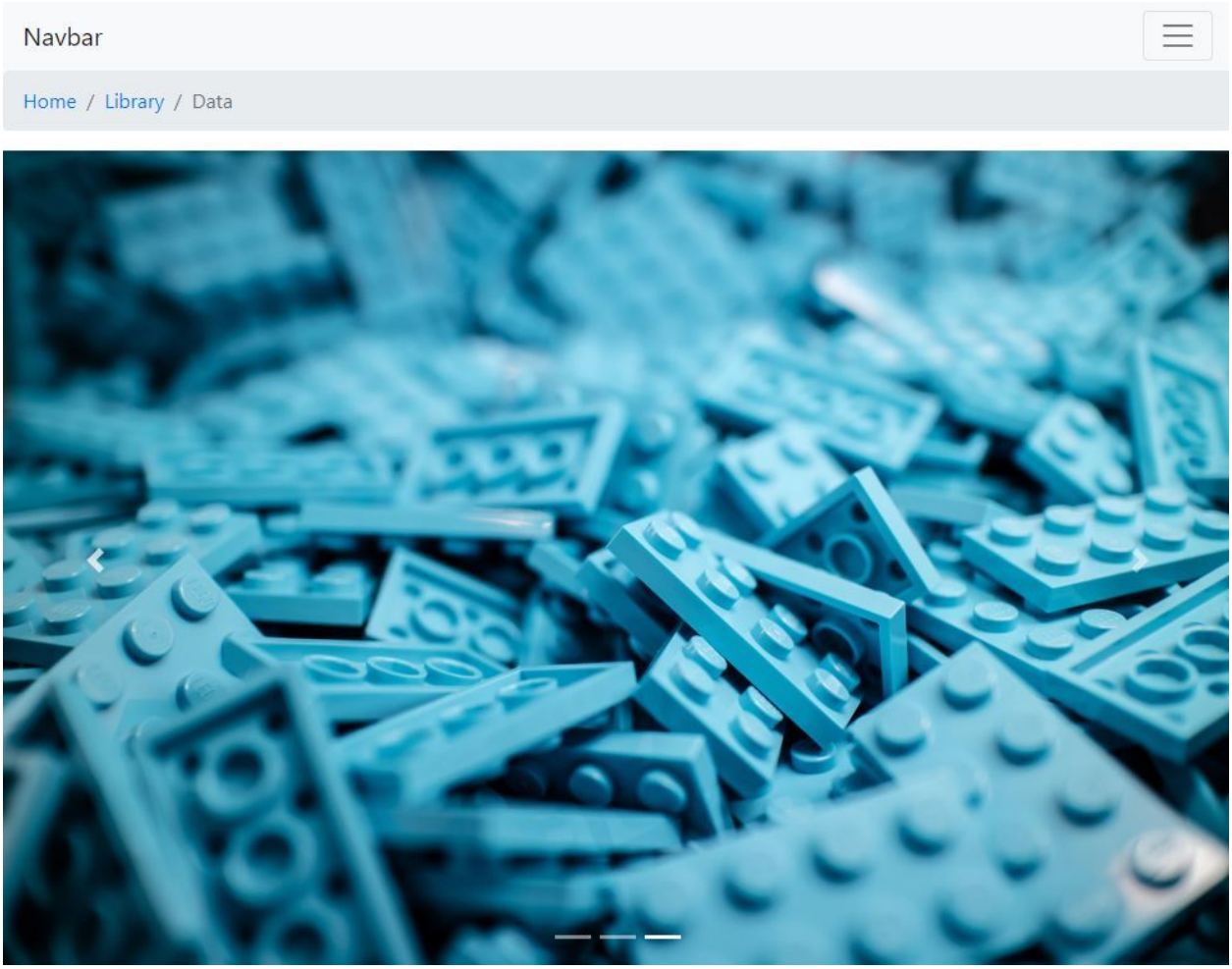
The live example of our updated layout now looks like this[15].

The only difference between the above code and the one in our live example is that in the live example I've actually added some images from unsplash.com, namely:

- yellow legos
- magenta legos
- blue legos

Here's our layout with carousel added:

[15] https://www.codingexercises.com/codelabs/2019-09-30-building-bootstrap-layouts-article-3-pt-4/



Our layout with carousel added

We've added three different *primary components* so far.

What we've got left is to add **the card**, **the jumbotron**, and **the pagination** components.

Adding the card component

The card component is very, very versatile.

Cards can be used for numerous web layout functionalities, such as, for example:

- biography sections on blogs
- product showcase on e-commerce sites
- pricing sections
- testimonial sections

The card component is ubiquitous.

In this article's layout, we'll add three card components in a single row.

Before we add the cards, let's plan it out a bit: we'll need a div with the class of `container-fluid`, and we'll add it a nice background, for example `bg-info`. Inside this div, we'll add another div, which will have the class of `container`. One more level deep, we'll have a row, and inside the row we'll nest three divs with `col-sm-4` classes (so that each card takes up a third of the width on `sm` and greater resolutions).

Here's the code we'll start with:

```
<div class="container-fluid bg-info">
  <div class="container">
    <div class="row">
      <div class="col-sm-4">card 1 here</div>
      <div class="col-sm-4">card 2 here</div>
      <div class="col-sm-4">card 3 here</div>
    </div>
  </div>
</div>
```

We'll replace the "card X here" with the following example from Bootstrap docs[16]:

```
<div class="card" style="width: 18rem;">
  <img src="..." class="card-img-top" alt="...">
  <div class="card-body">
    <h5 class="card-title">
      Card title
    </h5>
    <p class="card-text">
      Some quick example text to build on the card
      title and make up the bulk of the card's content.
    </p>
    <a href="#" class="btn btn-primary">
      Go somewhere
    </a>
  </div>
</div>
```

Again, we're adding some images of lego blocks inside the `src` attribute of the above card.

The updated layout can be found here[17].

[16] https://getbootstrap.com/docs/4.3/components/card/#titles-text-and-links
[17] https://www.codingexercises.com/codelabs/2019-09-30-building-bootstrap-layouts-article-3-pt-5/

To make the images look at least somewhat different, we're also adding an inline style of filter: `grayscale(1)` to each of them, a trick we learned in Chapter 24 of the first book in this book series.

Next, we'll be adding the jumbotron[18], an easy way to style a simple signup form quickly.

Adding the jumbotron component

We're copying the code from the second jumbotron example:

```
<div class="jumbotron jumbotron-fluid">
  <div class="container">
    <h1 class="display-4">Fluid jumbotron</h1>
    <p class="lead">This is a modified jumbotron that occupies the entire horizontal\
 space of its parent.</p>
  </div>
</div>
```

We also need to update it with a simple input and a submit button, copied from the forms section[19]:

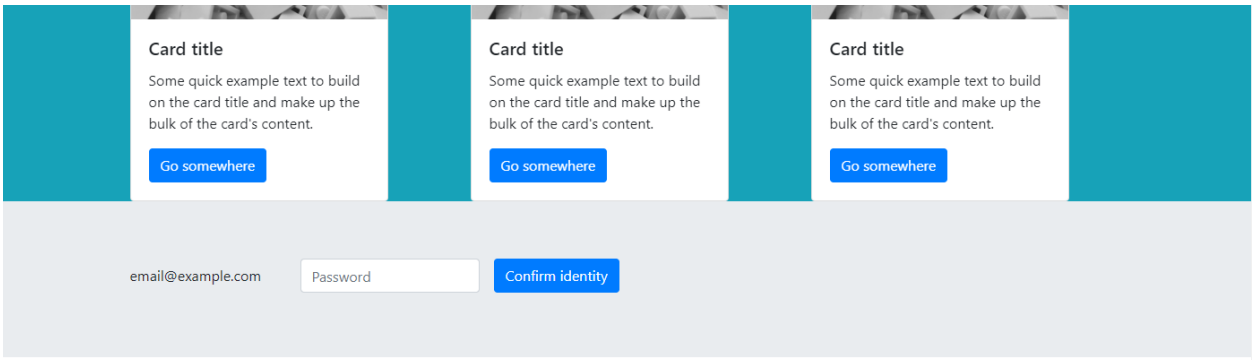
```
<form class="form-inline">
  <div class="form-group mb-2">
    <label for="staticEmail2" class="sr-only">Email</label>
    <input type="text" readonly class="form-control-plaintext" id="staticEmail2" val\
ue="email@example.com">
  </div>
  <div class="form-group mx-sm-3 mb-2">
    <label for="inputPassword2" class="sr-only">Password</label>
    <input type="password" class="form-control" id="inputPassword2" placeholder="Pas\
sword">
  </div>
  <button type="submit" class="btn btn-primary mb-2">Confirm identity</button>
</form>
```

Of course, in the updated layout[20], we're combining the two examples into this:

[18] https://getbootstrap.com/docs/4.3/components/jumbotron/
[19] https://getbootstrap.com/docs/4.3/components/forms/
[20] https://www.codingexercises.com/codelabs/2019-09-30-building-bootstrap-layouts-article-3-pt-6/

```
<div class="jumbotron jumbotron-fluid">
    <div class="container">
        <form class="form-inline">
        <div class="form-group mb-2">
            <label for="staticEmail2" class="sr-only">Email</label>
            <input type="text" readonly class="form-control-plaintext" id="staticEma\
il2" value="email@example.com">
        </div>
        <div class="form-group mx-sm-3 mb-2">
            <label for="inputPassword2" class="sr-only">Password</label>
            <input type="password" class="form-control" id="inputPassword2" placehol\
der="Password">
        </div>
        <button type="submit" class="btn btn-primary mb-2">Confirm identity</button>
        </form>
    </div>
</div>
```

At this point, our layout is almost complete. The newest addition looks like this:



We added the signup area to our layout

As we can see, there are a few problems with this section. There's still dummy text on the input, and we should add something like "Sign-up to our newsletter".

Actually, there are a few other places where our layout could be better-looking.

Improving our layout

Here are some major issues with our layout now:

- The color scheme is sort of inconsistent

- The carousel area's images should be made to look nicer, or the carousel area should be made shorter
- The cards all have dummy text and placeholder images
- There are some margin issues under the breadcrumbs area
- We need to fix the "zoomed out" effect

Let's fix these issues, one by one.

Adding better images to out layout

We'll be using Unsplash images here again.

For the carousel, we'll use these:

1. red shoes[21] by Lindsay Henwood
2. white shoes[22] by Clem Onojeghuo
3. yellow shoes[23] by Danny G

We'll also update the cards, with these images:

1. [dance shoes] by Caitlyn Wilson
2. [casual shoes] by freestocks.org
3. [fashion shoes] by Sylwia Bartyzel

Fixing the color scheme and the dummy text

As far as the color scheme goes, we've just made the backgrounds alternate between `bg-light` and `bg-dark` contextual color classes.

We're still using dummy text in the cards, but now it's shoes-themed dummy text. We've updated the Navbar's brand section similarly, so that it reads "Shoe experts".

[21] https://unsplash.com/photos/7_kRuX1hSXM
[22] https://unsplash.com/photos/jGUCAXEcTtw
[23] https://unsplash.com/photos/htYDlrrKfuM

Fixing margin issues and centering newsletter subscription

We're fixing the margin issues under the breadcrumbs area, using Bootstrap's margin utility classes.

Finally, we're using the narrow container trick (from Chapter 24 of the first book) to easily center the navbar area.

That's it for all the improvements to our layout.

To see the completed layout, check out this link.

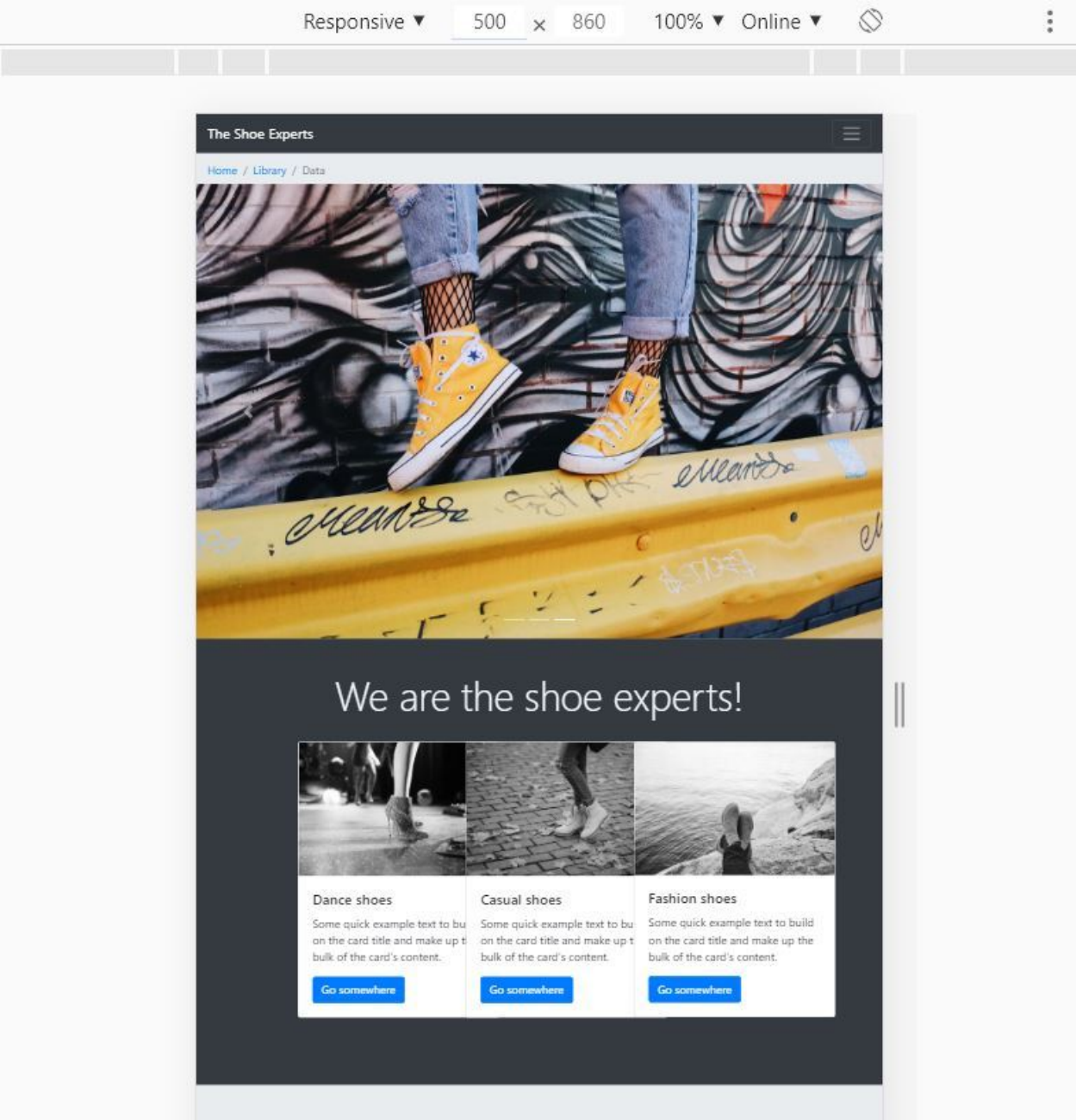
That's it for all the improvements to our layout.

To see the completed layout, check out this link[24].

Fixing the zoomed out "effect"

If you opened the above layout on a mobile, you'll see a weird bug: the entire page is zoomed out.

[24] https://www.codingexercises.com/codelabs/2019-09-30-building-bootstrap-layouts-article-3-pt-7/



One weird bug: the entire page is zoomed out

What we need to do to make it work is add our first `<meta>` tag to the `<head>` element of our layout. This meta tag looks as follows:

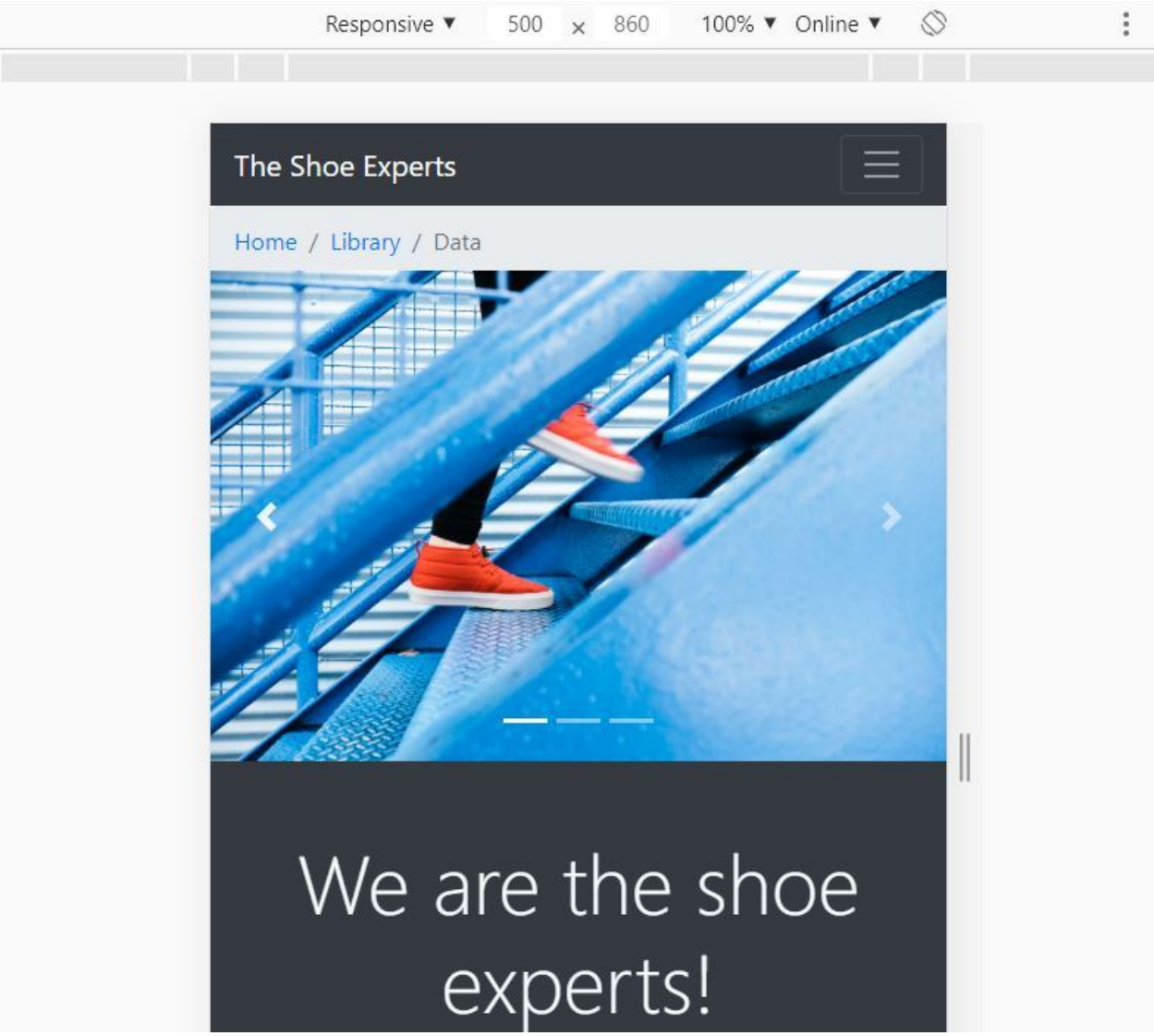
```
<meta
  name="viewport"
  content="width=device-width, initial-scale=1">
```

The above piece of code will fix our layout's zoom issue. The tag is used to make the browser "obey" the width of the screen the layout is viewed on.

Whatever device you're on, the browser will honour the width of the screen (the viewport), as ***the correct width for the layout***. The `initial-scale=1` will make the layout start at the natural zoom of 1:1, but the user can zoom in on a mobile device using finger gestures.

Here's the updated layout[25] after we add the above `meta` tag.

Our site now looks a lot more like what we're used to see on mobile:



The correct zoom is achieved

[25] https://www.codingexercises.com/codelabs/2019-09-30-building-bootstrap-layouts-article-3-pt-8/

Conclusion

In this chapter, we've learned about Bootstrap 4 components, why they're important and how to use them.

We've also seen how they are one of our best tools in prototyping layouts fast.

In the next chapter, we'll explore some other components by building a more complex layout.

Chapter 4: Improving Bootstrap's official examples

In this chapter, we'll upgrade the official Album example[26] that can be found ender the *Custom components* title on the Bootstrap 4 website.

Here's what we've learned so far in this book:

- utility classes and contextual colors
- containers, rows, and column grids
- "primary" and "secondary" components
- solving the zoom problem on layouts (i.e using meta tags)

We've now got most of the absolute basics down.

We could delve into theory and look at flexbox and how flexbox works, so that we're better equipped to work with Bootstrap layouts.

Alternatively, we could simply jump right into layouts, and solve problems when we come across them.

Let's start with this alternative approach, by upgrading an exaple layotu from the Boootstrap docs'; the layout is called Album[27].

Upgrading the official Album layout

We're looking at ways to upgrade the official Album layout[28] for two reasons:

1. To see, in practice, how easy it is to replace components in a layout with other components
2. To practice customizing layouts with both Bootstrap's own CSS, and with custom styles

Let's start by replacing the toggle bar on the top with a regular navbar, copied from the docs[29]:

[26] https://getbootstrap.com/docs/4.3/examples/#custom-components
[27] https://getbootstrap.com/docs/4.3/examples/album/
[28] https://getbootstrap.com/docs/4.3/examples/album/
[29] https://getbootstrap.com/docs/4.3/components/navbar/#nav

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <a class="navbar-brand" href="#">Navbar</a>
  <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#\
navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle naviga\
tion">
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbarNav">
    <ul class="navbar-nav">
      <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="#">Features</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Pricing</a>
      </li>
      <li class="nav-item">
        <a class="nav-link disabled" href="#" tabindex="-1" aria-disabled="true">Dis\
abled</a>
      </li>
    </ul>
  </div>
</nav>
```

We'll also upgrade the *Album example* area. Bonus points for you if you realized that this section is actually a Jumbotron.

Here's the custom CSS we'll give it:

```
.jumbotron-image {
        background-image: url('https://images.unsplash.com/photo-1458560871784-56d23406c091\
?ixlib=rb-1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=1934&q=80');
        background-size: cover;
        background-repeat: no-repeat;
}
```

We'll now add this new `jumbotron-image` class to update our jumbotron area, like this:

```
<section class="jumbotron jumbotron-image text-center">
```

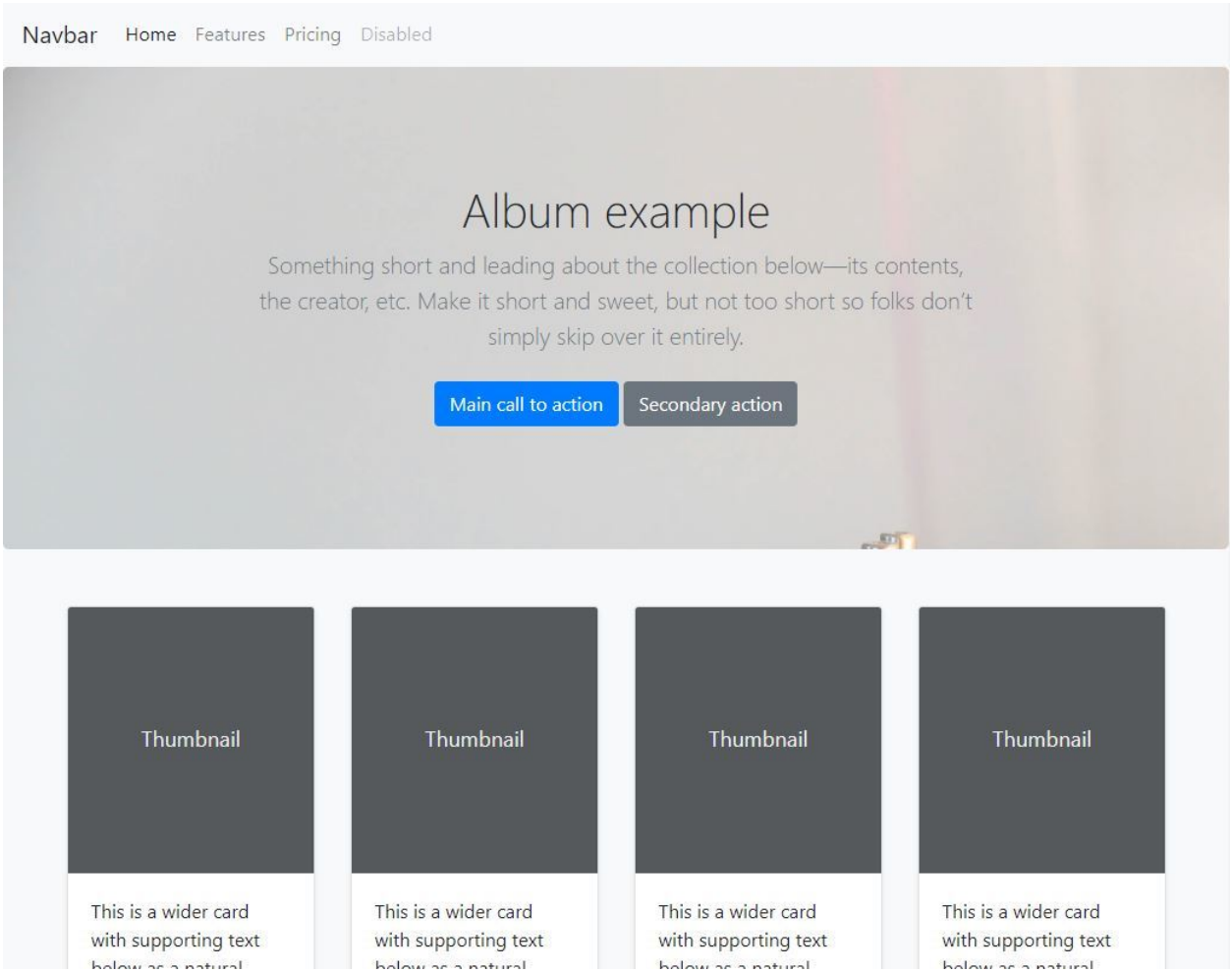
Above, we can see an example of *not overriding Bootstrap's default classes, but rather extending them.* It's something we've discussed in Book 1, Chapter 24.

Fun fact: The image used for the above jumbotron background is from Adrian Korte[30].

As for the cards, let's fit four cards instead of three on each row. The update we're making is that inside each row, we need to make sure to have the following structure:

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

After these updates, here's our customized Album layout's screenshot:



Altered album layout

As we can see, our layout looks a bit better, but we've still got some improvements to add.

[30] https://unsplash.com/photos/5gn2soeAc40

For example, we could update the background image so that the gramophone could be seen. That would mean that we need to show the background image from the bottom.

We could also change the navbar's color to a darker background and turn letters a lighter color.

Let's make these improvements.

Improving the upgraded Album layout

First, we'll fix the image by just adding another CSS declaration to our custom `jumbotron-image` class:

```
background-position: bottom;
```

Next, we'll fix the navbar colors, from this:

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
```

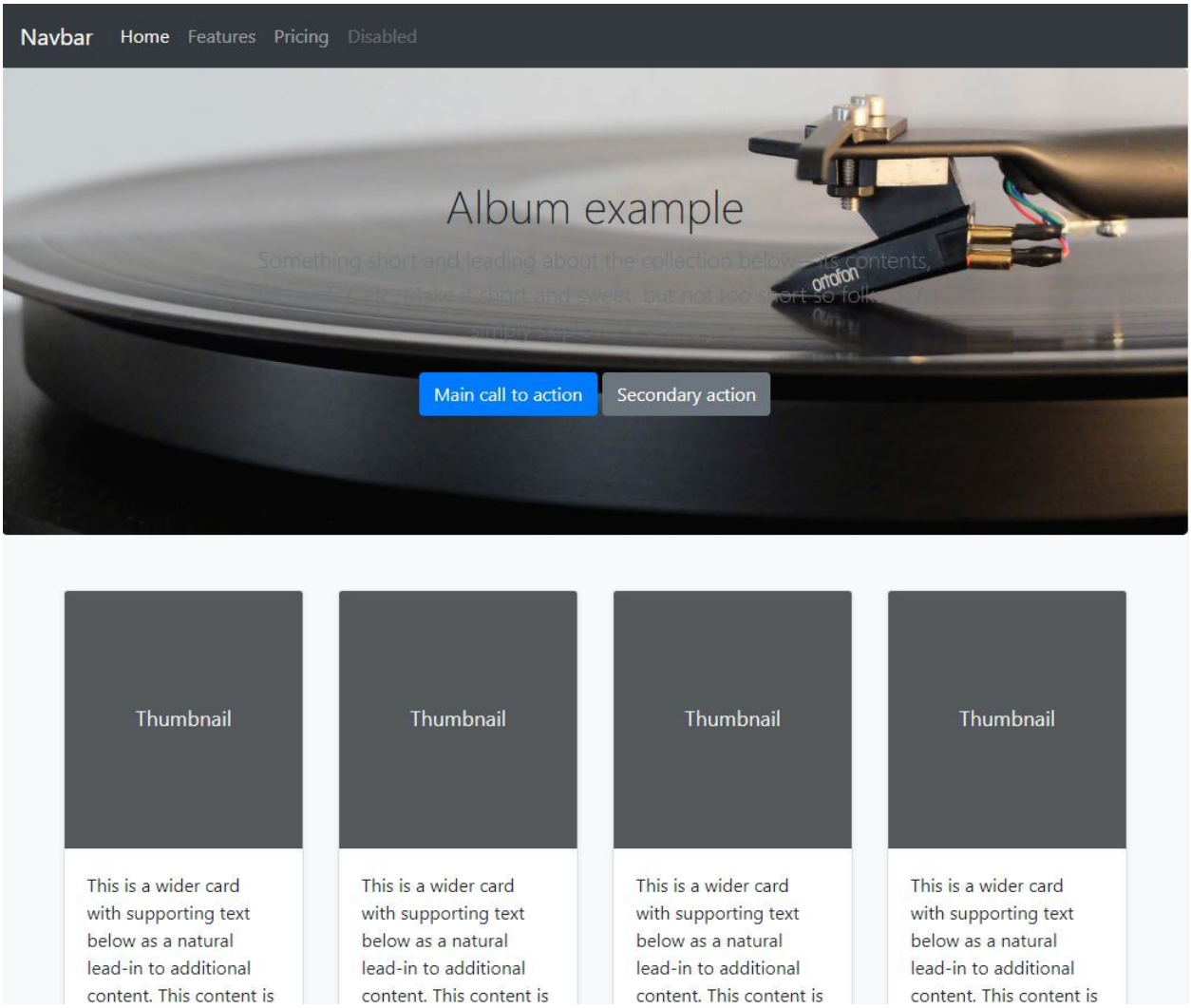
...to this:

```
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
```

Here's the updated layout[31].

And here is the screenshot after the update:

[31] https://www.codingexercises.com/codelabs/2019-10-01-building-bootstrap-layouts-article-4-pt-2/



Altered album layout with dark navbar

While our layout does look better in general, now our jumbotron's text is not showing any more. Let's see a quick fix for this problem.

Making dark letters on dark background visible with Bootstrap's contextual `bg-*` classes

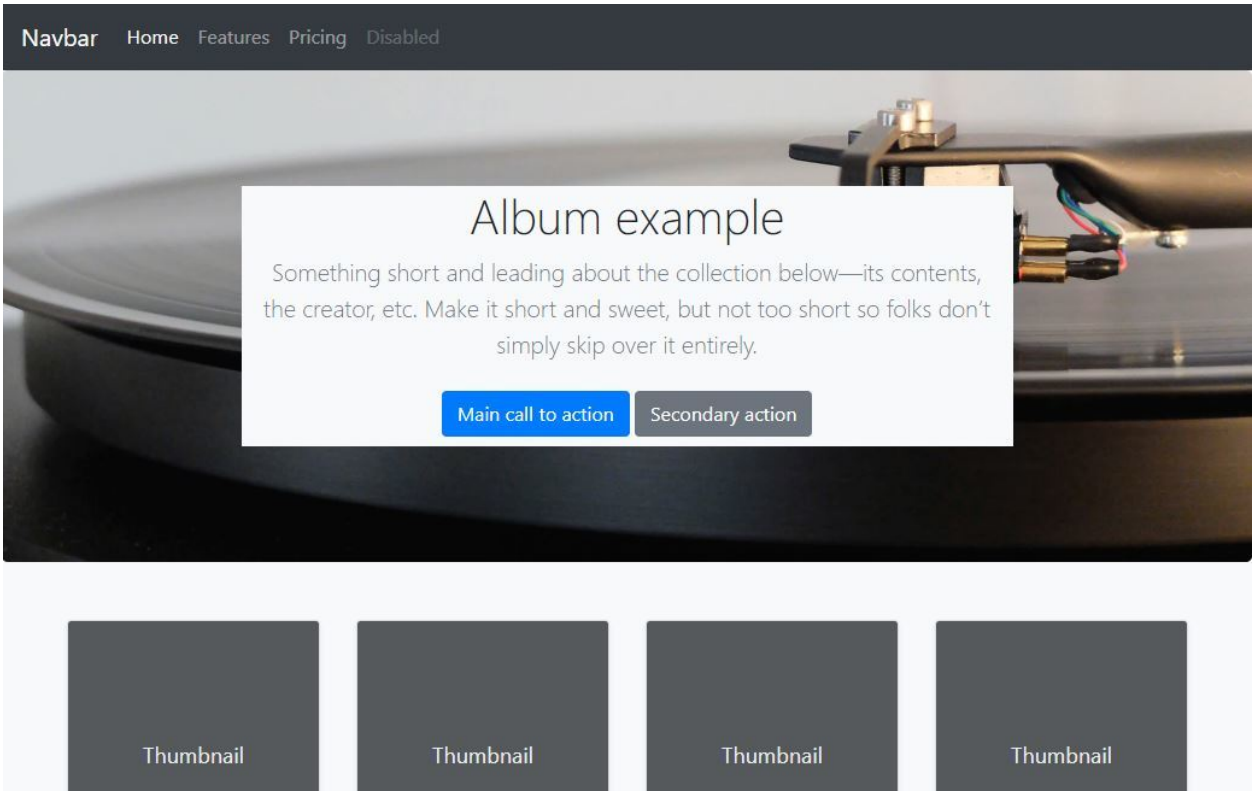
To make these letters visible, let's look at our layout's jumbotron again:

```
<section class="jumbotron jumbotron-image text-center">
    <div class="container">
    <h1 class="jumbotron-heading">Album example</h1>
    <p class="lead text-muted">Something short and leading about the collection belo\
w—its contents, the creator,
        etc. Make it short and sweet, but not too short so folks don't simply skip o\
ver it entirely.</p>
    <p>
        <a href="#" class="btn btn-primary my-2">Main call to action</a>
        <a href="#" class="btn btn-secondary my-2">Secondary action</a>
    </p>
    </div>
</section>
```

To quickly make the letters visible, let's just add a contextual background color to the div with the class of `container`, like this:

```
<div class="container bg-light">
```

With just this one little change, our layout will update to this:



Dark letters with dark background with bg-light

If you're in a hurry to complete a layout, sometimes just adding a few little updates, such as this one, is enough.

However, we could also improve this quick fix further, by just adding the alpha to the background color.

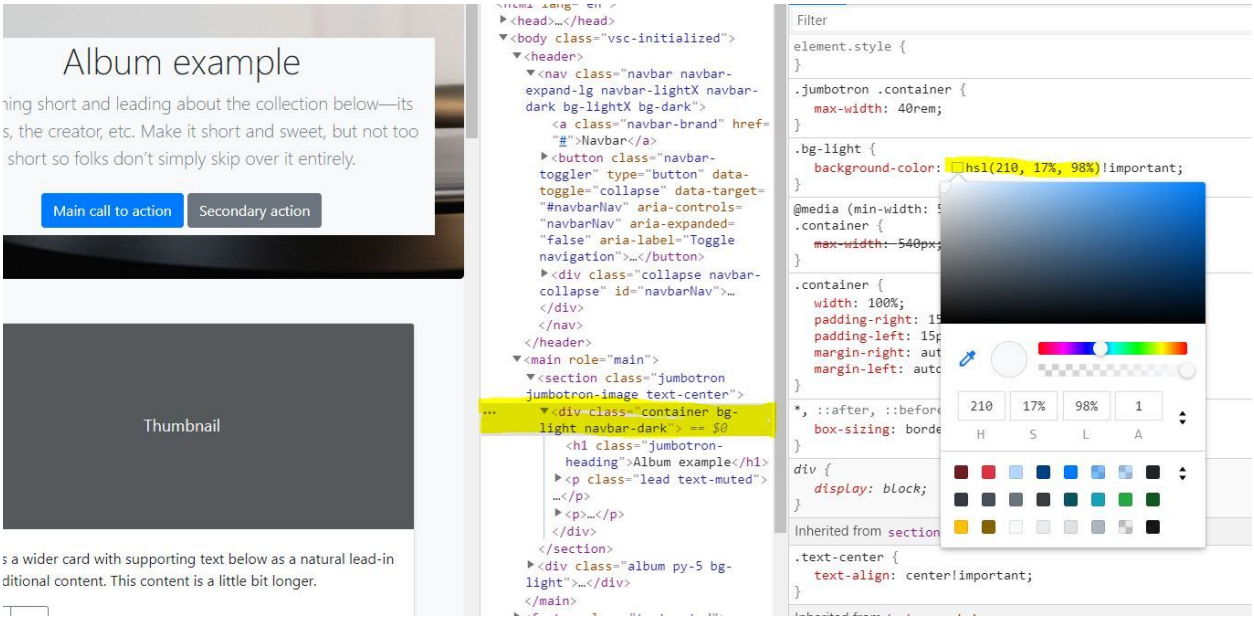
Adding transparency to the background color

Note that this is not Bootstrap-specific, it's a simple CSS technique you can use with or without a CSS framework.

Here's how:

1. Right click on the background you want to change, and click "Inspect" to open developer tools
2. In this case, we're looking for the `bg-light` class in the *Styles* sub-tab of the devtools *Elements* tab
3. Once you've found it, you'll see `background-color: #f8f9fa!important;` inside the `bg-light` CSS class; there's also a little gray box - a preview of our background color
4. Hold down the SHIFT key and click on the little gray color preview square. Doing this will change the color notation from hex to rgb, then to hsl, and finally back to hex
5. Once you've switched to hsl, just click onto the little gray box, and you'll see a color picker.
6. At the bottom of the color picker, there's an input for each of the HSLA values. To get a transparent version of your gray color, just lower the A (alpha) to any value under 1. The lower the value, the greater the transparency.

Here's a screenshot of how your screen should look before step 6:



Dark letters with dark background with bg-light

When you've found a reasonable alpha value, you can simply copy-paste the color into your layout.

Remember not to override the existing class! This time we'll make an entirely new class and not even use the default `bg-light`:

```
.bg-lighter {
    background-color: hsla(210,17%,98%, 0.75);
}
```

Here's the completed layout[32].

Conclusion

In this chapter, we've seen how easy it is to alter Bootstrap's example layouts.

We've also seen how building layouts with a CSS framework basically consists of combining two approaches:

- Using the framework's components
- Adding custom CSS

That's it for this chapter. In the next one, we'll improve another default Bootstrap layout.

[32] https://www.codingexercises.com/codelabs/2019-10-01-building-bootstrap-layouts-article-4-pt-3/

Chapter 5: Working with SCSS in Bootstrap 4

In this chapter, we'll upgrade the official Pricing example[33] that can be found under the *Custom components* title on the Bootstrap 4 website.

Let's have a look at the pricing example on getbootstrap.com[34].

What's the difference between it, and the premium pricing example here[35]?

Obviously, they have almost the same features, except that the premium pricing example looks a lot better than the default one.

In this article, we'll look at how we can combine the two layouts; that is, we'll see how we can copy some of the features from the premium layout so that our default layouts gets some customization.

Note that building a beautiful layout takes skill and time, so please buy the Landkit layout if you want to use it[36]. Also note that this is not an affiliate link, I just want to remind you that it's a paid product and I'm only showing you an example of how to learn building layouts by looking at some premium ones.

Ok, let's begin by copying the theme's CSS.

Copying CSS from another Bootstrap layout into our own

Sometimes the quickest way to freshen up a layout is to copy the styles from another one.

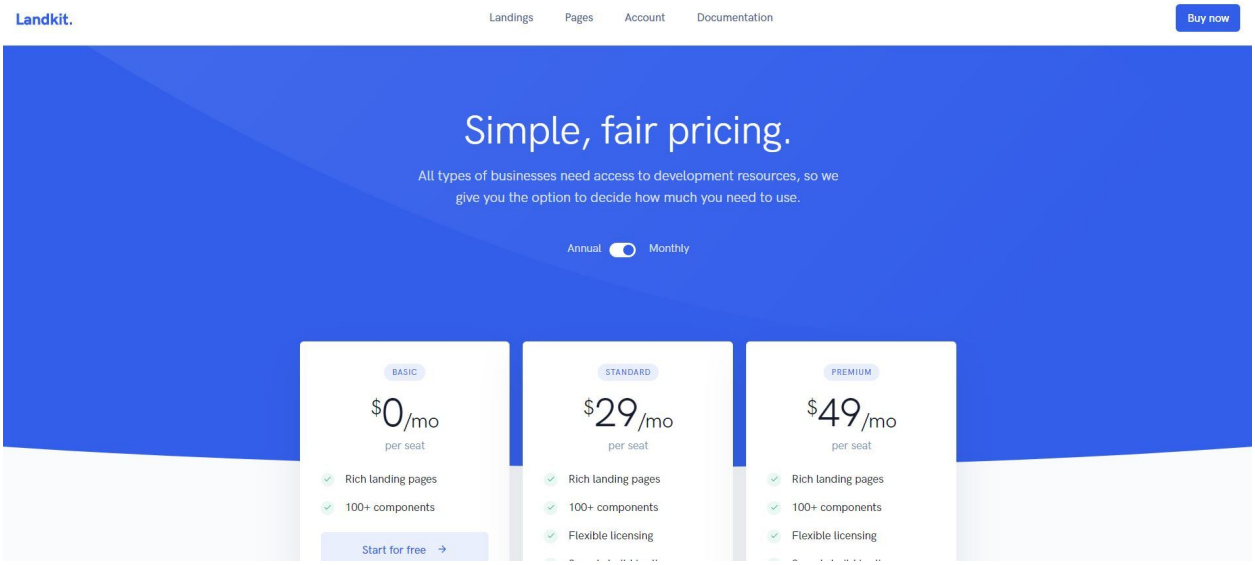
So let's locate the main stylesheet of the Landkit theme and copy the CSS into our own theme.

Doing this will involve lots of work in the developer tools, so if you wanted to learn more about how to work with devtools in Chrome, this will be a nice tutorial for you.

Locating the theme's main CSS file and unminifying it

Let's begin by opening the Pricing layout for the Landkit theme, available here[37].

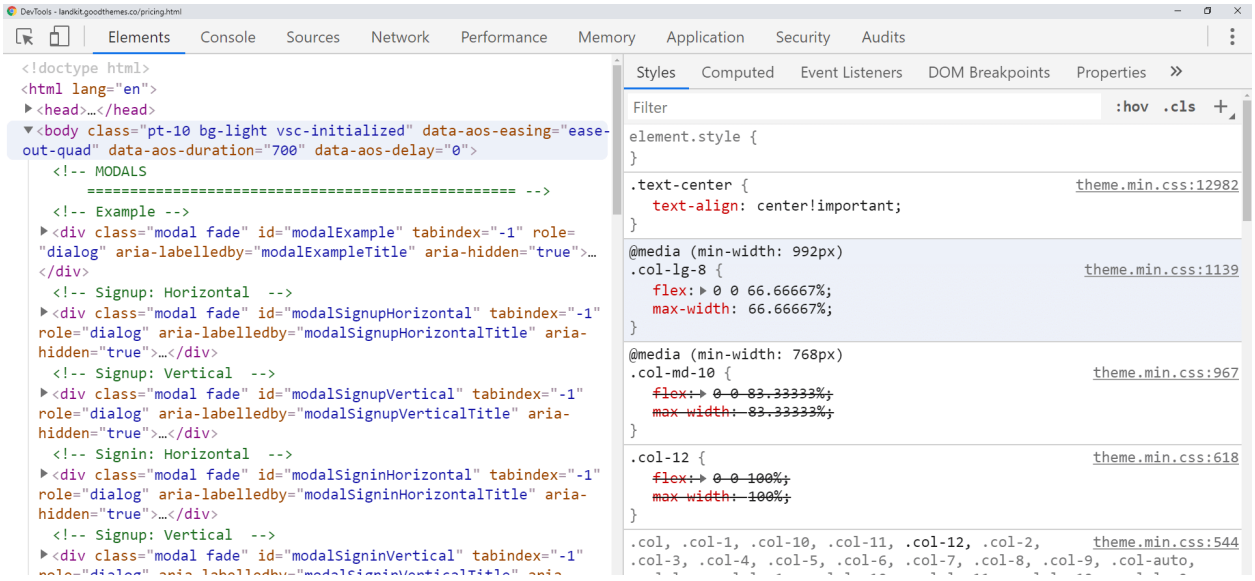
[33] https://getbootstrap.com/docs/4.3/examples/#custom-components
[34] https://getbootstrap.com/docs/4.3/examples/pricing/
[35] https://landkit.goodthemes.co/pricing.html
[36] https://themes.getbootstrap.com/product/landkit/
[37] https://landkit.goodthemes.co/pricing.html



Pricing layout Landkit theme

Next, let's turn on the developer tools.

Interestingly, this very same layout that you see above, can be represented like this:



Pricing layout Landkit theme in devtools

It's the same webpage! The first screenshot is the render, the second one is the DOM - the code - of the page in developer tools.

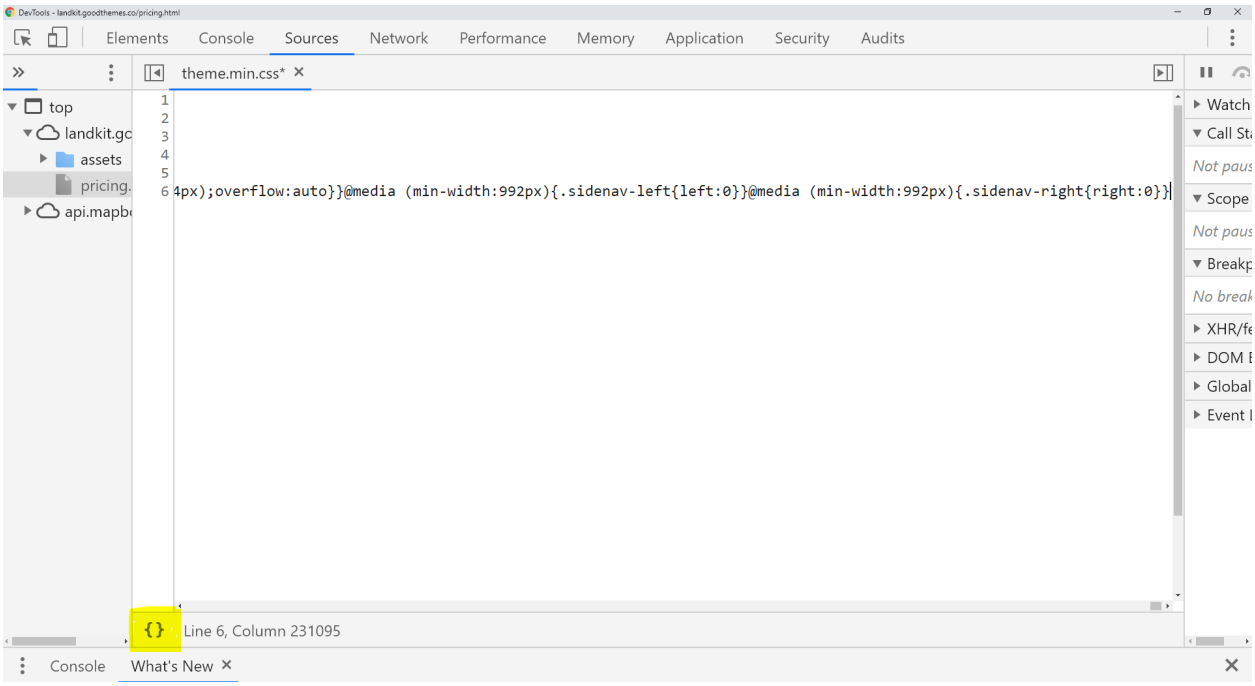
The devtools screenshot shows the HTML structure on the left half of the picture, and the styles that affect the HTML structure in the right half of the picture.

On this right-hand side, inside the *Styles* tab, you can see some CSS class definitions (such as `text-center`, `col-lg-8`, `col-md-10`, and `col-12`). But, where are these classes coming from? Devtools

actually helps us with that.

To the right of these listed CSS class definitions, there's the title of the CSS file where each CSS class definition is located. Note that for each of them, there is the same source file: `theme.min.css`.

The title of the file - in this case, the `theme.min.css` - is actually a link, so if you click on it, you'll see the file inside devtools.

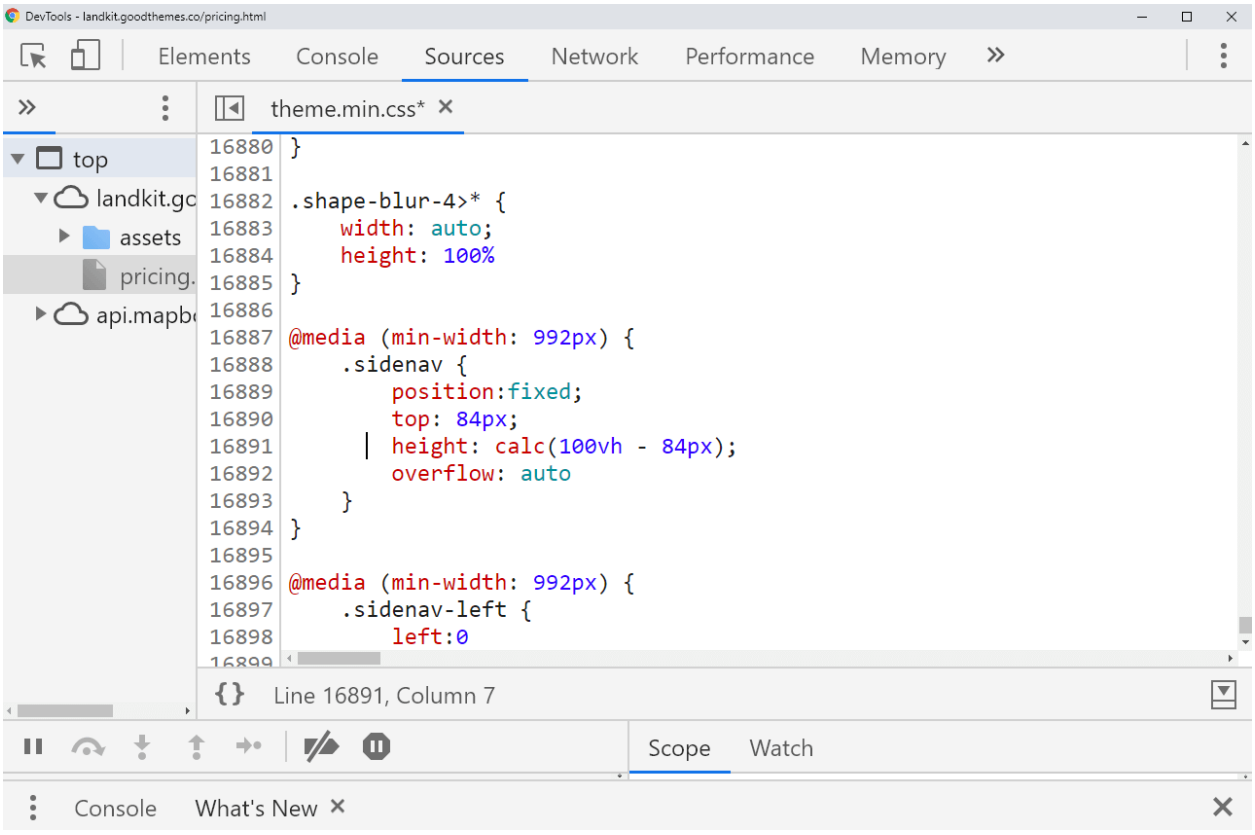


The theme CSS file

Note that this file is ***minified***, meaning that it's been stripped of whitespace, which effectively makes the entire file into one big line of CSS.

To ***unminify*** it - meaning, to make it human-readable - you can simply click the `{}` sign, highlighted in yellow in the picture above (hovering over that sign will make a little tooltip apprear - the tooltip reads the word "Format").

If you click it, you'll see the same CSS file, unminified.

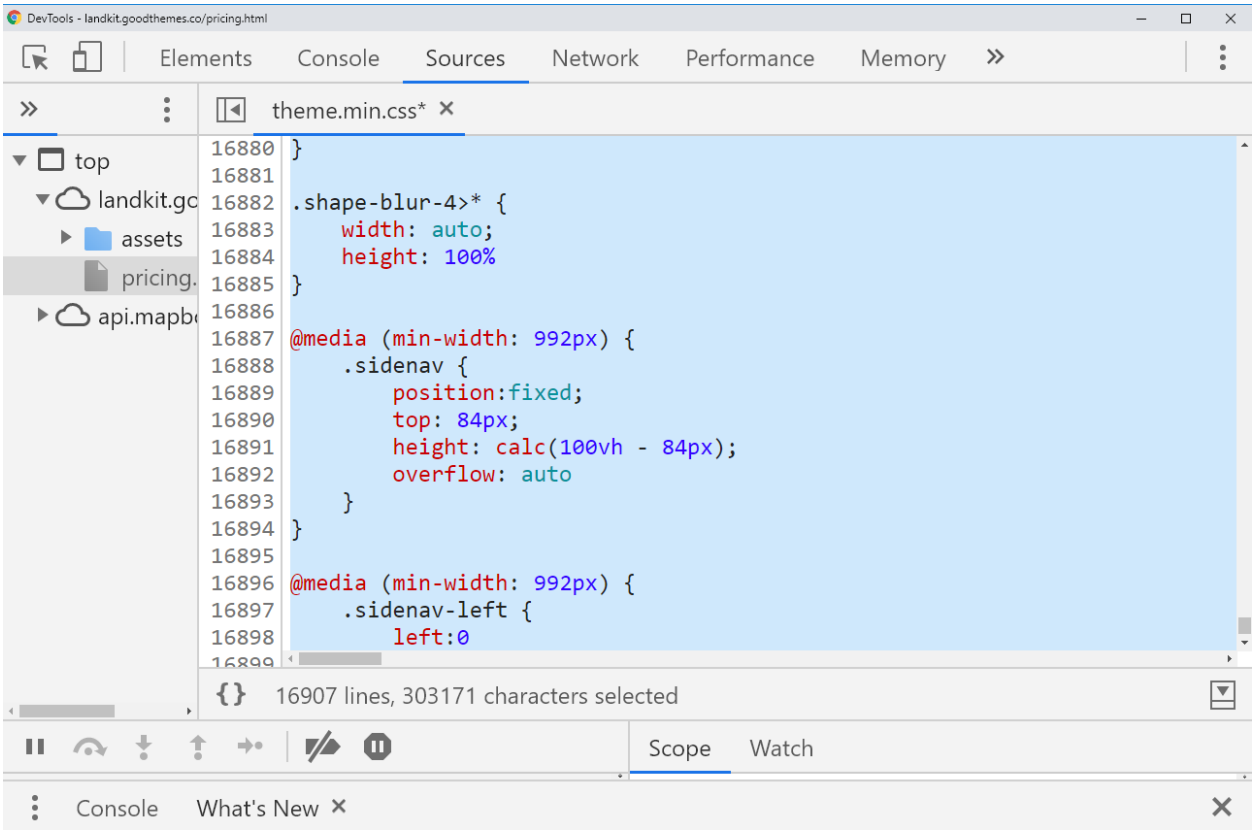


The theme CSS file, unminified

Regardless of how you view the file, the next thing to do is copy the whole thing.

Copying the theme's CSS file

To do that, simply click inside the file, and press the `CTRL a` keyboard combination. This will select the entire file.

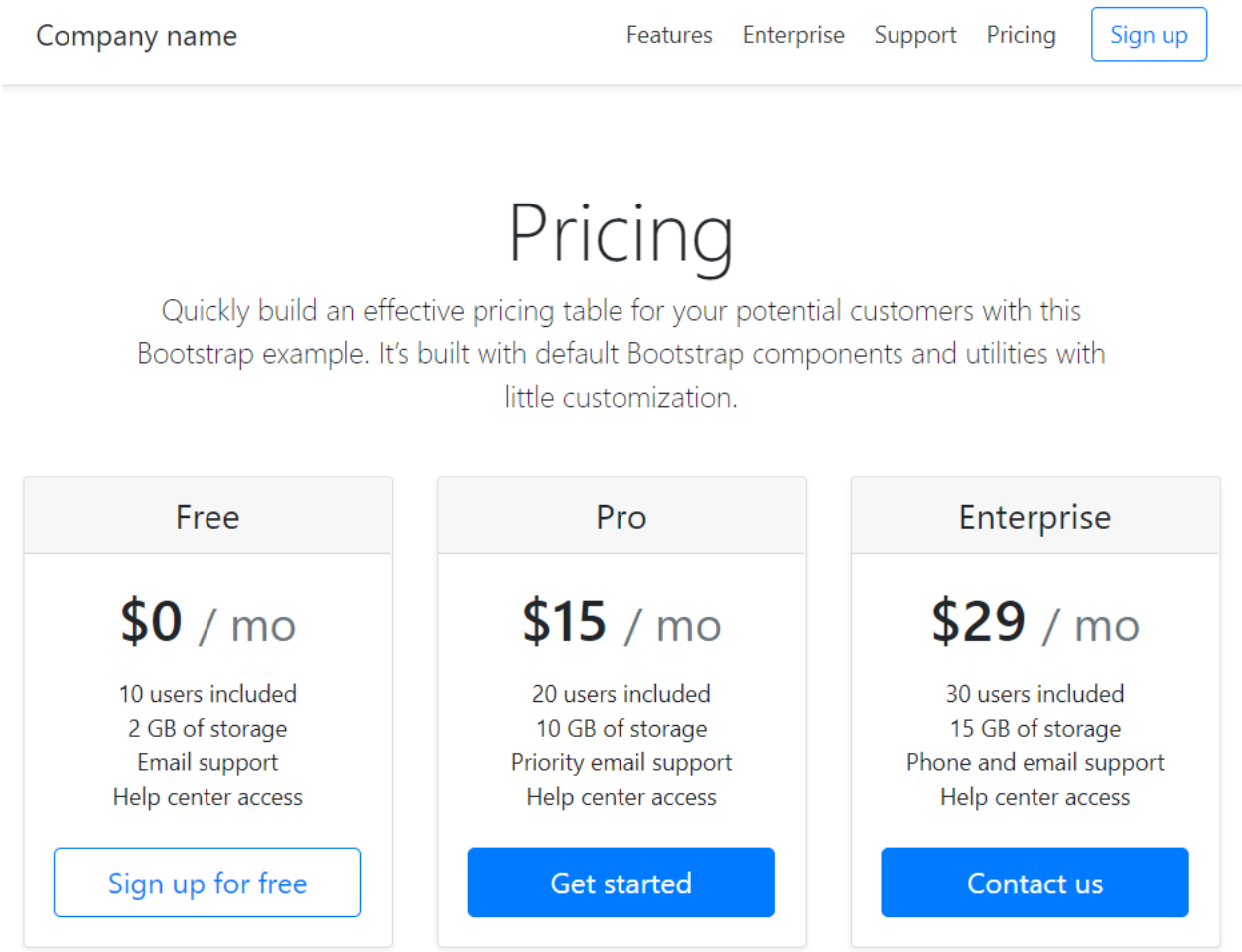


The theme CSS file, unminified, selected

Now we need to copy the file; the easiest way is to just press `CTRL c` keyboard shortcut.

Next, we need to go back to our default pricing theme from the Bootstrap docs[38].

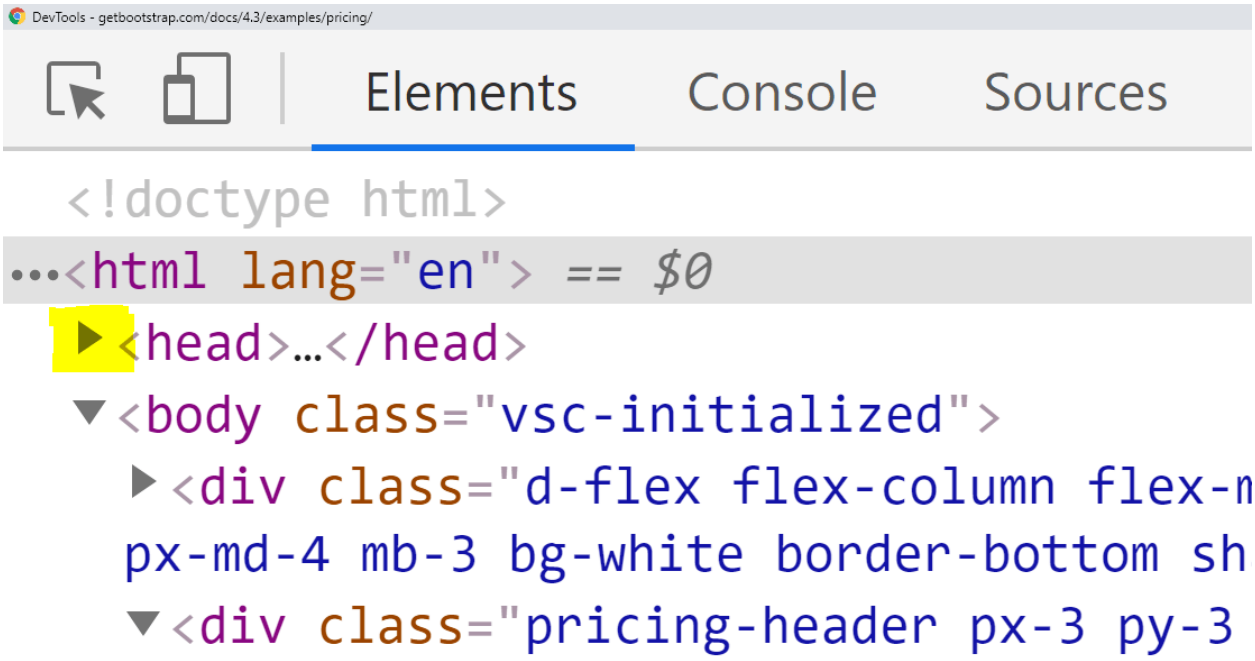
[38] https://getbootstrap.com/docs/4.3/examples/pricing/



The default Bootstrap 4 pricing theme

Pasting in the copied CSS code into the default theme

Back inside the default pricing theme, we'll open the elements pane inside the devtools: the goal is to paste in the entire copied CSS inside a new `<style>` tag in the `<head>` of the default pricing theme.



The elements pane (zoomed in)

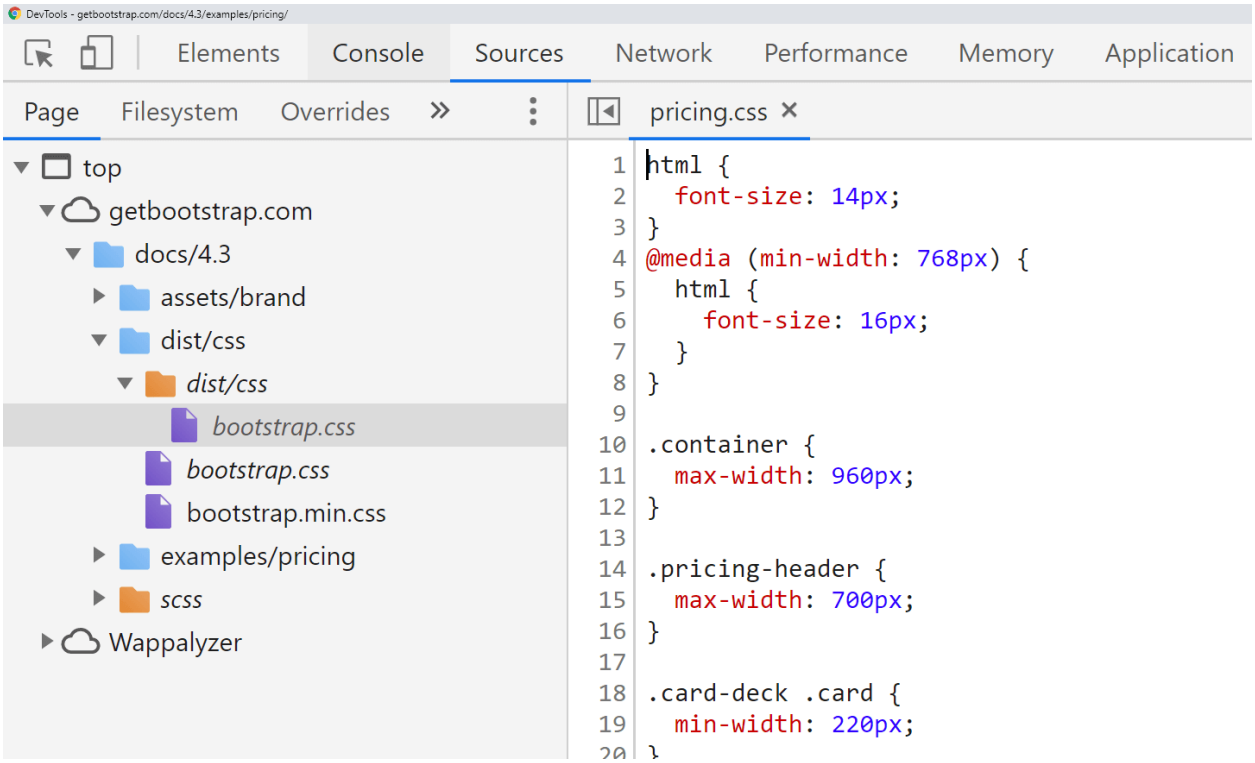
The highlighted little triangle in front of the `<head>` tag shows you where to click to "twirl open" the head tag so you can see the elements inside it.

Once you twirl open the `<head>` element, scroll all the way down to its closing `</head>` tag. Right above the closing `</head>` tag, you'll see the following line of code:

```
1  <link href="pricing.css" rel="stylesheet">
```

The `pricing.css` file holds the custom styles for the default bootstrap pricing example. To see the actual file, right-click on the actual value of the `href` attribute, `pricing.css`. A contextual right-click menu will appear; click on the first menu item, the one that reads "Reveal in sources panel".

Once you've clicked the "Reveal in sources panel" command, this is what you'll get:



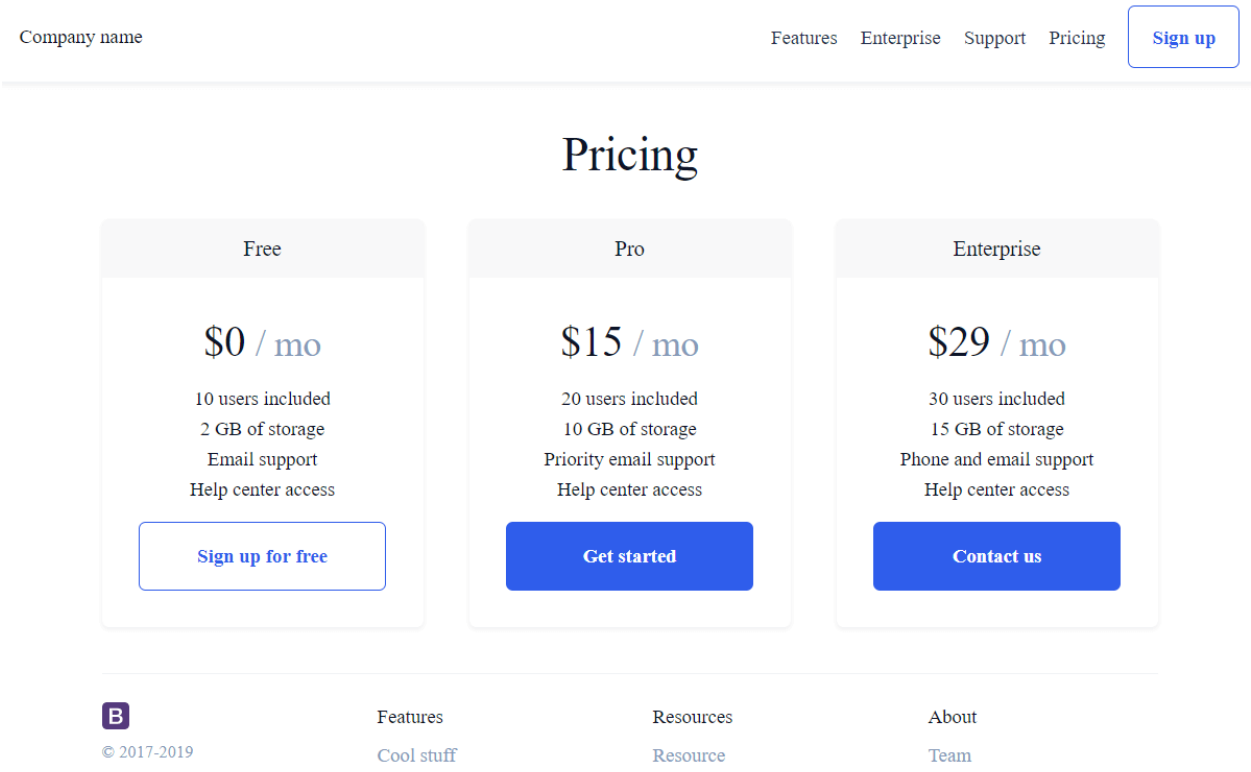
Revealing a CSS file in the Sources panel

Now we'll paste in the CSS we copied from the premium theme:

1. First, click inside the `pricing.css` file
2. Next, press the `CTRL END` keyboard shortcut to select all the file's contents
3. Finally, press the `CTRL v` keyboard shortcut to paste in the copied code

You'll immediately see that the default bootstrap layout has been updated.

Here's the default theme with the new styles applied:



Revealing a CSS file in the Sources panel

We can see that just like that, we've pasted in a lot of styles from the original theme.

If you'd like to compare your updated layout with the previous one, here's a little trick you can use: with the Elements tab in focus in developer tools, press the `CTRL z` shortcut keys to undo the update to the stylesheet. Pressing the `CTRL y` will redo the update to the stylesheet.

Now that we've seen how fast and easy it is to copy-paste the styles from a different theme, we need to see how we can save our changes.

Saving the changes to our theme and trimming the unused CSS

We'll begin by saving the default Bootstrap theme to our local machine.

The fastest way to save it, is to first open it in Chrome, at this url[39].

Next, just right click anywhere in the layout, and click the "Save as..." command.

This will open a dialog box, and you can use your operating system's file system to save it to a location you choose. It's best to save it to a new folder; I called mine `the-theme`.

Once you've saved it to a new folder, this is the folder structure:

[39] https://getbootstrap.com/docs/4.3/examples/pricing/

```
the-theme/
  ├── Pricing example · Bootstrap_files/
  │   ├── bootstrap.min.css
  │   ├── bootstrap-solid.svg
  │   └── pricing.css
  └── Pricing example · Bootstrap.html
```

The folder structure of the-theme folder

Now, all we need to do is update the `pricing.css` file: just go back to the premium theme, locate the CSS file as described above, copy its' contents and paste it back into the `pricing.css` file.

Now just double click the `Pricing example · Bootstrap.html` file to view it in the browser. If you want to open it in a specific browser, but that browser's not the default one in your system, you can simply click on the file, than drag and drop it into the browser of your choice. Some operating systems will also simply let you choose the browser.

Now we can trim unused CSS, all with the help of the Chrome browser.

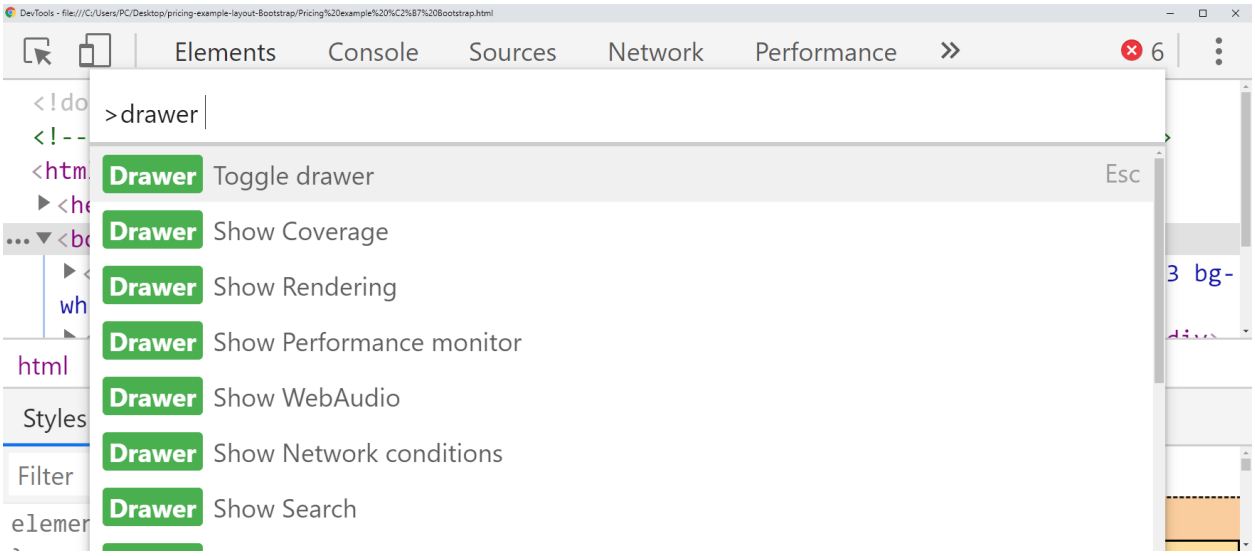
Trimming unused CSS with Chrome's devtools

With our local copy of the pricing example layout open, let's open the devtools again.

Once the devtools window is in focus, we'll use a keyboard shortcut to open ***the Command Menu***.

The shortcut is `CTRL SHIFT p`. You can just type into the command input that appears. The command we need is ***drawer coverage***.

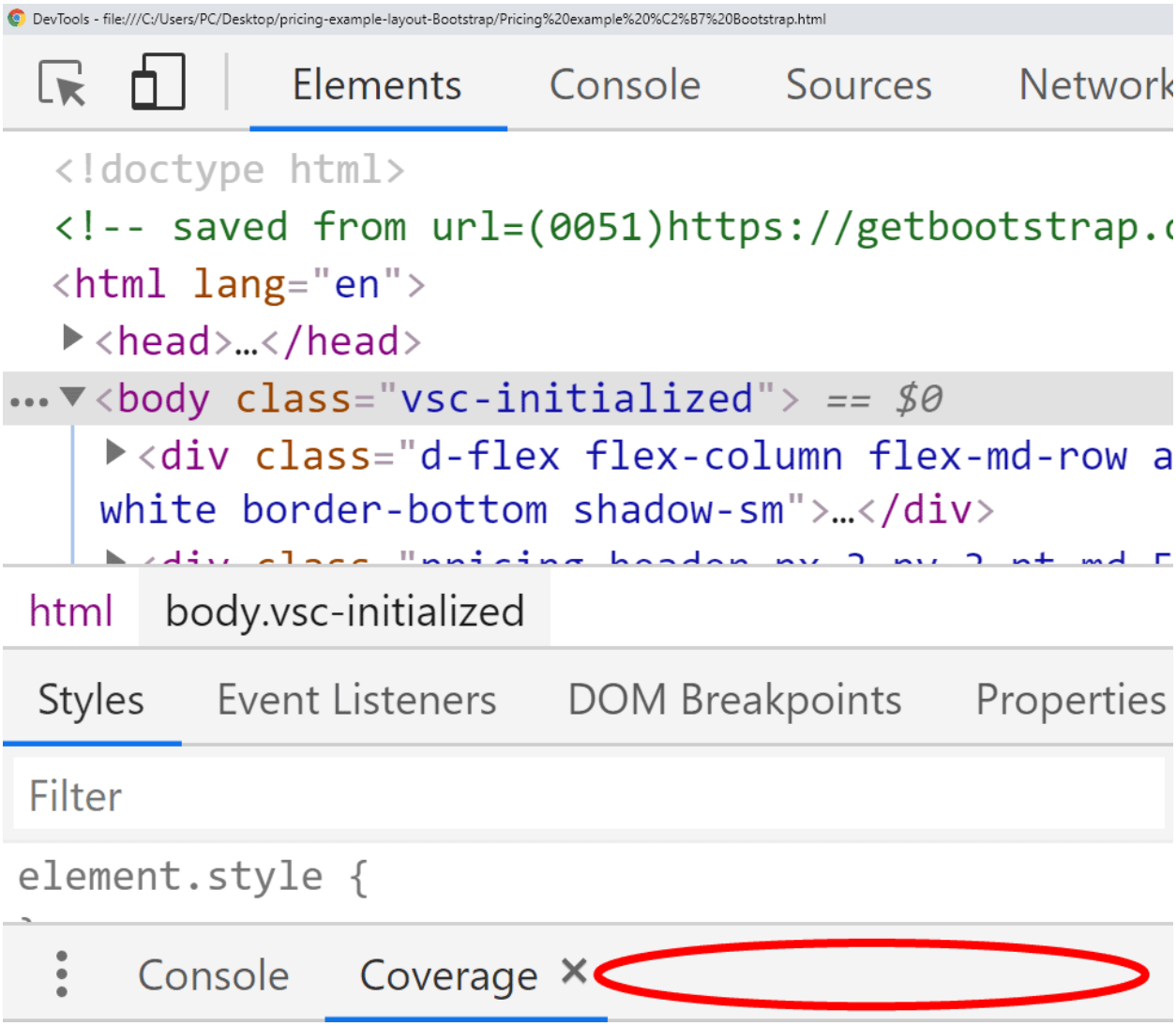
Even if you typed a few letters, the commands matching what you've typed will start appearing:



Launching Chrome's command menu

Click on the ***Show Coverage*** command.

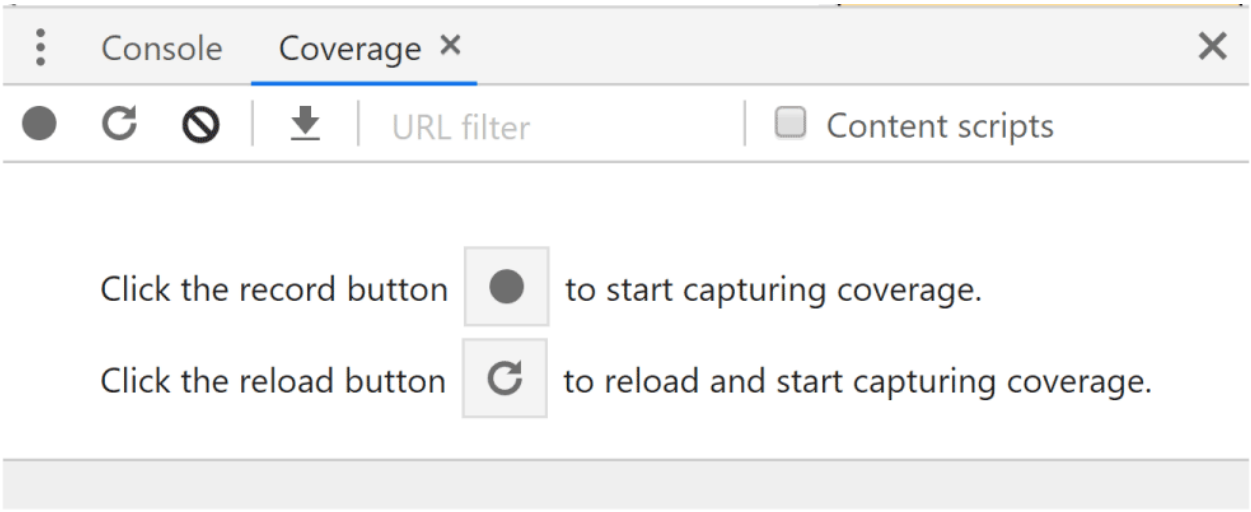
The ***Coverage tab*** will appear, but sometimes you wont' easily notice it, like in the following screenshot:



Coverage tab is sometimes hard to locate

In the above image, you can see the Coverage tab "hidden" at the very bottom of the devtools window. To see more of it, you can simply hover your mouse on the gray background to the right of the Console and Coverage tabs. The area where you should hover the mouse is highlighted with a red ellipse in the image above. Doing this will make a two-pointed arrow appear; now you can click, hold, and drag up the Coverage tab, to see more of it.

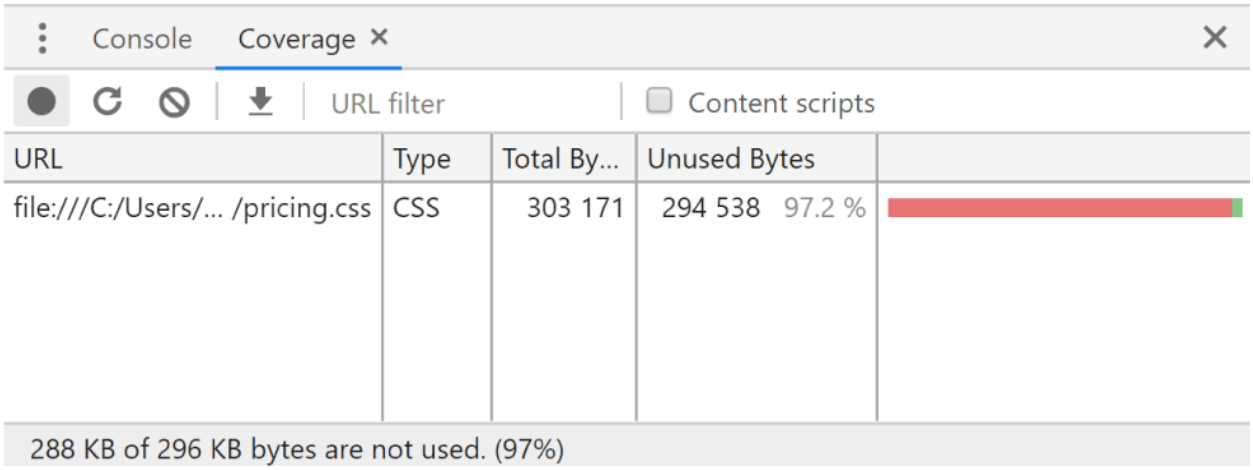
Once you've done that, the screen should look like this:



Coverage tab options

As the instructions above say, now you need to click the gray dot to start recording the coverage. Once you do, the gray circle will turn red. You can click on it again to stop it in a few seconds.

The Coverage tab will show the CSS file, the percentage of unused CSS, and the horizontal bar colored in red and green. The red color shows the amount of unused code; the green color shows the amount of the code that's actually used.



The result of recording CSS coverage

Now you can double-click on the row that shows unused bytes; this will open up the CSS file with green or red vertical bars showing, depending on whether or not the CSS is used.

To test it out, you can dowload the project up to now from this zip file[40].

After double-clicking the row that shows the unused bytes in the `pricing.css` file, we can see the exact unused CSS, line by line: the css that's used will have a green bar to its left, next to line numbers. All the other lines will have a red bar to its left.

[40] https://www.codingexercises.com/downloads/2019-10-02/pricing-example-layout-Bootstrap.zip

One way to remove unused CSS is to simply make a local copy of your CSS file and cut out all the unused CSS from it. Alternatively, you can do the opposite, and use the unused bytes information from Chrome to copy-paste the used code into a brand new file, than save that file as the CSS file to use.

Hopefully, if your project's not too big, this shouldn't take too much time.

Another way to remove unused CSS is to automate it with a taskrunner such as Grunt or Gulp, and a plugin such as UnCSS.

Yet another way to do it is to use Purge CSS with webpack[41] (a JavaScript module bundler[42]).

Whatever you decide to do, just make sure you don't remove unused CSS before you're absolutely sure you won't use it later. With automated tools this is not such a huge issue, since you can have the source CSS with all the unused code, and then the production-ready CSS with only the code that is being used.

Let's now go back to our layout and complete the updates to the official pricing example.

Completing the changes to the pricing example layout

To complete our layout, we'll do only a couple of things:

- Set Arial as the layout's font
- Update the Pricing "jumbotron" section

To set Arial as the layout's font, we can just add this CSS to our layout:

```
* { font-family: Arial !important }
```

Next, we'll update the pricing section, which currently looks like this:

```
<div class="pricing-header px-3 py-3 pt-md-5 pb-md-4 mx-auto text-center">
  <h1 class="display-4">Pricing</h1>
  <p class="lead">Quickly build an effective pricing table for your potential custom\
ers with this Bootstrap example. It's built with default Bootstrap components and ut\
ilities with little customization.</p>
</div>
```

We'll simply alter the above code by adding another class to wrap the `h1` and `p` tags, and give it a class of `container-narrow`:

[41] https://www.codingexercises.com/guides/understanding-node-npm-and-javascript-modules#what-is-webpack
[42] https://www.codingexercises.com/guides/understanding-node-npm-and-javascript-modules#module-bundlers-a-way-to-use-modules-in-the-browser

```
<div class="pricing-header px-3 py-3 pt-md-5 pb-md-4 mx-auto text-center">
    <div class="container-narrow bg-secondary">
        <h1 class="display-4">Pricing</h1>
        <p class="lead">Quickly build an effective pricing table for your potential \
customers with this Bootstrap example. It's built with default Bootstrap components \
and utilities with little customization.</p>
    </div>
</div>
```

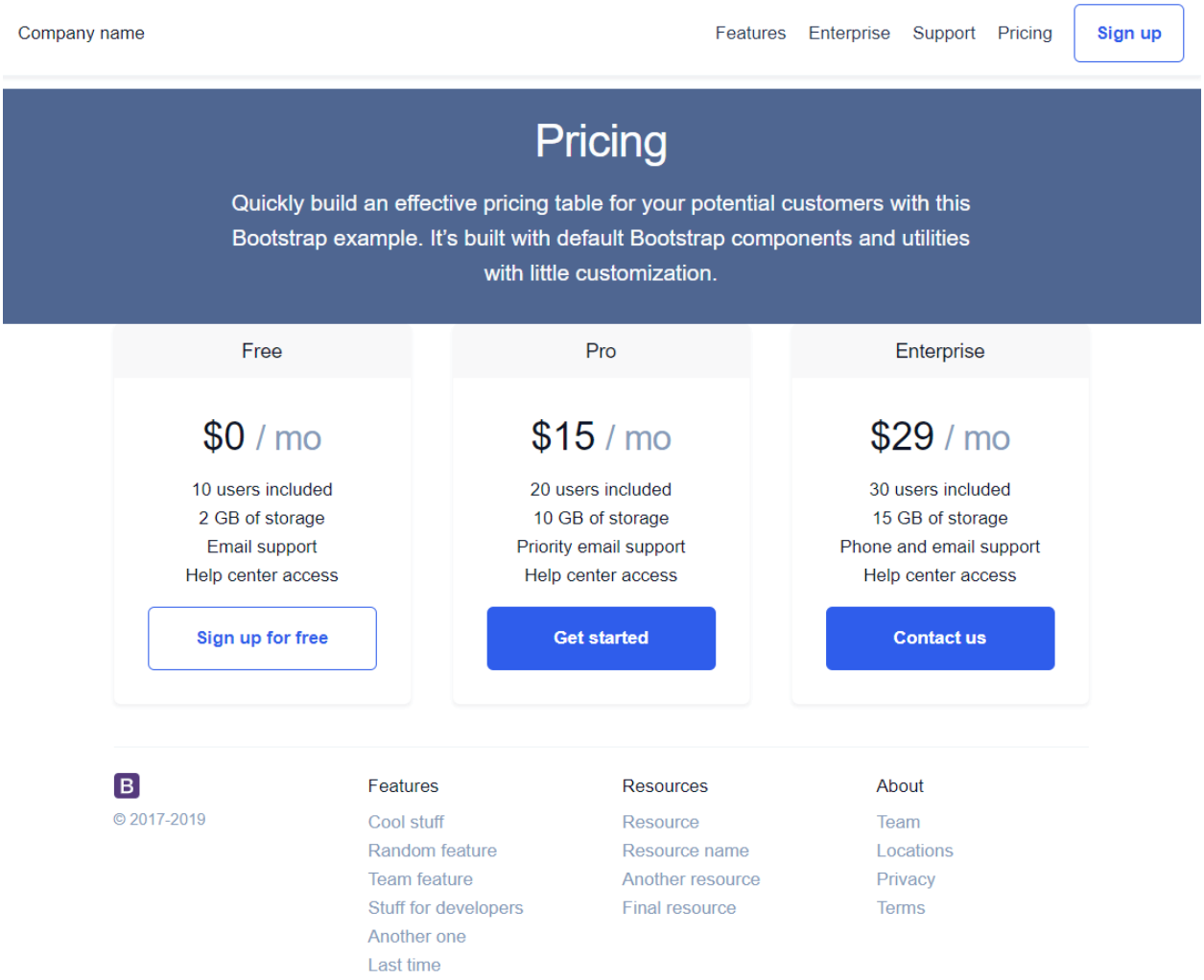
Next, we'll add the class to our CSS:

```
.container-narrow {
    max-width: 700px;
    margin: 0 auto;
}
```

Note: we've discussed how `margin: 0 auto` works in Book 1 in this book series, in chapter 24.

We've also added the classes of `bg-primary` and `text-light` to the outmost wrapping div, to make our pricing copy look a bit nicer.

After all these changes, here's the result:



The pricing layout customized with bg-secondary

What we can do now to make our layout look nicer is:

- remove the margin between the navbar and the pricing section
- make the pricing background drop down some more (so that its bottom section fits under the vertical middle of pricing cards)
- do something about these buttons!

Removing the margin between the navbar and the pricing section and making the pricing background drop

A quick way to "remove the margin is to "lift" the pricing section up, by adding its outer-most wrapping div the following CSS:

```
position: relative;
top: -20px;
```

Next, we need to give it the `min-height` of, say, 350 pixels:

```
min-height: 350px
```

Finally, we'll move up the cards section by giving them a negative value for `margin-top` on this HTML:

```
<div class="card-deck mb-3 text-center">
```

We'll just add some inline style to it, because that's the fastest way to do it (meaning, we don't have to make a CSS class and then reference it):

```
<div class="card-deck mb-3 text-center" style="margin-top: -150px">
```

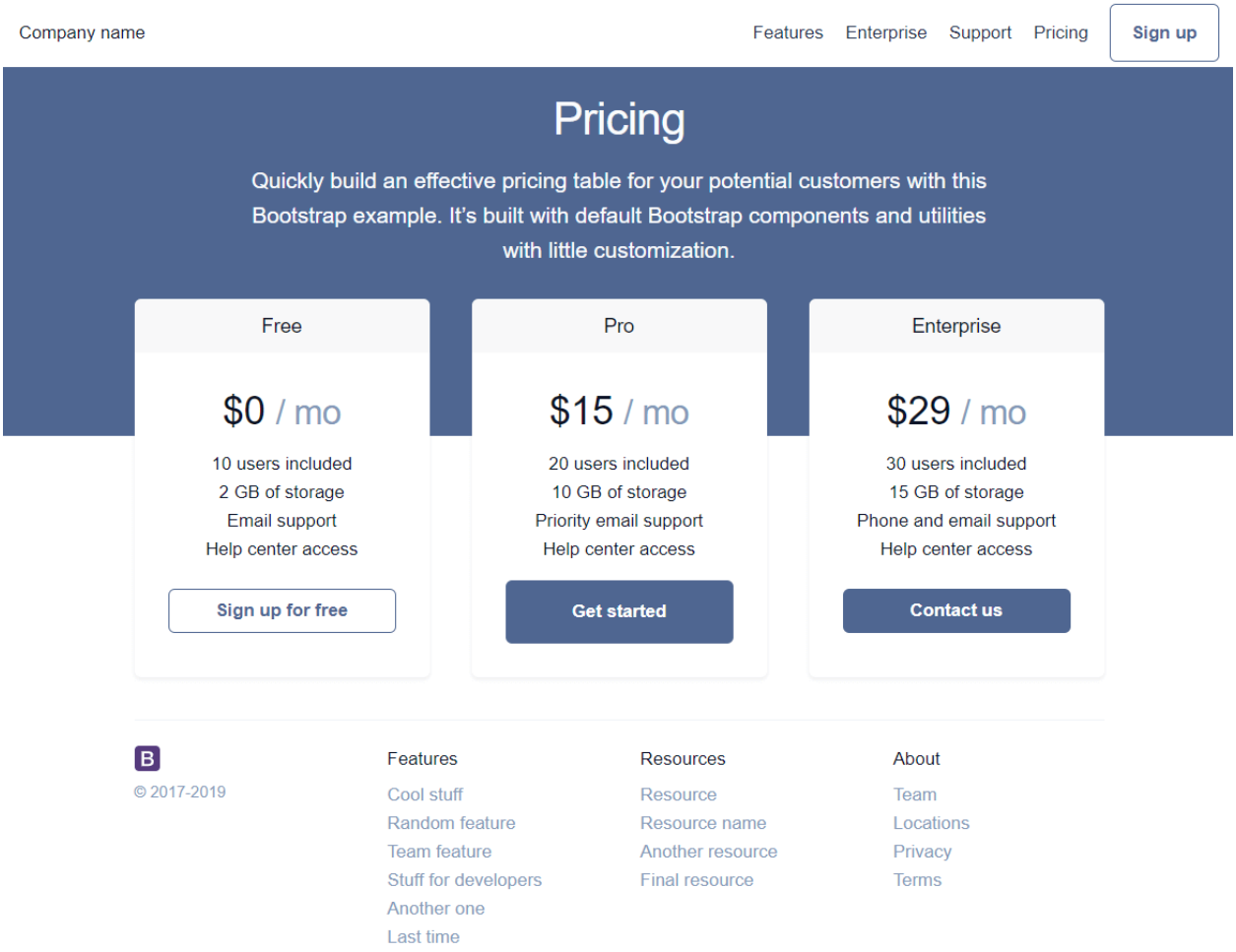
Most of the time, you *should* make a CSS class and then reference it; we're cutting corners here.

Fixing the buttons

To fix the buttons, we'll do a few things:

- change them from primary to secondary contextual colors
- Make the middle one stand out, and the other buttons smaller
- Center the buttons horizontally

This is the look that we're trying to achieve:



The pricing layout customized complete

To achieve this look, we need to replace all the instances of primary buttons with secondary buttons.

First we'll update the *Sign up* button in the navbar, from this:

```
<a class="btn btn-outline-primary" href="https://getbootstrap.com/docs/4.3/examples/\
pricing/#">Sign up</a>
```

... to this:

```
<a class="btn btn-outline-secondary" href="https://getbootstrap.com/docs/4.3/example\
s/pricing/#">Sign up</a>
```

Next we'll update the buttons in the pricing table. While we're at it, we might as well make the first button small, and give it the `mt-5` utility class so that all the buttons are seemingly vertically centered.

For the first button, with the `Sign up for free` text, we'll go from this:

```
<button type="button" class="btn btn-lg btn-block btn-outline-primary">Sign up for f\
ree</button>
```

...to this:

```
<button type="button" class="btn btn-sm mt-5 btn-block btn-outline-secondary">Sign u\
p for free</button>
```

On the second button, we'll just update the contextual color to this:

```
<button type="button" class="btn btn-lg btn-block btn-secondary">Get started</button>
```

On the third button, we'll repeat what we did with the first one. We'll go from this HTML:

```
<button type="button" class="btn btn-lg btn-block btn-primary">Contact us</button>
```

...to this:

```
<button type="button" class="btn btn-sm mt-5 btn-block btn-secondary">Contact us</bu\
tton>
```

We've added all the updates so far, and we could call it a day.

But if the background bothers you, maybe you could add a background image. In the next update, we're adding a background image and we're overlaying it with a box-shadow trick:

```
position: relative;
top: -20px;
min-height: 650px;
max-height: 350px;
background-image: url(https://images.unsplash.com/photo-1511988617509-a57c8a288659?i\
xlib=rb-1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=1951&q=80);
background-position: -20px 1040px;
box-shadow: inset 0 0 0 1000px hsla(212, 100%, 50%, 0.4)
```

In the code above, we're using an image from Helena Lopes on Unsplash[43].

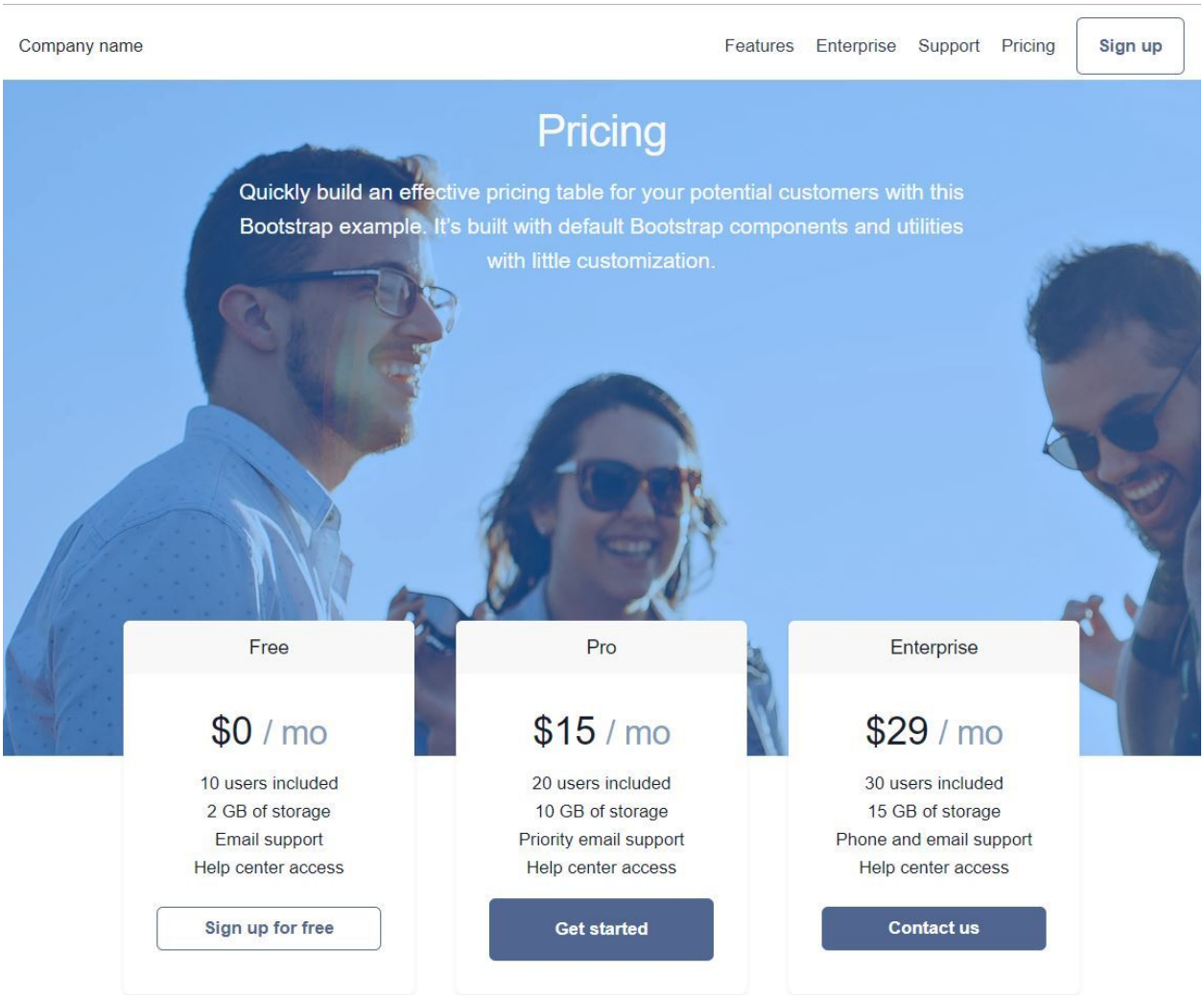
The inset `box-shadow` gives a `1000px` blur over our image, and the color of the shadow is set to a transparent `hsla(212, 100%, 50%, 0.4)` color.

Without transparency, this color would be `hsla(212, 100%, 50%, 1)`, and we could add that color to our pricing table's buttons.

[43] https://unsplash.com/photos/e3OUQGT9bWU

We're not going to do that at this point, but we'll rather leave it as is.

Our pricing layout[44] now looks like this:



The pricing layout customized finished

The completed layout can be downloaded as a zip from here[45].

Conclusion

In this article, we've seen many different techniques, tips and tricks of working with layouts. We've seen how to:

- Copy CSS from another layout into ours

[44] https://www.codingexercises.com/codelabs/2019-10-02-building-bootstrap-layouts-article-5-pt-2
[45] https://www.codingexercises.com/downloads/2019-10-02/pricing-example-layout-Bootstrap-finished.zip

- Locate and manipulate CSS in developer tools
- Unminify CSS files in the browser
- Paste CSS into devtools
- Work with the Elements panel
- Use the Command launcher and the Coverage command to check for unused CSS
- Save our layouts to our local machine
- Creatively use negative margins
- Combine background images and inset shadows to quickly give a faux background color to them

At this point, you might think that the layout needs some additional improvements before we can call it done.

However, the goal of the first ten chapter of this book is not really to show you how to build complete layouts.

Our goal at this point is to learn the work flow, the process, to acquire the techniques. We want to be able to look at a blank screen - or a very basic Bootstrap starter theme - and know two things:

1. What things we can do to update the theme
2. Know that we are able to do it - since we've already done it before

For example, in this tutorial we've seen how we can take a dull background and change it into an image, and then, even better, color that image any way we want, using the inset box-shadow trick.

Each of these simple tricks don't do much on their own, but when you combine them, you get a powerful mix, and you can come up with your own ideas easily.

For example, let's look one more time at the completed layout.

To update the background image from the blue shade to an orange one, all we need to do is update the hsla color from this:

```
box-shadow: inset 0 0 0 1000px hsla(212, 100%, 50%, 0.4);
```

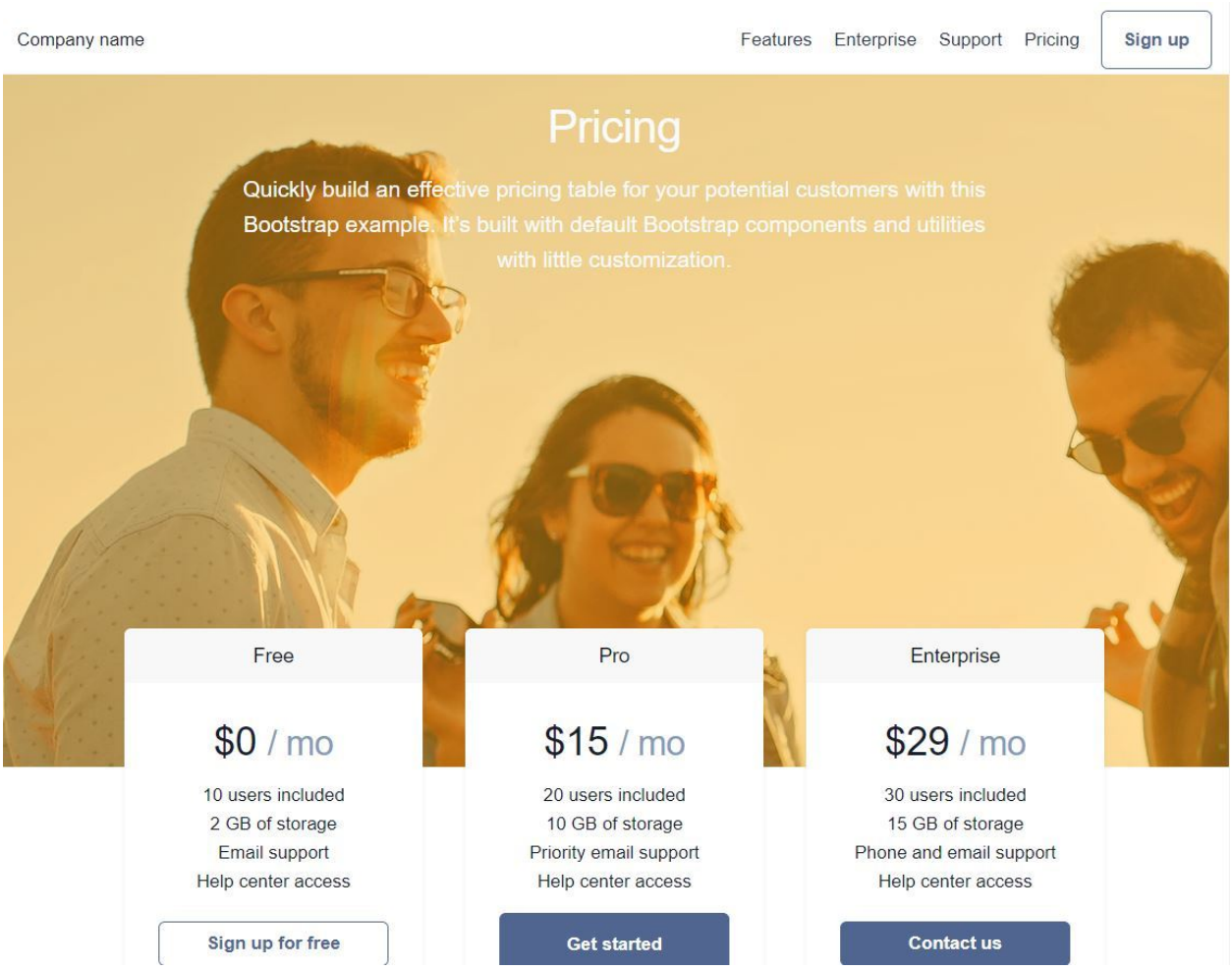
...to this:

```
box-shadow: inset 0 0 0 1000px hsla(40, 100%, 50%, 0.4);
```

Go ahead, give it a try! Just use the devtools to type the values right in.

My goal with this article series is to teach you how to work independently, like you did just now if you followed the little exercise above.

If you haven't followed the exercise, here's the updated layout's screenshot:



The pricing layout customized completed (orange shade)

That's it for this chapter. In the next one, we'll look at working with SCSS in Bootstrap 4.

Chapter 6: Building a typography-focused layout in Bootstrap 4

In this chapter, we'll look at customizing the checkout form layout with the help of CSS pre-processors.

We'll be altering the Checkout form example[46] from the Custom components section on the Bootstrap 4 website[47].

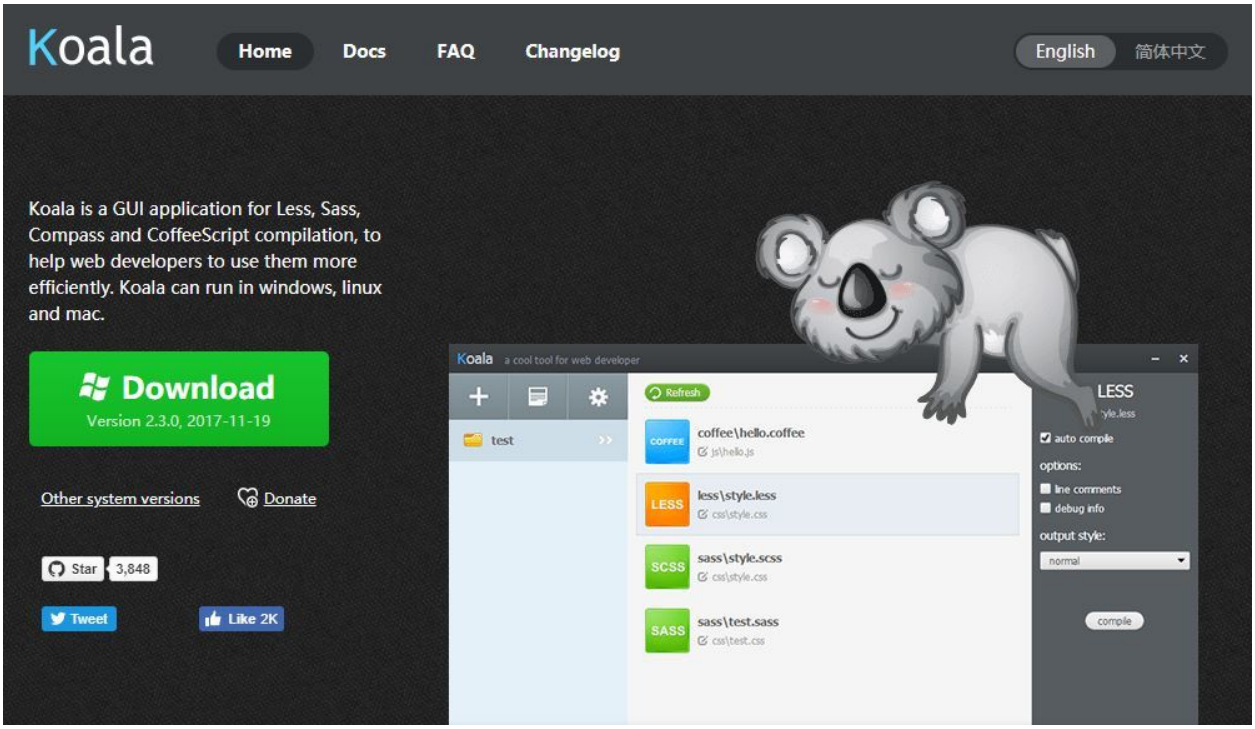
To do that, we'll use a software called Koala.

Why Koala?

Well, if you've ever tried working with SCSS, you might have found out that it might be difficult to set up, especially for beginners.

That's why I've decided to use the Koala app[48] in this chapter.

[46] https://getbootstrap.com/docs/4.3/examples/checkout/
[47] https://getbootstrap.com/docs/4.3/examples/#custom-components
[48] http://koala-app.com/



A screenshot of the Koala app homepage

Why SCSS?

The obvious question is: if it's not so easy to set up, maybe it's not meant for beginners to tinker with it?

Why use SCSS with Bootstrap 4 in the first place?

The answer is: it's an easy way to quickly change the styles on your layouts, en masse.

You'll see it in action shortly.

Installing and using Koala

To install, simply click the big green Download button, then install it just like any other program on your machine.

While Koala is installing, you might use the spare time to save the official Checkout form example layout[49] from Bootstrap 4.3 docs.

To easily save the layout, open the above link in Chrome, then follow the instructions that can be found here[50].

[49] https://getbootstrap.com/docs/4.3/examples/checkout/

[50] https://www.codingexercises.com/guides/building-bootstrap-layouts-part-5#saving-the-changes-to-our-theme-and-trimming-the-unused-css

I saved the theme into a new folder called checkout-example.

Here's the structure of the folder, after I save the theme:

```
checkout-example/
  ├── Checkout example · Bootstrap_files/
  │   ├── bootstrap.bundle.min.js.download
  │   ├── bootstrap.min.css
  │   ├── bootstrap-solid.svg
  │   ├── form-validation.css
  │   ├── form-validation.js.download
  │   └── jquery-3.3.1.slim.min.js.download
  └── Checkout example · Bootstrap.html
```

The structure of the folder after default theme is saved

As we can see, there's no SCSS in sight. We'll need to fix that next.

To use this folder as a starting point, you can download it here[51].

Adding Bootstrap 4.3 SCSS files

To start, download the source files from getbootstrap.com[52].

Once unzipped, the structure of these source files looks as follows:

[51] https://www.codingexercises.com/downloads/checkout-example.zip
[52] https://getbootstrap.com/docs/4.3/getting-started/download/#source-files

```
bootstrap/
├── dist/
│   ├── css/
│   └── js/
├── site/
│   └──docs/
│      └── 4.3/
│          └── examples/
├── js/
└── scss/
```

The structure of the bootstrap source files

To read up a bit more on the above file structure, have a look at the Bootstrap source code section[53] on the official website.

What we will do with the above download is, we'll copy only the `scss` folder. Then we'll paste that folder into our `checkout-example` folder, so that the updated structure looks like this:

[53] https://getbootstrap.com/docs/4.3/getting-started/contents/#bootstrap-source-code

```
checkout-example/
  ├── Checkout example · Bootstrap_files/
  |   ├── scss/
  |   |     ├── mixins/
  |   |     ├── utilites/
  |   |     ├── vendor/
  |   |     ├── _alert.scss
  |   |     ├── _badge.scss
  |   |     ├── _breadcrumb.scss
  |   |     └── ...etc...
  |   ├── bootstrap.bundle.min.js.download
  |   ├── bootstrap.min.css
  |   ├── bootstrap-solid.svg
  |   ├── form-validation.css
  |   ├── form-validation.js.download
  |   └── jquery-3.3.1.slim.min.js.download
  └── Checkout example · Bootstrap.html
```

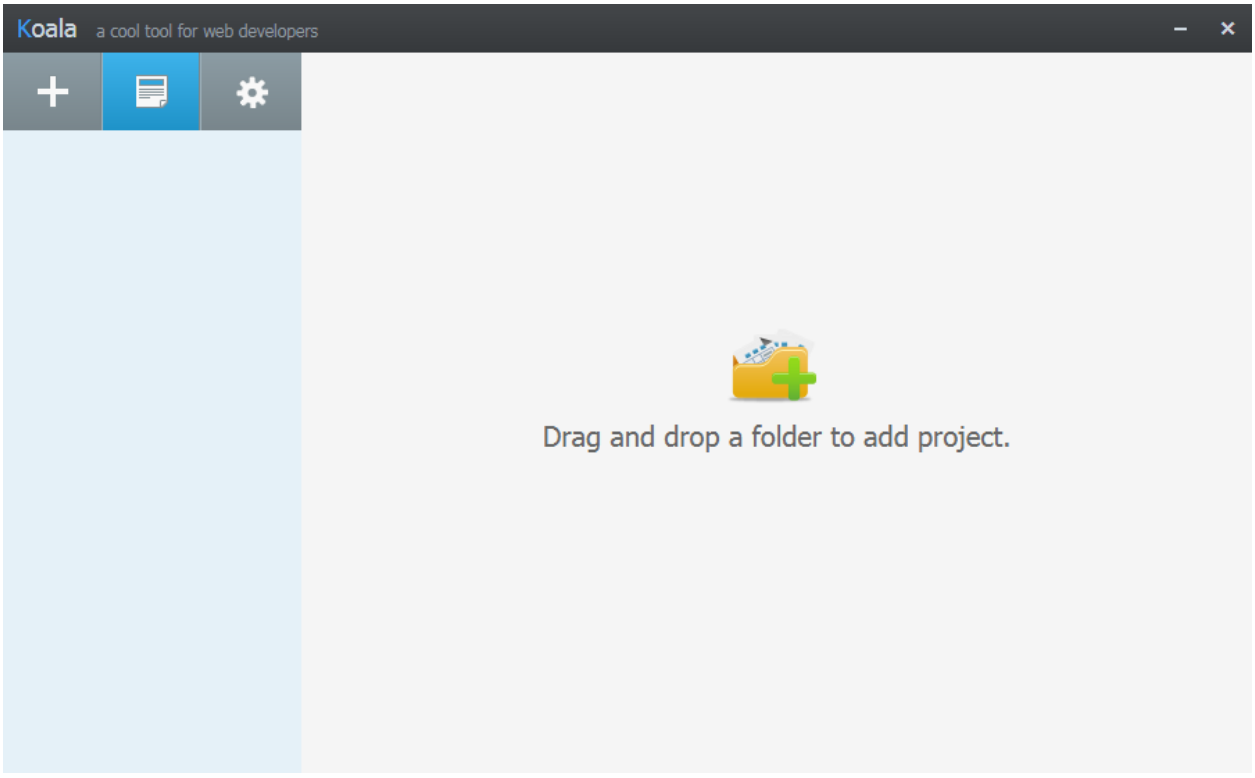
The updated folder structure once we impored the scss folder into the checkout-example folder

Now that we've pasted in the SCSS folder, we need to see how Koala will compile it.

Compiling SCSS with the Koala app

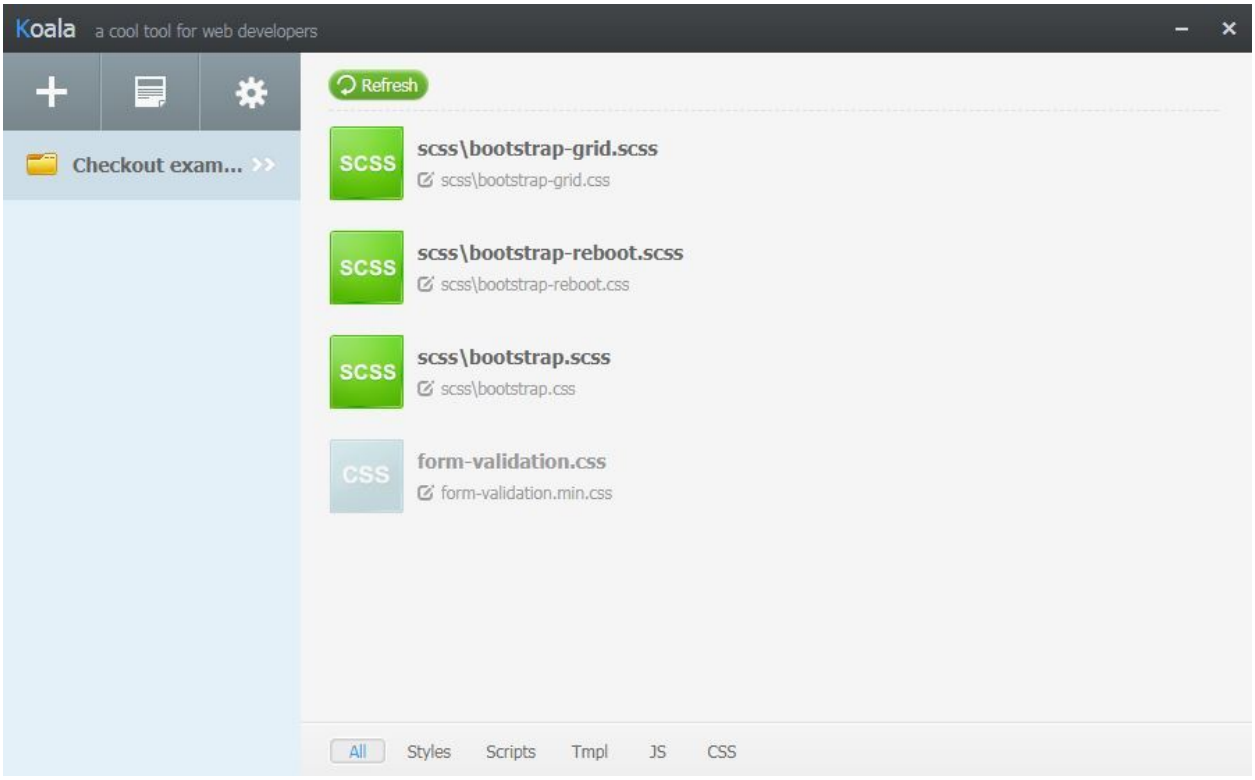
We'll compile all our SCSS files from the scss folder into the `bootstrap.min.css` file, with the help of the Koala app.

To start, open the Koala app.



The Koala starting window open

Next, press the plus sign and locate the SCSS folder:



The Koala starting window open

Now we'll need to update the `_variables.scss` file.

This file is known as a partial file: the _ character in front of the file name is a convention in SCSS that denotes that we're in fact dealing with a partial file.

When you import partial SCSS files, you don't have to write the _ or the `.scss` extension:

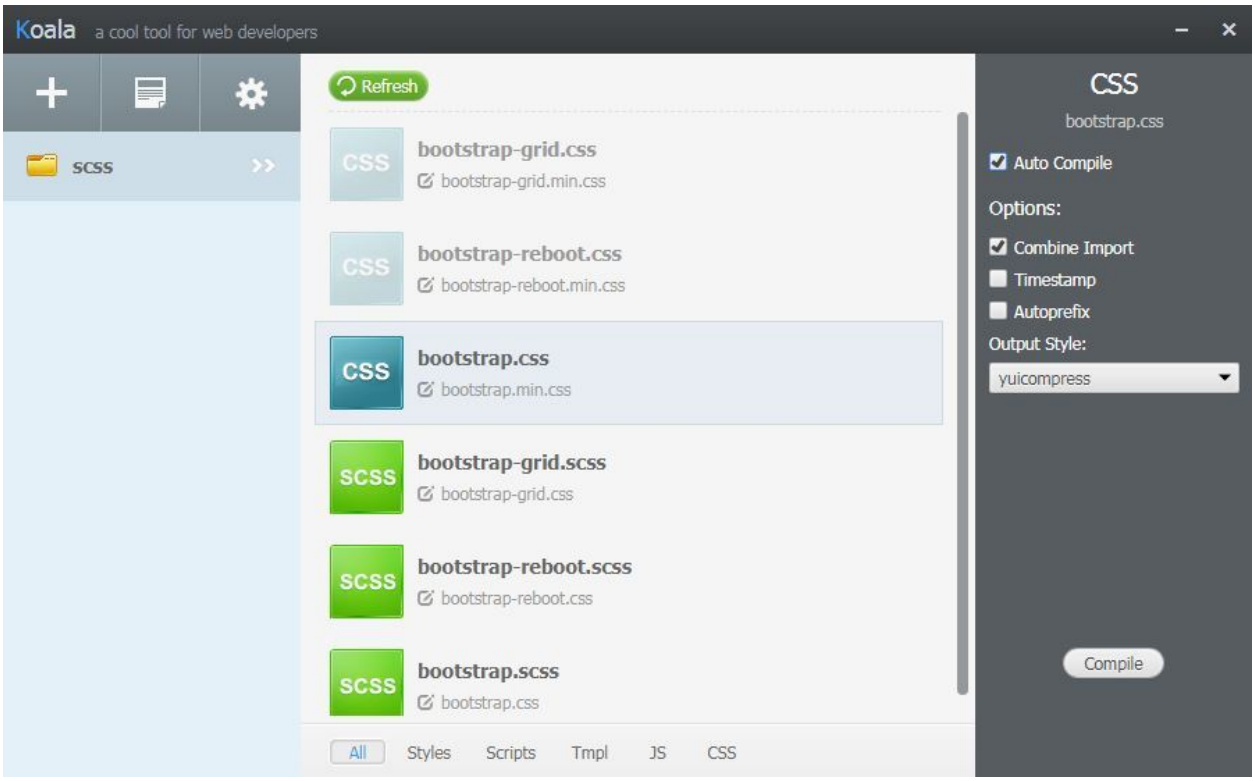
```
/*!
 * Bootstrap v4.3.1 (https://getbootstrap.com/)
 * Copyright 2011-2019 The Bootstrap Authors
 * Copyright 2011-2019 Twitter, Inc.
 * Licensed under MIT (https://github.com/twbs/bootstrap/blob/master/LICENSE)
 */

@import "functions";
@import "variables";
@import "mixins";
//...etc (code below skipped for brevity)
```

Now that we've saved everything, let's open the Koala app and hit the Refresh button.

You'll see three new files, grayed out.

They're grayed out because we still need to choose how to compile them.

To make that choice, just click on a file, and a sidebar will open. Click the auto-compile option, so that you don't have to do this again. It will compile automatically from now on.



The Koala starting window open

We'll repeat the same thing for the other two grayed-out files: `bootstrap-reboot.css`, and `bootstrap-grid.css`.

Now let's open the `_custom_variables.scss` file and make some changes.

Changing default Bootstrap variables

When you open the `_custom_variables.scss` file, you'll see that there are a bunch of variables inside it, used for different settings.

Let's take the easy route and update the `$border-radius` variable to zero, like this:

```
1  //$border-radius: .25rem !default;
2  $border-radius: 0 !default;
```

As we can see, we've first commented out the original variable, and then we updated the variable with the custom value on the line below. That way we can easily get back to the original variable if there is a need to do so.

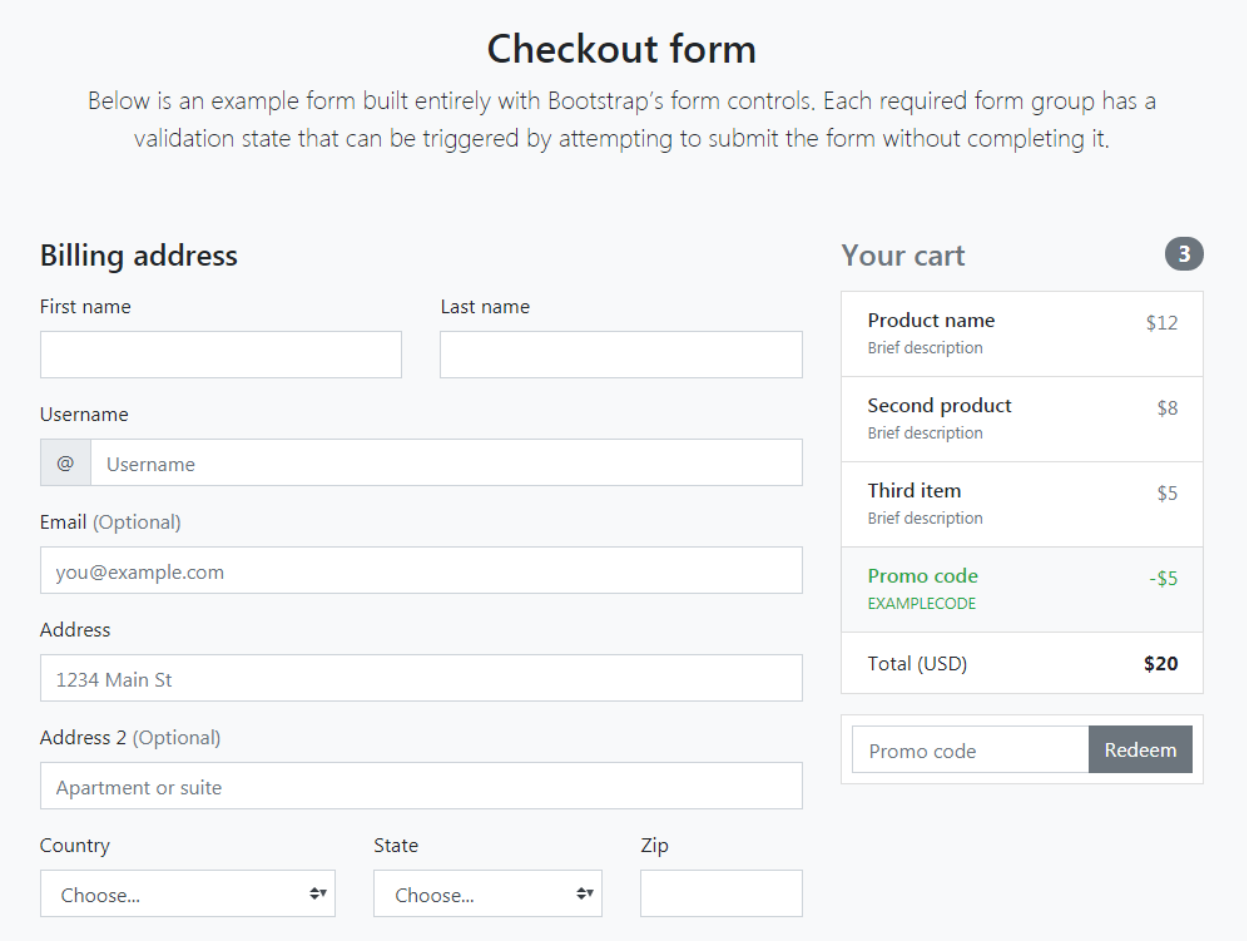
Now we need to press compile on any single button, and all three specified CSS files will be compiled.

What we're expecting from this update to the variable is that our Checkout example layout's form input borders will no longer be rounded. Instead, they should be squared.

All we have to do now is reference these files in our template's HTML file, on line 14:

```
<link href="./CheckoutexampleBootstrap_files/scss/bootstrap.min.css" rel="stylesheet\
">
```

If you've done everything right, you should see the updated Checkout example layout:



Square borders on inputs

Hopefully you can appreciate how easy it was to change some global, site-wide styles this way.

Also, this technique allows us to get rid of redundant CSS even before it compiles: We can simply comment-out the unused SCSS partials.

For example, if we know for a fact that we will not be using, for example, ***the alert component***, we can simply comment out its import from the `bootstrap.scss` file:

```
// ...code skipped for brevity
@import "badge";
@import "jumbotron";
//@import "alert";
@import "progress";
@import "media";
@import "list-group";
// ...code skipped for brevity
```

To conclude this section: once you've set up SCSS on your machine - which should be easy with apps like Koala - it becomes a breeze to do two important things:

- trim your unused CSS
- easily make global changes to your layouts

This powerful approach to building Bootstrap layouts allows us to further speed up our layout customization.

The official checkout form layout we have updated with SCSS is available as a zip here[54].

Next, let's build that typography-focused layout; we'll begin by downloading Google fonts to use in our layout.

Downloading Google fonts

For this layout, we'll be using 2 fonts: *Roboto* and *Roboto Slab*. Both fonts are available at the Google fonts website[55].

Once you've found the font you like, to the right of the font name you have a little plus icon in a red circle. If you click it, the font you clicked will be added to the list of font families selected, showing in a toggleable bar at the bottom of the screen.

I've clicked the plus sign on Roboto and Roboto Slab, so there's 2 Families Selected in the screenshot above.

We've got two options of embedding fonts, the *standard way* and the *@import*.

To import your fonts with the standard syntax, use this HTML code:

```
<link href="https://fonts.googleapis.com/css?family=Roboto|Roboto+Slab&display=swap"\
 rel="stylesheet">
```

You can also use the `@import` from within the stylesheet:

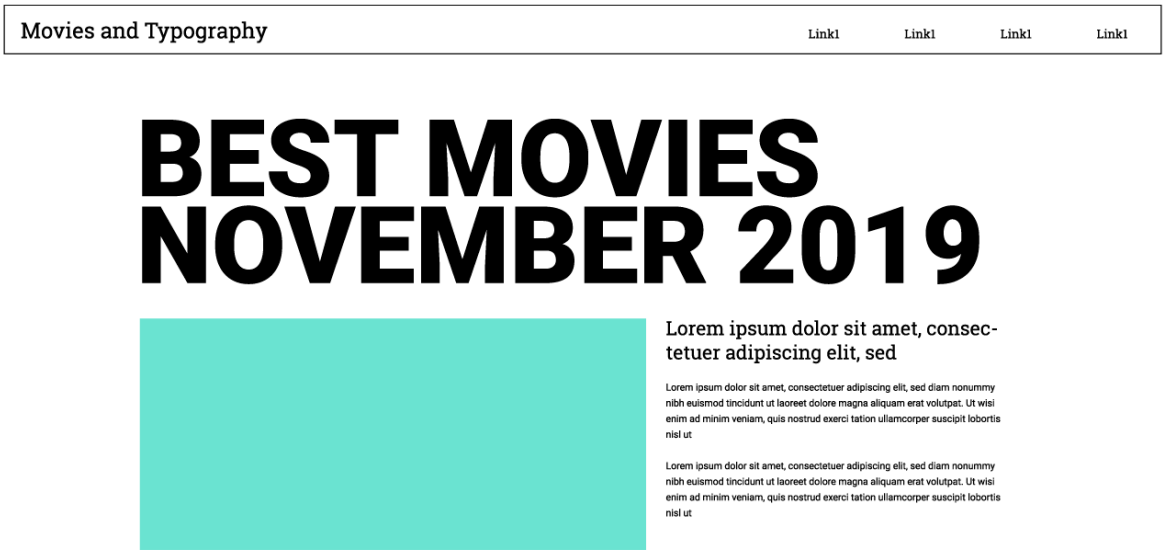
[54] https://www.codingexercises.com/downloads/2019-10-03/checkout-example-v2.zip
[55] https://fonts.google.com/

```
<style>
@import url('https://fonts.googleapis.com/css?family=Roboto|Roboto+Slab&display=swap\
');
</style>
```

In this example, we'll use the standard approach.

The mockup and the starter layout

Next, let's have a look at a mockup for the layout we'll be building.



Mockup of the typography-focused layout

Looking at the above mockup, we can notice a few things:

- the layout is supposed to have a lot of whitespace
- the headings in this layout are supposed to be huge
- We'll mix and match two fonts: the sans-serif (Roboto) and the serif (Roboto-slab)

To speed up our development even further, we'll be using the starter template from getbootstrap.com[56].

Open a new HTML file in your computer, and paste in the code you copy-pasted from the link above:

[56] https://getbootstrap.com/docs/4.3/getting-started/introduction/#starter-template

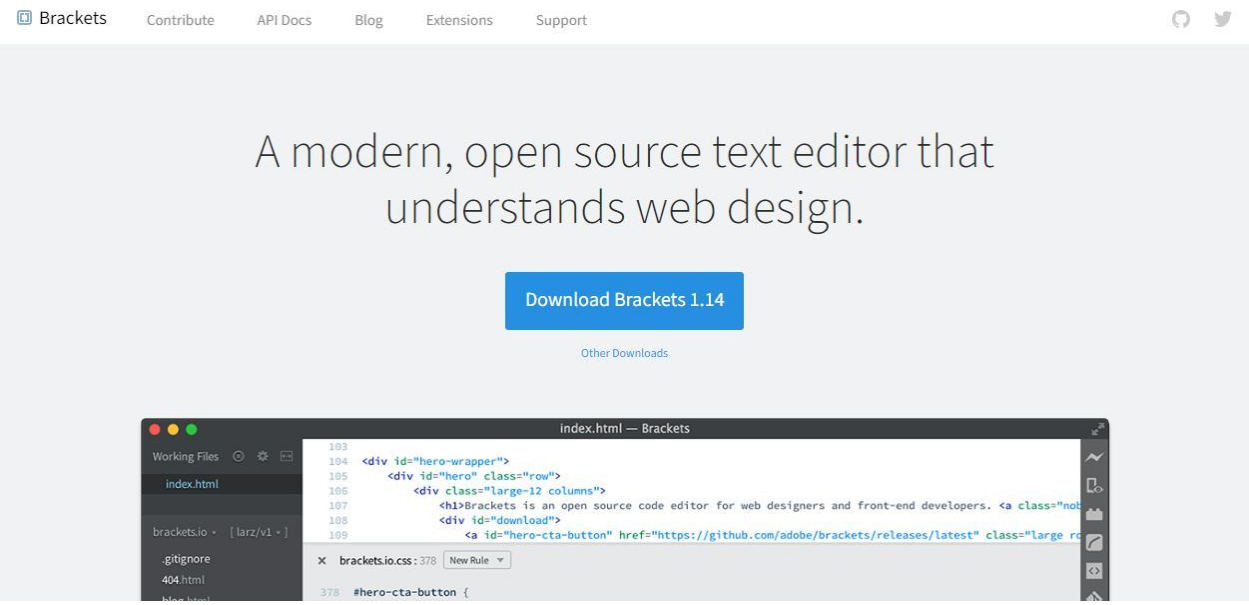
```
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta
        charset="utf-8">
    <meta
        name="viewport"
        content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link
        rel="stylesheet"
        href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.c\
ss"
        integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT\
2MZw1T"
        crossorigin="anonymous">

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

    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
        integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1P\
i6jizo"
        crossorigin="anonymous"></script>
    <script
        src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.\
js"
        integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIH\
NDz0W1"
        crossorigin="anonymous"></script>
    <script
        src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"
        integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+\
B07jRM"
        crossorigin="anonymous"></script>
  </body>
</html>
```

Save the file in a new folder. We'll call the new folder `typography-layout`, and the file name will just be `index.html`.

At this point of our book, we'll finally start using a code editor.

Choosing the code editor

We have many different choices of editors, but we'll go with Brackets[57], mostly because it's very beginner-friendly.



A screenshot of the Brackets' editor's homepage

Once you've downloaded the Brackets editor, install it using the regular installation process of your operating system.

After the installation is complete, open the Brackets window, and you'll see this:

[57] http://brackets.io/

The live preview icon in the brackets code editor

One of the best features of Brackets is this little *live preview* icon (highlighted in a magenta circle in the above image). To make your website reload every time you save a change in your layout, just turn on the live preview button.

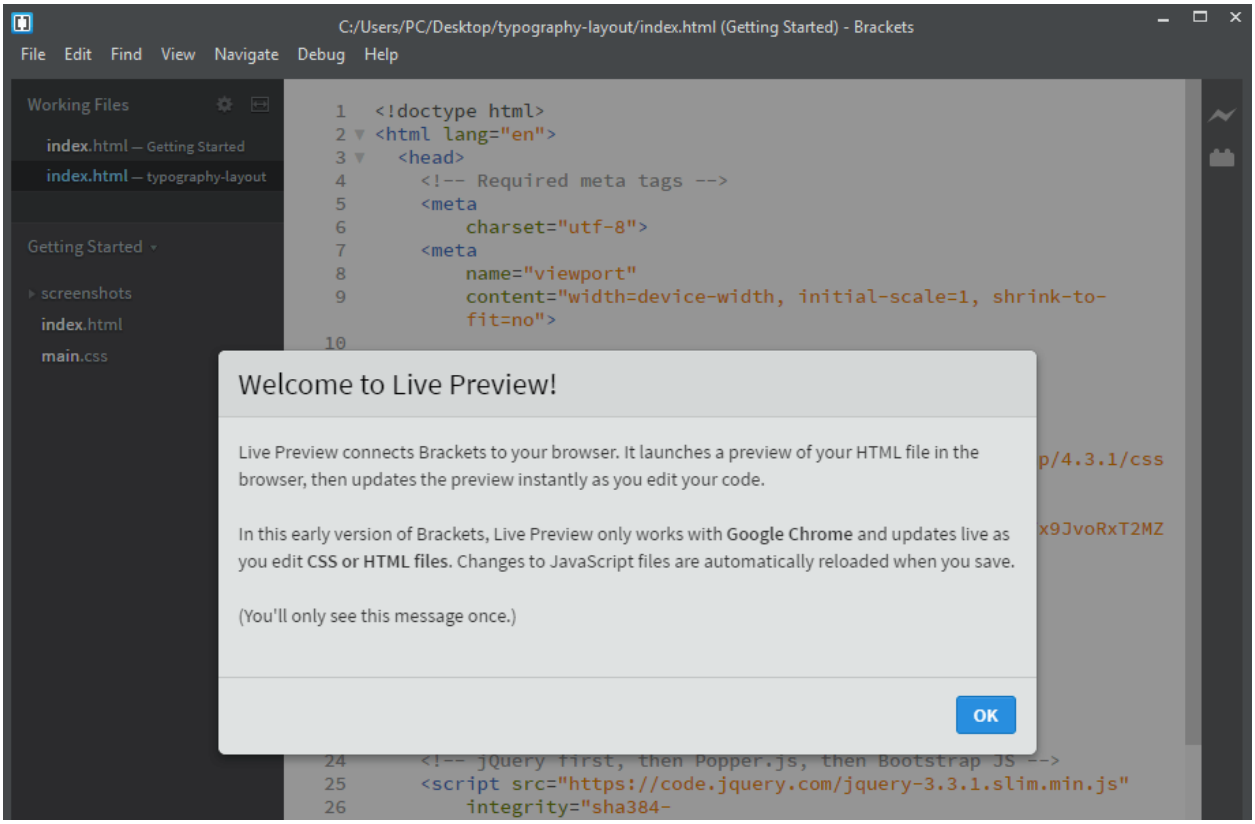
Alright, we've finished all the prep work! Now let's start building the layout.

Opening the starter template

Once you've downloaded brackets, open the `index.html` you've copy-pasted earlier.

If your Brackets window is already opened, you can just drag and drop the `index.html` file into the editor.

Next, click the live preview button, and you'll see the following message pop-up:

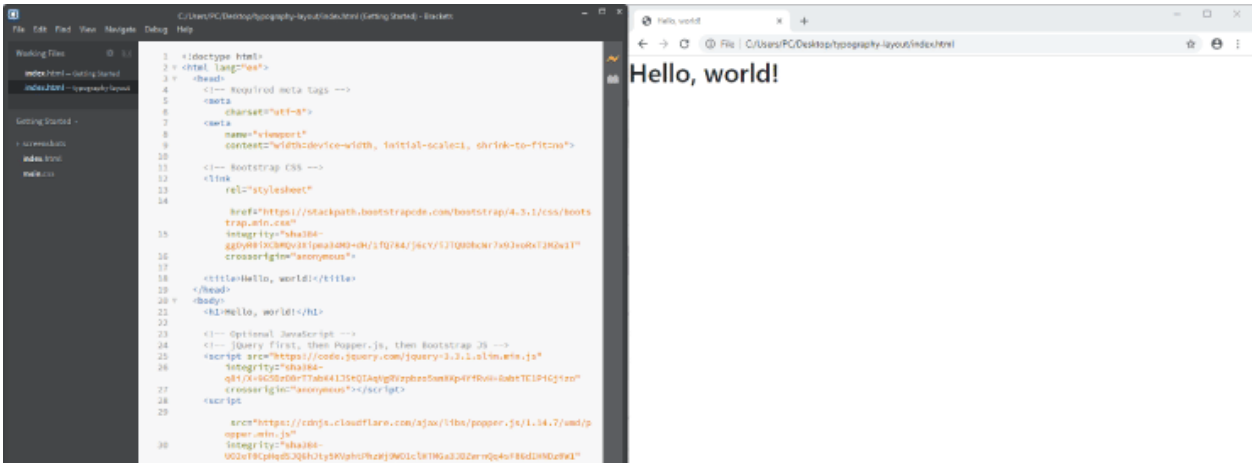


Brackets' Live preview notification

This one-time notification just says that it will launch and refresh your Chrome browser on every save of `index.html`.

Once you click ok, you'll have to wait a couple of seconds, and you'll see a new Chrome window launching.

Here's a split-screen view of this happening:



Brackets' Live preview launching our Bootstrap 4 starter template

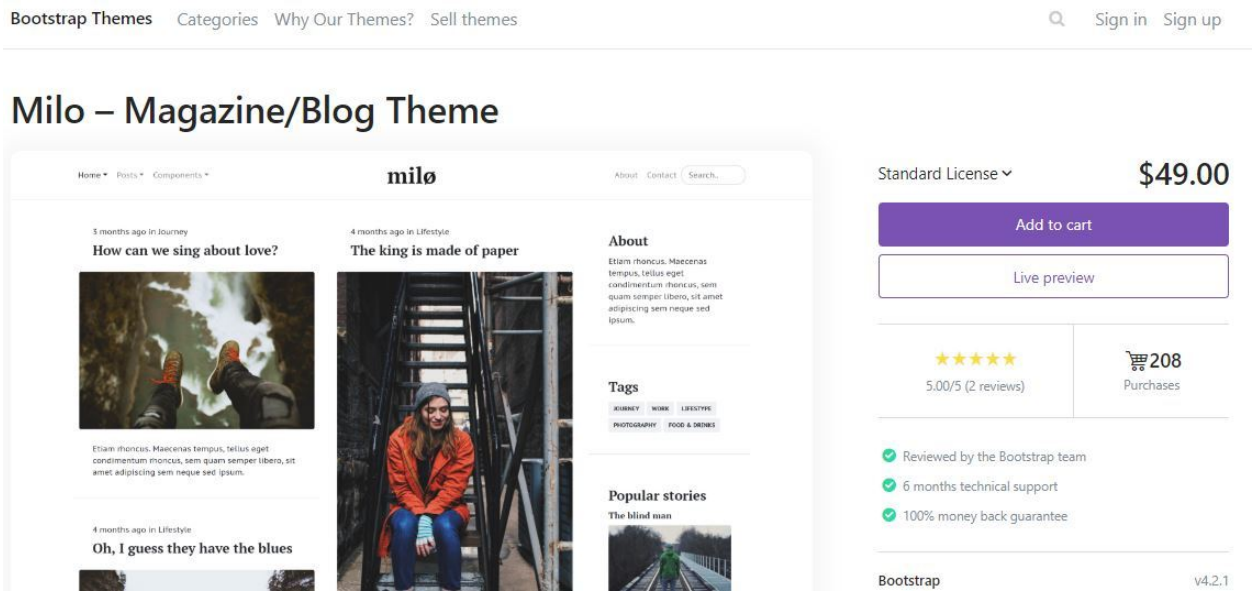
Now we are ready to add the navbar.

Adding the navbar

To add the navbar to our layout, we first need to pick one.

This time, we'll go to the themes section on the getbootstrap.com website[58].

Browsing through the themes, I've found one with an interesting navbar. The theme's called Milo[59].

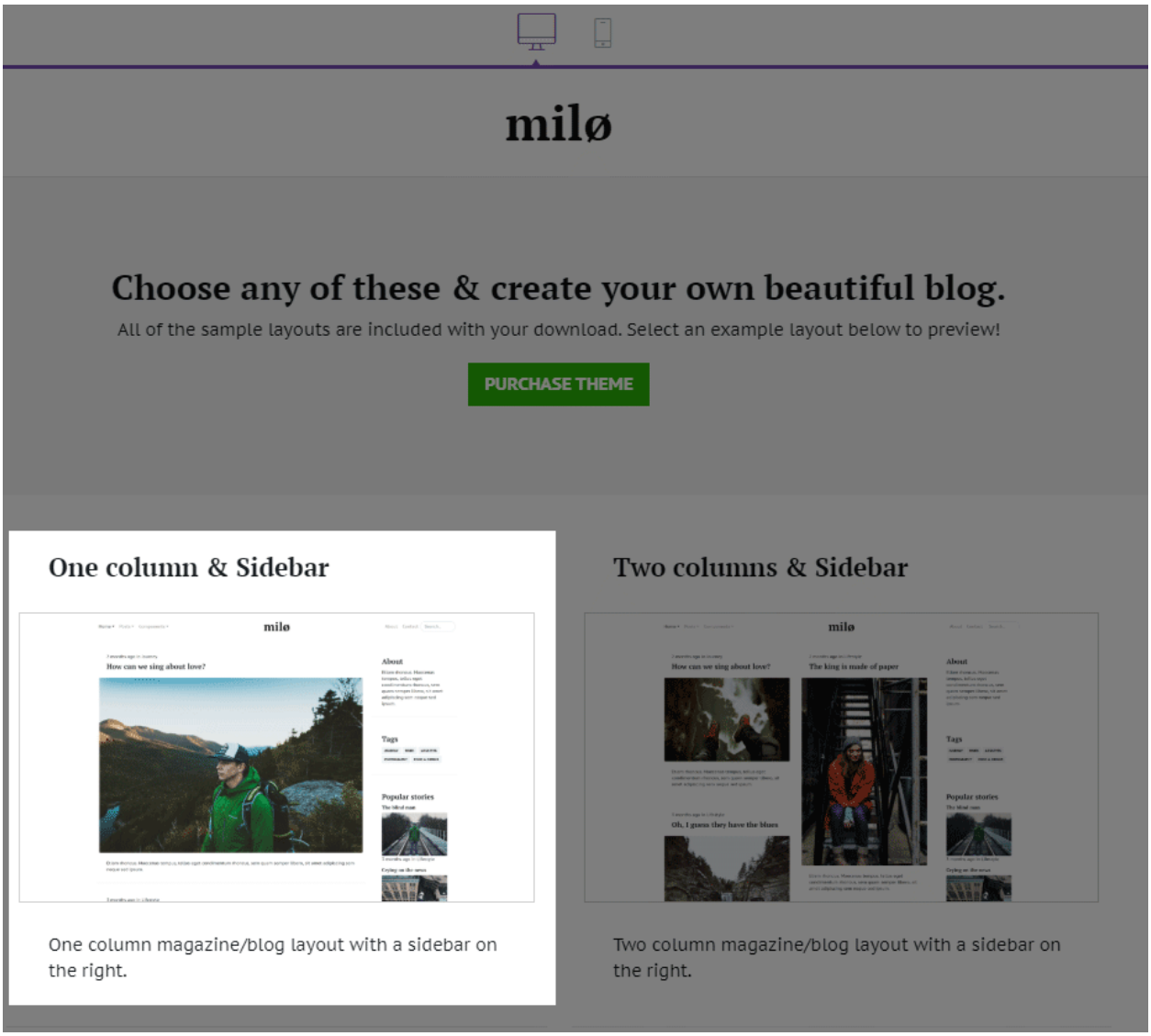


Milo Bootstrap theme preview

[58] https://themes.getbootstrap.com/
[59] https://themes.getbootstrap.com/product/milo-magazineblog-theme/

Click on the live preview button for the theme, and a new window will open.

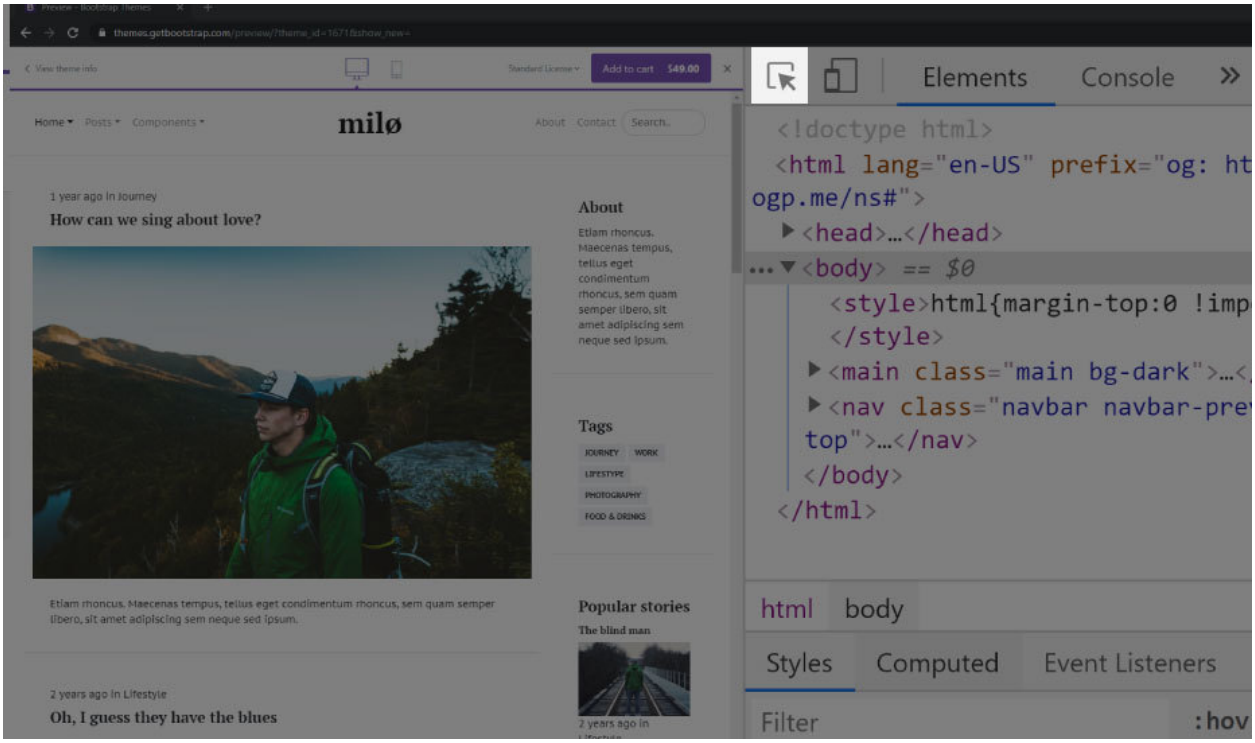
Next, click the ***One column & Sidebar*** theme preview.



Choosing Milo theme's One column layout

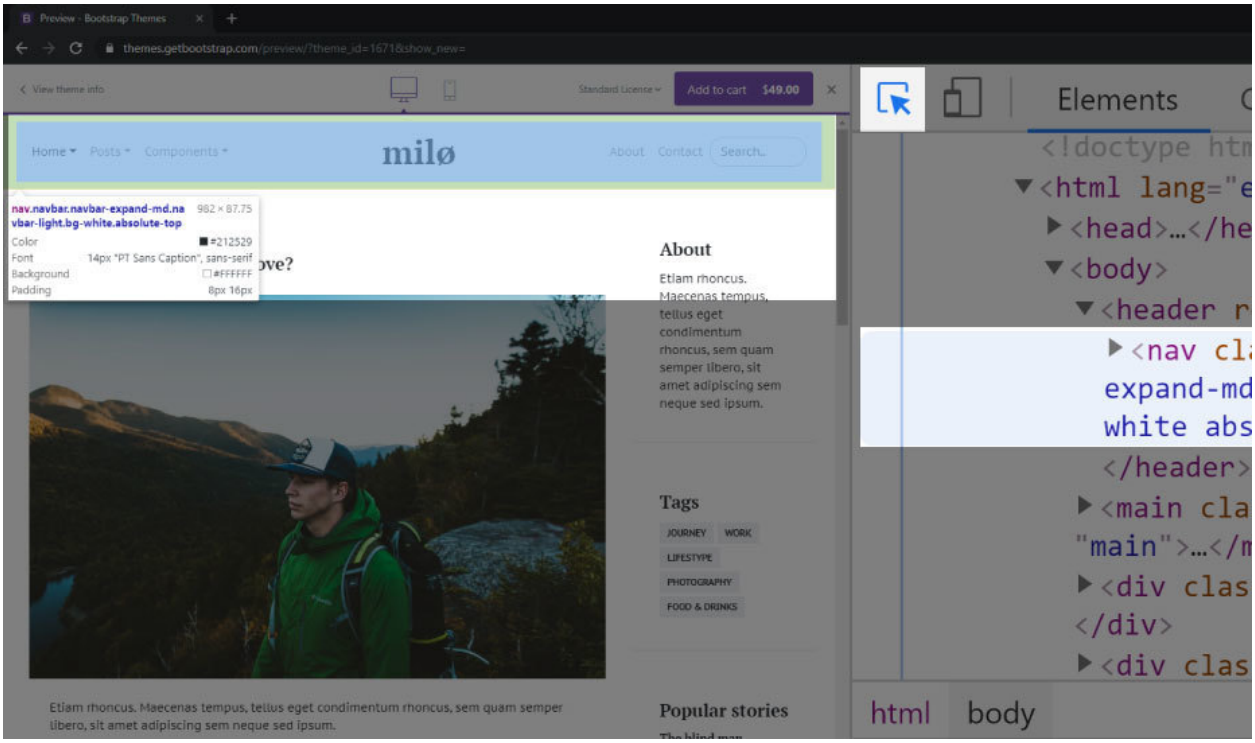
Once the theme is open, right-click on the navbar and click `Inspect`.

Alternatively, just press F12 and the developer tools will open. Now, you need to click the icon that shows a rounded square and an arrow over it:



Select an element in the page to inspect it

Now with the tool active, just hover over sections on the page until you find the navbar.

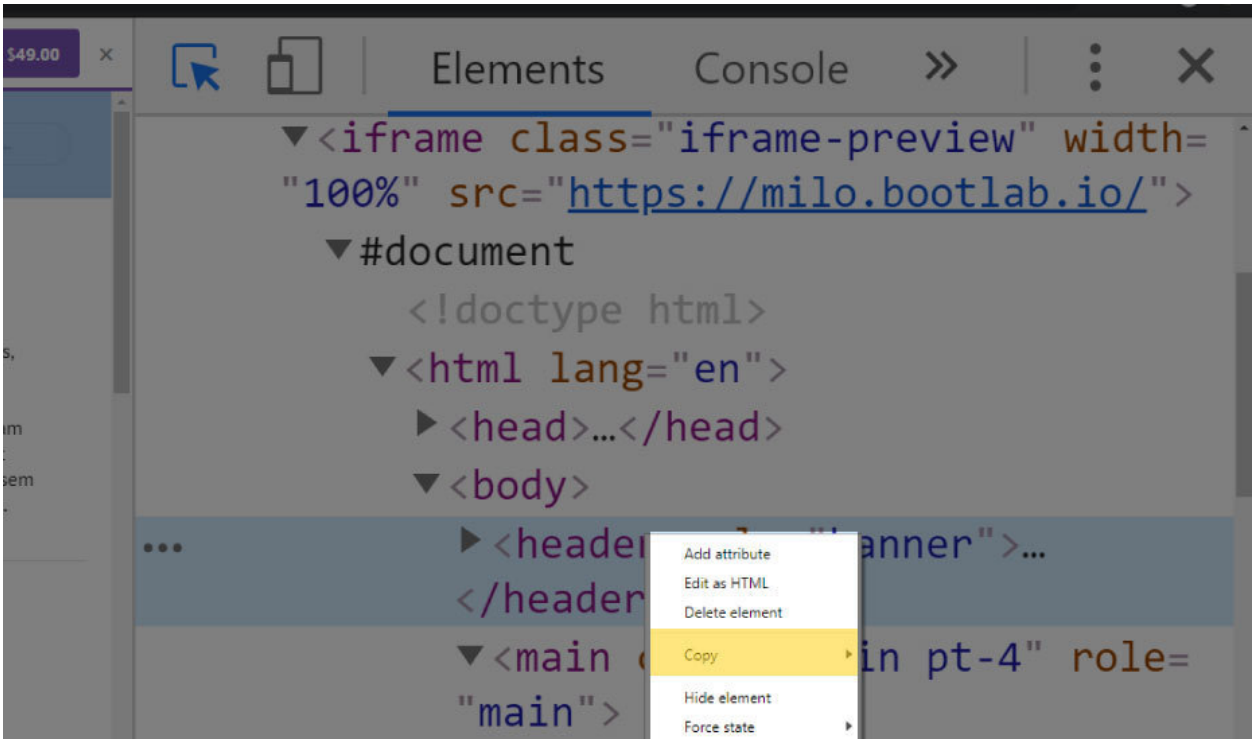


Copying an element from the milo theme, part 1

The above image shows the navbar in focus, highlighted both in the rendered web page and in the code in the Elements panel.

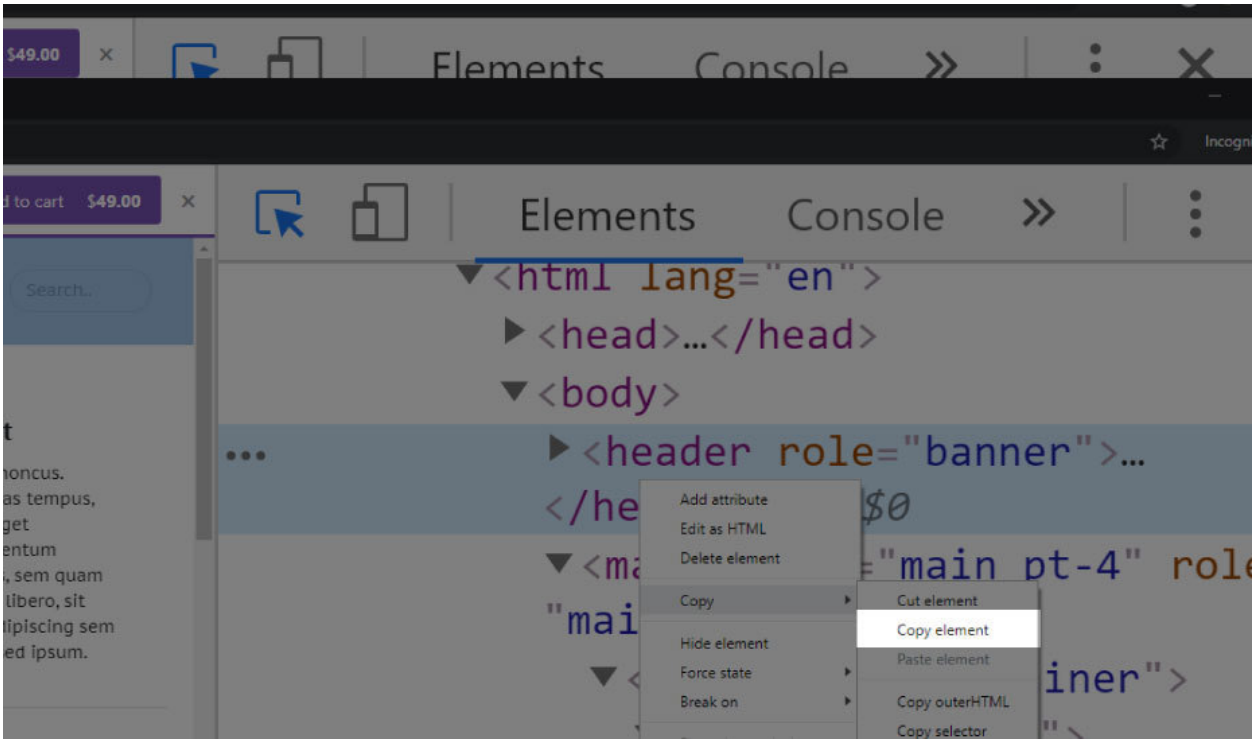
Now it's just a matter of copying the HTML code, together with the wrapping element for the navbar. In this case, that element is the `header` element, located right under the opening `<body>` tag.

To copy an element, right-click it. This will open a right-click contextual menu, and now you can click the `Copy` command (highlighted in yellow in the image below):



Right-click on an element in the devtools to copy it

Now click the Copy element command from the sub-contextual menu.



Click the copy element from the sub-contextual menu

Here's the copied HTML:

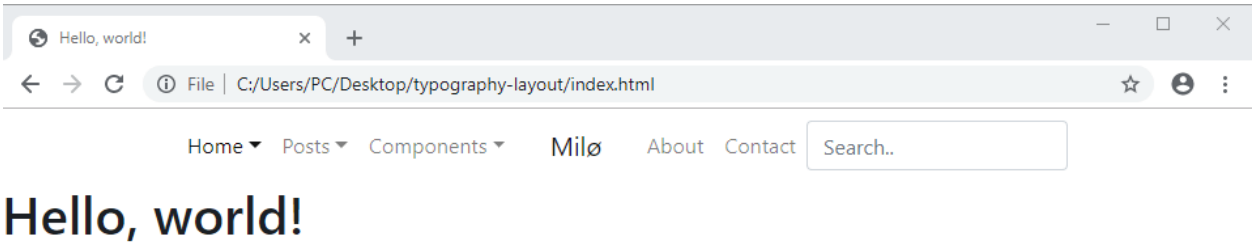
```
<header role="banner">
    <nav class="navbar navbar-expand-md navbar-light bg-white absolute-top">
      <div class="container">

        <button class="navbar-toggler order-2 order-md-1" type="button" data-toggle="\
"collapse" data-target="#navbar" aria-controls="navbar" aria-expanded="false" aria-l\
abel="Toggle navigation">
          <span class="navbar-toggler-icon"></span>
        </button>

        <div class="collapse navbar-collapse order-3 order-md-2" id="navbar">
          <ul class="navbar-nav mr-auto">
            <li class="nav-item dropdown active">
              <a class="nav-link dropdown-toggle" href="#" id="dropdown02" data-togg\
le="dropdown" aria-haspopup="true" aria-expanded="false">Home</a>
              <div class="dropdown-menu" aria-labelledby="dropdown02">
                <a class="dropdown-item" href="home-onecolumn.html">One column</a>
                <a class="dropdown-item" href="home-twocolumn.html">Two column</a>
                <a class="dropdown-item" href="home-threecolumn.html">Three column</\
a>
```

```
                <a class="dropdown-item" href="home-fourcolumn.html">Four column</a>
                <a class="dropdown-item" href="home-featured.html">Featured posts</a>
                <a class="dropdown-item" href="home-fullwidth.html">Full width</a>
              </div>
            </li>
            <li class="nav-item dropdown">
              <a class="nav-link dropdown-toggle" href="#" id="dropdown02" data-togg\
le="dropdown" aria-haspopup="true" aria-expanded="false">Posts</a>
              <div class="dropdown-menu" aria-labelledby="dropdown02">
                <a class="dropdown-item" href="post-image.html">Image</a>
                <a class="dropdown-item" href="post-video.html">Video</a>
                <a class="dropdown-item" href="post-new.html">New story</a>
              </div>
            </li>
            <li class="nav-item dropdown">
              <a class="nav-link dropdown-toggle" href="#" id="dropdown03" data-togg\
le="dropdown" aria-haspopup="true" aria-expanded="false">Components</a>
              <div class="dropdown-menu" aria-labelledby="dropdown03">
                <a class="dropdown-item" href="doc-typography.html">Typography</a>
                <a class="dropdown-item" href="doc-buttons.html">Buttons</a>
                <a class="dropdown-item" href="doc-tables.html">Tables</a>
                <a class="dropdown-item" href="doc-forms.html">Forms</a>
                <a class="dropdown-item" href="doc-cards.html">Cards</a>
              </div>
            </li>
          </ul>
        </div>

        <a class="navbar-brand mx-auto order-1 order-md-3" href="index.html">Milø</a>

        <div class="collapse navbar-collapse order-4 order-md-4" id="navbar">
          <ul class="navbar-nav ml-auto">
            <li class="nav-item">
              <a class="nav-link" href="page-about.html">About</a>
            </li>
            <li class="nav-item">
              <a class="nav-link" href="page-contact.html">Contact</a>
            </li>
          </ul>
          <form class="form-inline" role="search">
            <input class="search js-search form-control form-control-rounded mr-sm-2\
" type="text" title="Enter search query here.." placeholder="Search.." aria-label="S\
earch">
```

```
64            </form>
65          </div>
66        </div>
67      </nav>
68    </header>
```

Now you can paste in the copied code into Brackets, right under the opening `<body>` tag.

Once you save the update, the Brackets' Live Preview will update the page to this:

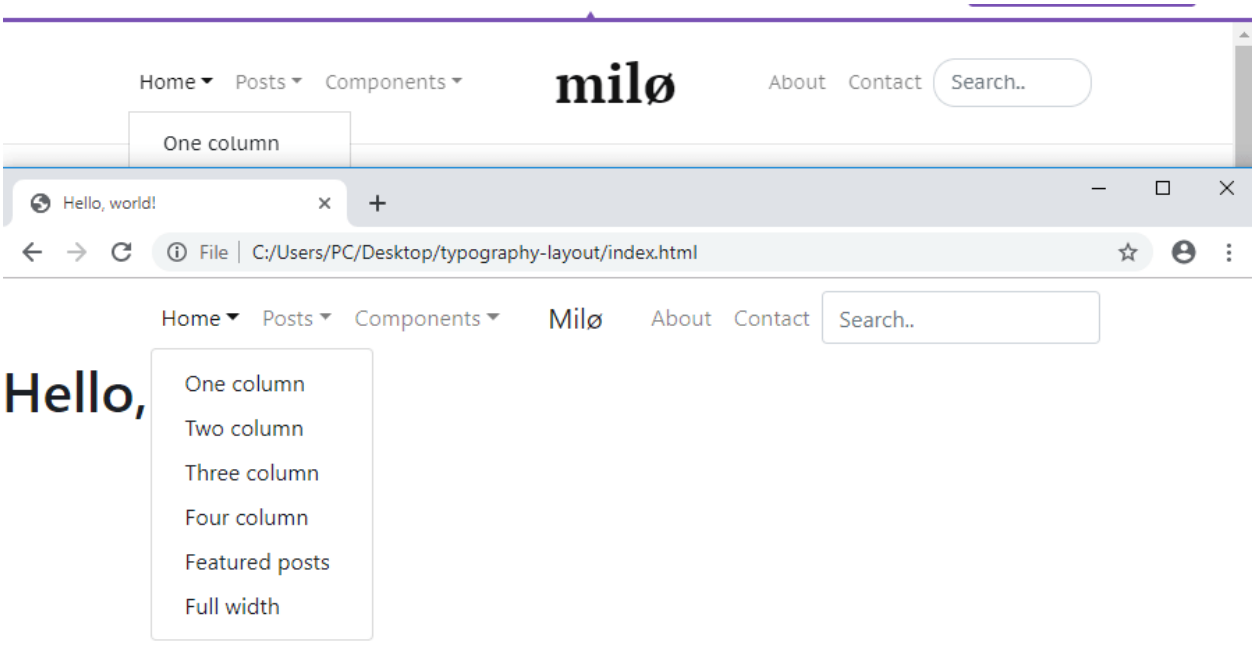


The updated layout with the copied header

Next, let's update the CSS.

Visually comparing the navbars

Before we make changes to our new navbar's CSS, let's visually compare the two navbars.



Visually comparing the two navbars

With this quick comparison, we see there are just 2 main differences between the two navbars:

1. The dropdowns' borders are not rounded in the source theme
2. The search input's borders are rounded
3. There's a bottom border on the nav element in the source theme

To fix this quickly, we'll add a `style` tag right above the closing `</head>` tag in our `index.html` file and update it as follows:

```
<!-- code skipped for brevity -->
  <style>
    .dropdown-menu, .form-control {
      border-radius: 0
    }
  </style>
</head>
```

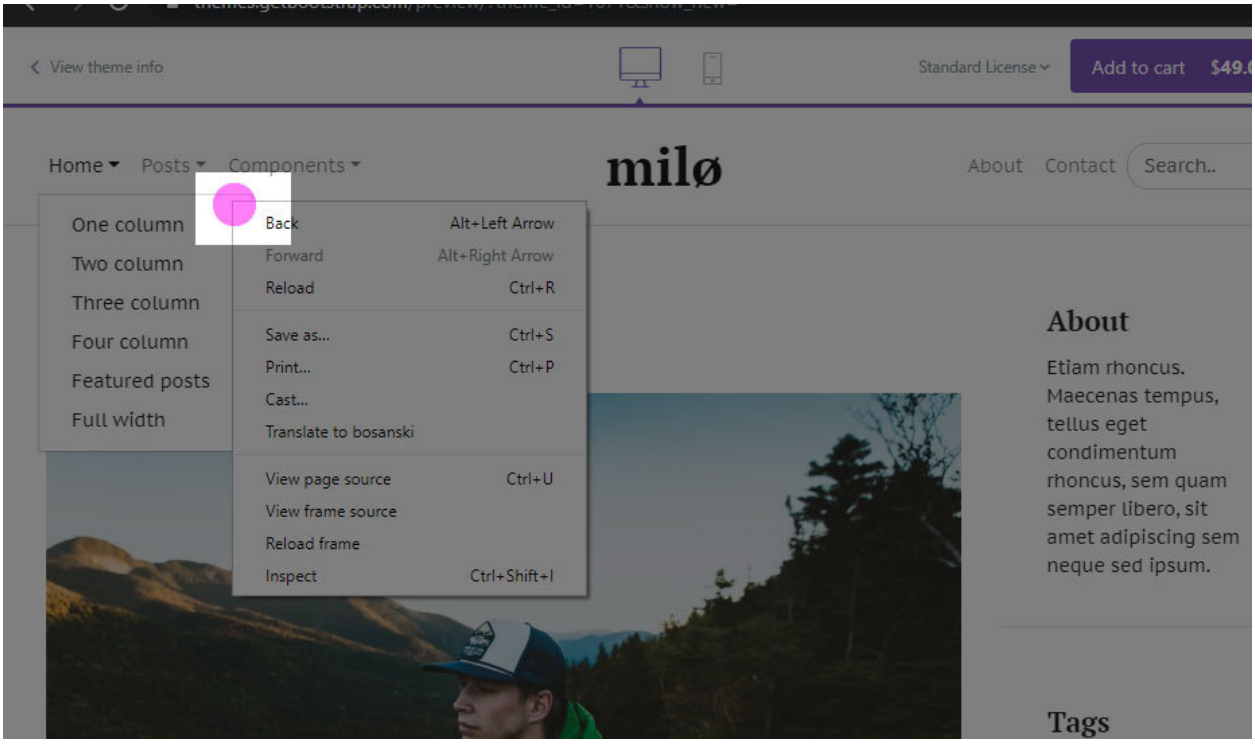
We've settled for an easy solution: there's no rounded borders on either the dropdown menu or the search input.

But, how did we find these styles?

Finding the styles to update using developer tools

To find the styles that need updating, we'll use a similar technique that we used to copy HTML.

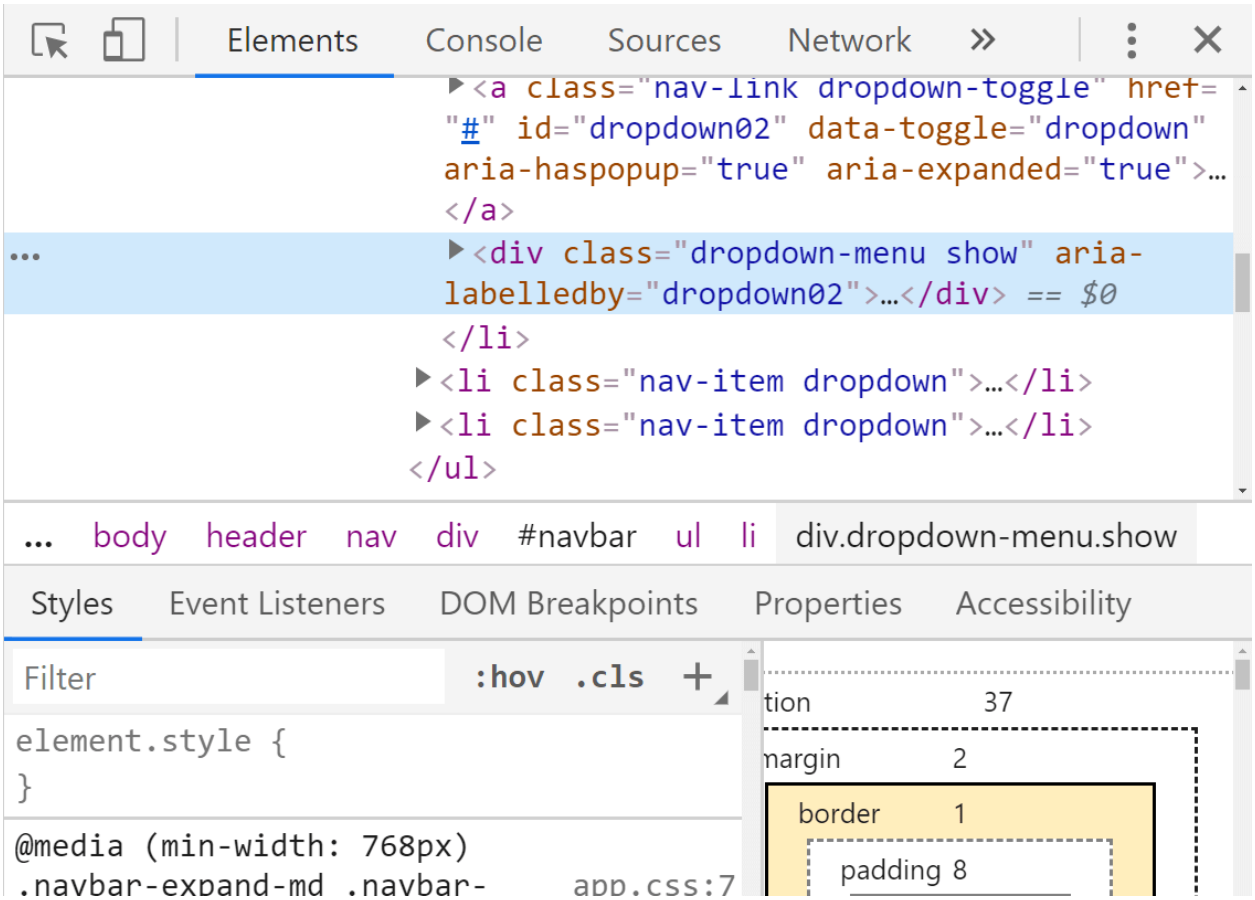
First, we'll open the preview of the Milo theme again. Next, we'll click the "Home" menu-item, so that its dropdown menu appears.



Right-click inside the dropdown, very close to its border

Next, click the `Inspect` command on the right-click's contextual menu.

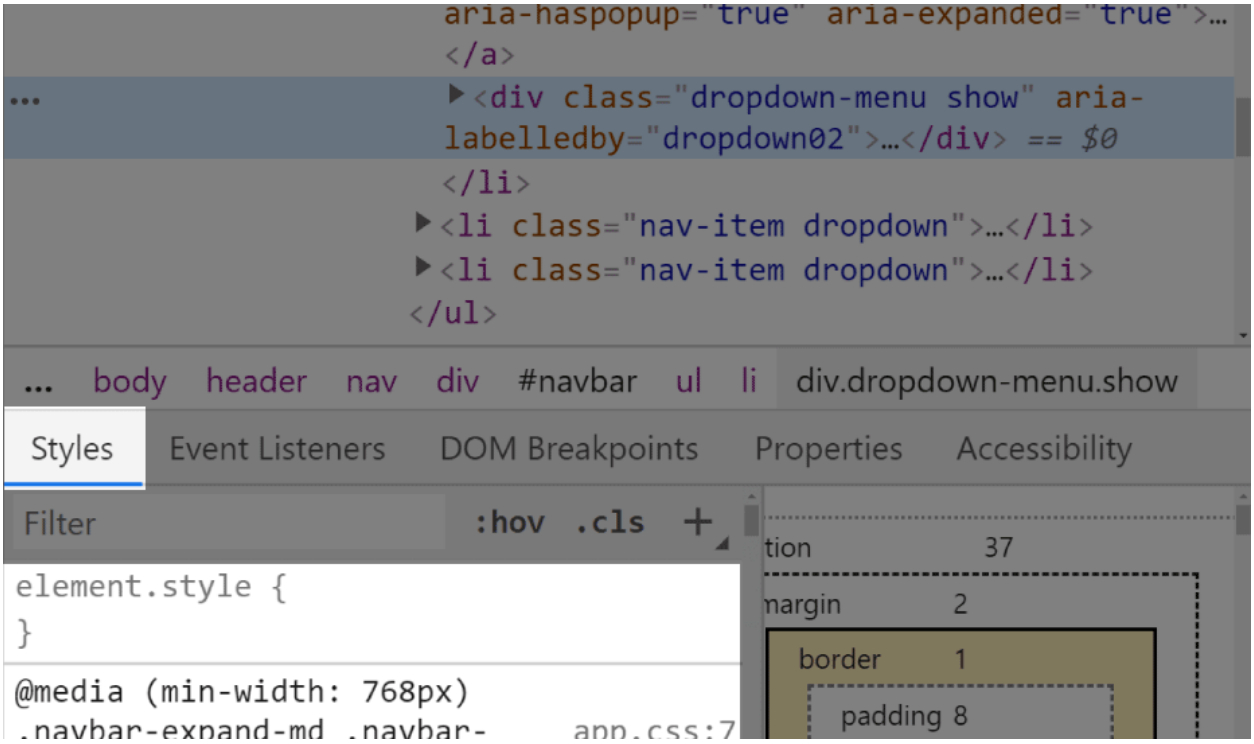
This will bring the dropdown's wrapping element into focus in the *Elements panel* inside devtools.



The dropdown menu element in focus in the Elements panel in devtools

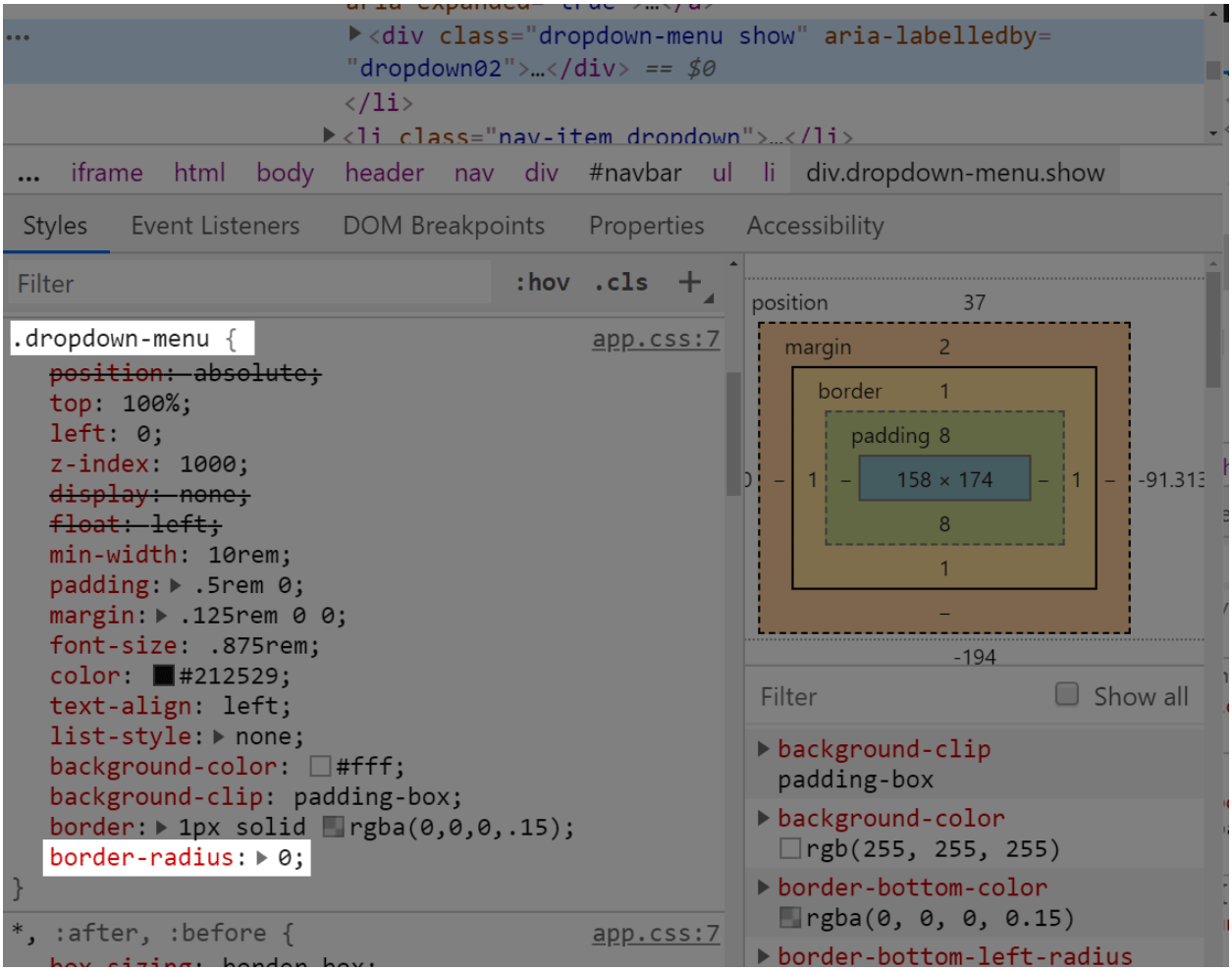
Now we need to focus on the *Styles* panel. Simply hover your mouse inside the styles under the *Styles* panel, and scroll down your mouse wheel. You're looking for the CSS class of `dropdown-menu`, and a value for the `border-radius` property on this CSS class.

The below image shows the area inside the *Styles* panel in which you need to point your mouse and then scroll down.



The area inside the styles panel where you need to scroll the mouse

With a bit of searching, it shouldn't be too hard to find.



Border radius value inside the dropdown-menu CSS class

Now, as mentioned above, you just need to add these styles to your own theme's `dropdown-menu` class.

```
.dropdown-menu, .form-control {
    border-radius: 0
}
```

In this CSS code, we are giving the same CSS declaration to more than one CSS class. This is perfectly normal in CSS. In fact, it's a very common technique.

To add a bottom border, we simply use the class `border-bottom`, and add it on the nav element:

```
<nav class="navbar navbar-expand-md navbar-light bg-white absolute-top border-bottom\
">
```

Now we can wrap up our layout's navbar by updating the branding. The first `ul` element inside the `nav`, there's an `<a>` tag (the anchor element), which holds the navbar's brand. We'll update it to this:

```
<a class="navbar-brand mx-auto order-1 order-md-3" href="index.html">Movies &amp; Ty\
pography</a>
```

What is the `&` in HTML

There is a special class of characters in HTML, called HTML entities, and the ampersand is a member of this group.

The `&` marks the beginning of any such HTML entity, and the `;` marks its end.

Why even use them?

We use them because sometimes browsers might get tripped up when they see such special characters, including the ampersand.

Examples include the ***less than*** sign, i.e `<`, and the ***greater than*** sign, i.e `>`.

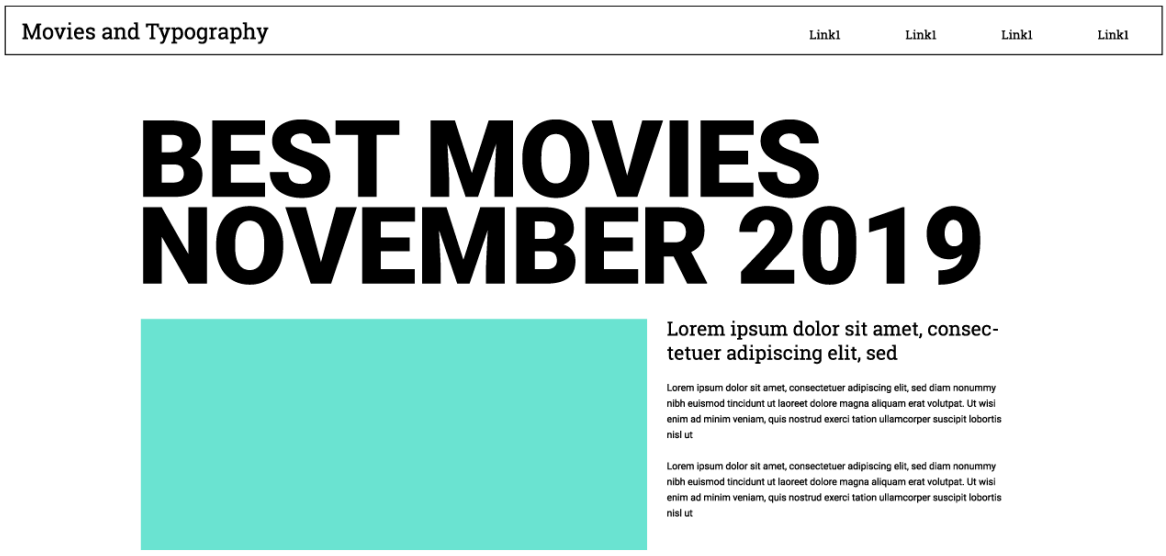
Written as an HTML entity, the less than sign becomes `<`, and the greater than sign becomes `>`.

TL;DR: If you wanna use the ampersand character, write it like this: `&` in your HTML.

Next, let's start adding the site's h1 heading.

Adding the h1 to our site

Earlier in this section, we saw the mockup for our layout:



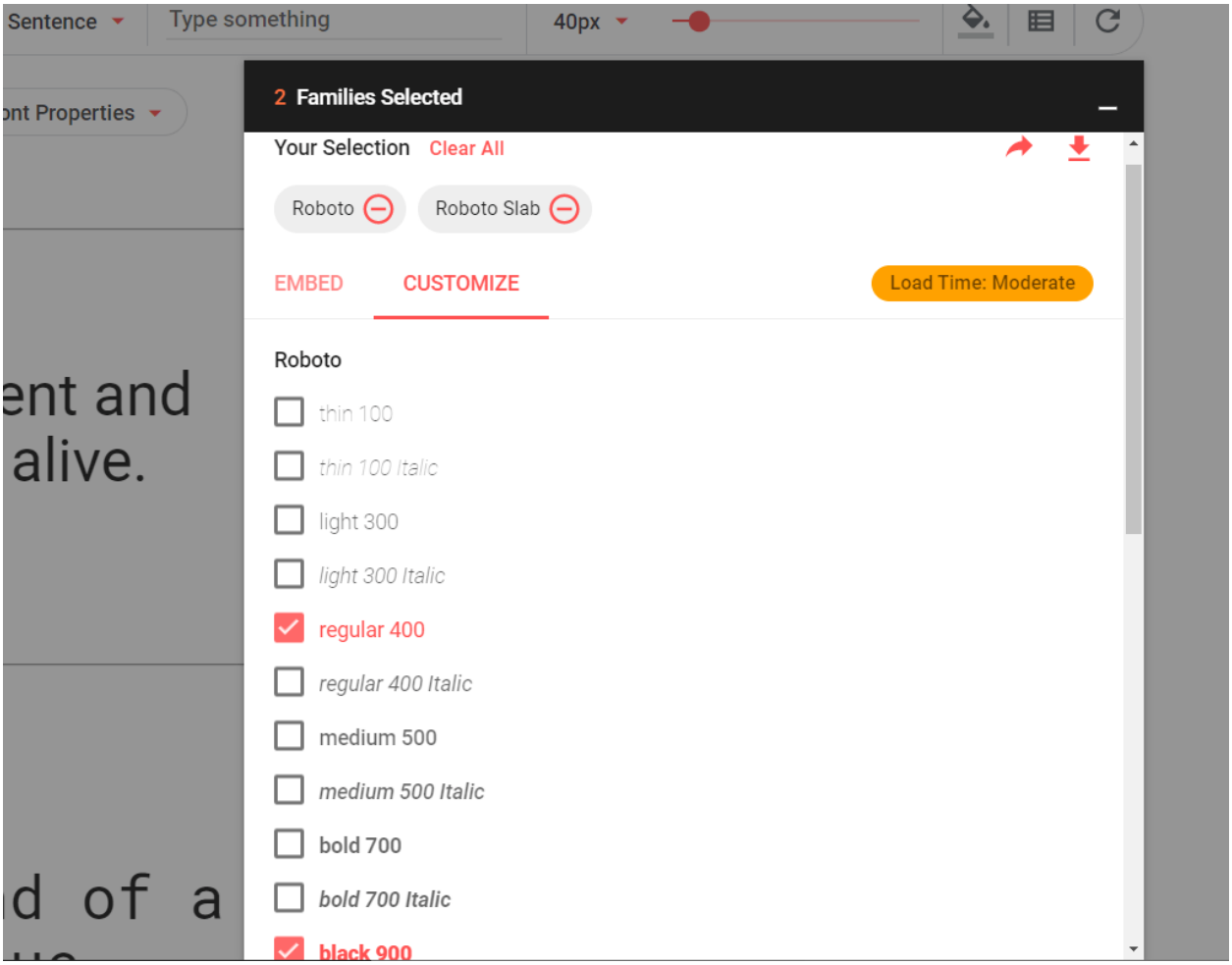
Mockup of the typography-focused layout

Now we'll add the heading: "BEST MOVIES IN NOVEMBER 2019".

The font used is Roboto with `font-weight` of `900`, and the font size is 116 pixels.

Let's add it from Google Fonts to our layout.

To choose the font weights to inlude in your Google fonts, click the ***Customize*** tab next to the ***Embed*** tab in the "2 Families Selected" window on Google fonts.



Customize Google fonts

Now we'll add the copied font import into the `<style>` tag just above the closing `</head>` tag in our layout:

```
<style>
    @import url('https://fonts.googleapis.com/css?family=Roboto+Slab|Roboto:400,900&\
display=swap');
    .dropdown-menu, .form-control {
        border-radius: 0
    }
</style>
```

Next, we can add the huge h1 heading styles:

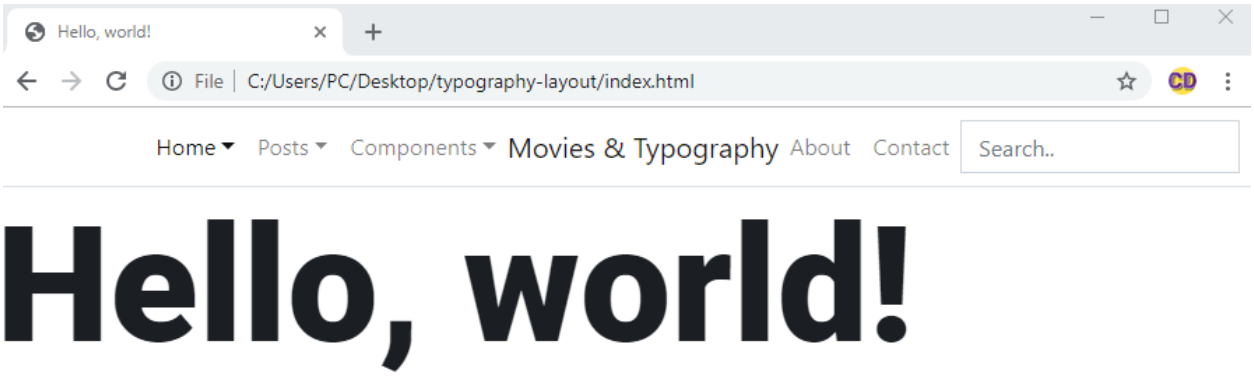
```
<style>
    @import url('https://fonts.googleapis.com/css?family=Roboto+Slab|Roboto:400,900&\
display=swap');
    .dropdown-menu, .form-control {
        border-radius: 0
    }
    .display-biggest {
        font-family: Roboto, sans-serif;
        font-weight: 900;
        font-size: 116px;
    }
</style>
```

Now we can use our new `display-biggest` class. Note that the reason we're using this class name is to follow Bootstrap's own large font class naming convention of `display-1`, `display-2`, `display-3`, and `display-4`.

For now, let's apply the `display-biggest` on the "Hello world" text under the navbar, on line 86 of our layout:

```
<h1 class="display-biggest">Hello, world!</h1>
```

Let's see what our layout looks like now:



Add the display-biggest class to an h1 element

This is much better, our layout is slowly becoming what we imagined.

At this point, it might be good to link to a live example of our incomplete layout[60].

The layout at this stage is also available for download[61].

Next, let's revisit a subject we covered earlier: containers.

Using containers to quickly structure our layouts

In the first chapter, we covered containers in Bootstrap 4.

We saw how we have two options, that is, two Bootstrap CSS classes to use:

1. `container`
2. `container-fluid`

We further mentioned that the fluid container spans the full width of the page, while the regular container gets centered in the page, and by default, its maximum width is 1140 pixels.

Why are this facts important for our layouts?

Because you can, and should, think about Bootstrap layouts as layers of containers. Let's see exactly what that means.

[60] https://www.codingexercises.com/codelabs/2019-10-04-building-bootstrap-layouts-article-7-pt-1
[61] https://www.codingexercises.com/downloads/2019-10-04/typography-layout-pt-1.zip

Bootstrap layouts as layers of containers

Bootstrap is very modular. We can just add and replace components (code) to get a new look.

We saw this approach earlier in this chapter when we replaced the default navbar with a custom one.

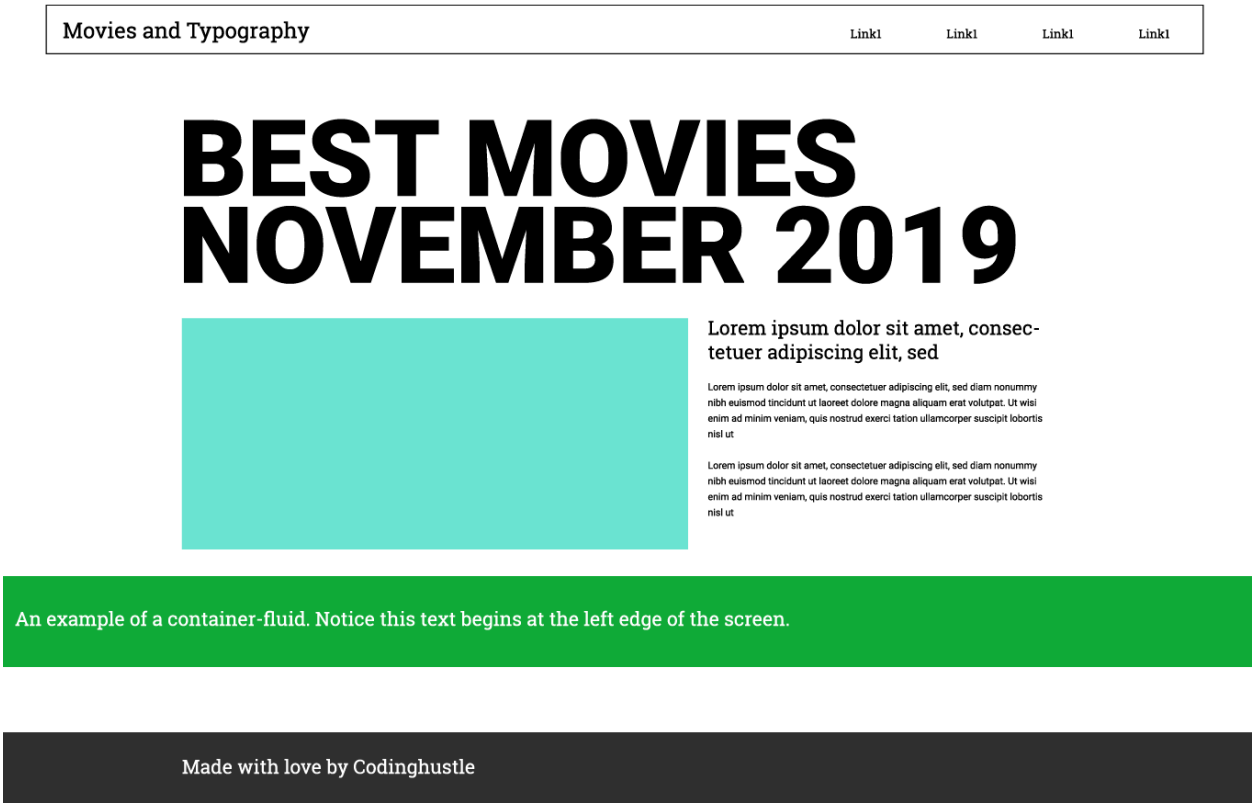
But this modularity goes even further: you can think of your entire layout as a layer cake.

Literally, this:

A layer cake

In our Bootstrap layouts, these layers are containers.

Let's revisit our mockup:



A mockup of the layout with container examples

In the above mockup, we see that with containers, we have three options:

1. We can use the `container` class, then wrap it in a `container-fluid` if we want to add it as a centered container with a full-width background
2. We can use the `container` class without the wrapping `container-fluid`
3. We can use the `container-fluid` class without the `container` class inside of it

Let's add each of these combinations to our layout.

1. Wrapping container in container-fluid

We'll build our footer using this technique, because our footer needs to stand out a bit.

We'll make it stand out by giving it a darker background color, and some lighter font color.

We'll add the darker background color to the wrapping `container-fluid` element, i.e the `<footer>`. Then we'll specify the font color on the inner `container` div.

Here's the code:

```
<footer class="container-fluid bg-dark">
    <div class="container text-light">
        Made with love by <a href="https://www.codingexercises.com">codingexercises.\
com</a>
    </div>
</footer>
```

2. Using container without wrapping it inside container-fluid

For now, we'll use this technique to give our *Hello world* some horizontal margin:

```
<div class="container">
    <h1 class="display-biggest">Hello, world!</h1>
</div>
```

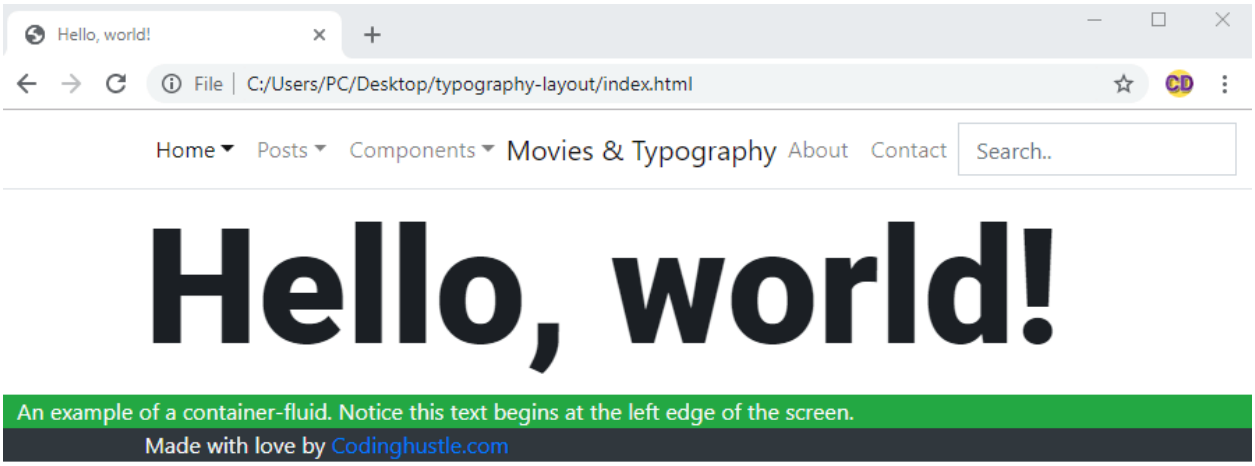
Above, we're just wrapping the `h1` element inside a `container` element.

3. Using `container-fluid` class without a container inside of it

In this layout, we don't really have much use for just a div with `container-fluid`, but we'll temporarily add it to our layout, just to see what it looks like in practice:

```
<div class="container-fluid bg-success text-light">
    An example of a container-fluid. Notice this text begins at the left edge of the\
 screen.
</div>
```

With all these updates added, our layout now looks like this:



The layout, part 2

We can still improve our containers, by adding some padding utility classes[62].

We'll update the green container to this:

```
<div class="container-fluid bg-success text-light pt-3 pb-3">
    An example of a container-fluid. Notice this text begins at the left edge of the\
 screen.
</div>
```

In the above code, we've added the `pt-3` (padding-top, "intensity" 3), and the `pb-3` (padding-bottom, "intensity" 3) CSS classes. The `pt-3` is the class for adding the CSS declaration of `padding-top` to an HTML element, and the `pb-3` updates the `padding-bottom` property on an element.

We'll update the footer with a short version of these two classes:

```
<footer class="container-fluid bg-dark py-3">
    <div class="container text-light">
        Made with love by <a href="https://www.codingexercises.com">codingexercises.\
com</a>
    </div>
</footer>
```

Understanding spacing utility classes in Bootstrap

If you want to add the same "level of intensity" on both the top and bottom padding classes, you can replace the `pt-*` and `pb-*` with `py-*`. Note that the `*` here needs to be the same number.

[62] https://getbootstrap.com/docs/4.3/utilities/spacing/#notation

For example, you can replace `pt-5` and `pb-5` with `py-5`. Why the `y` letter in this class name? Because we're updating the padding on the `y` axes.

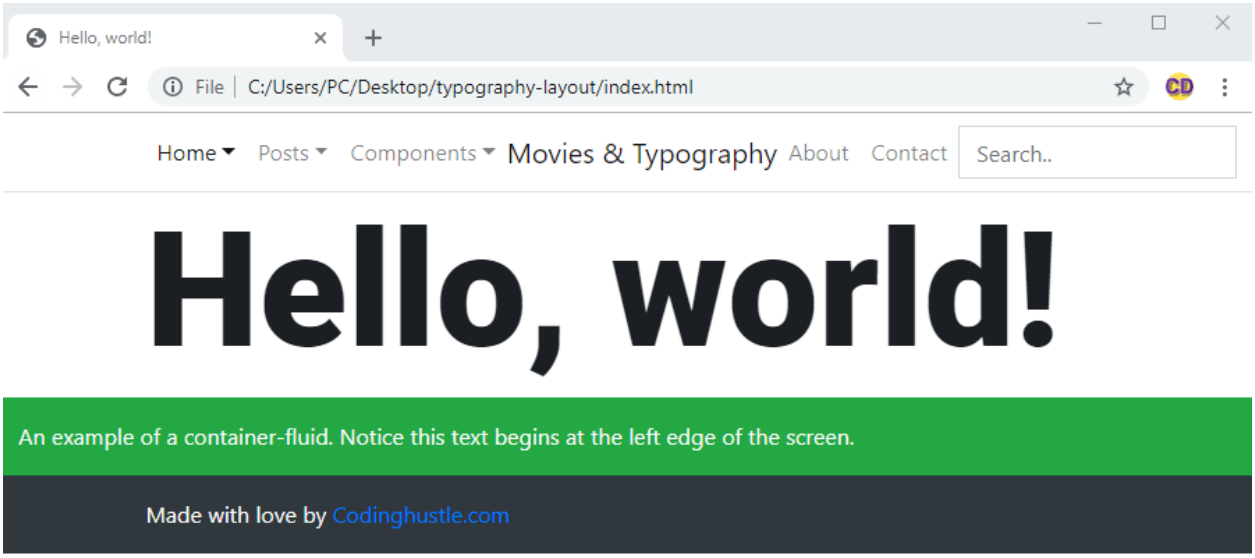
We can do a similar thing with the horizontal padding, e.g replace `pr-5` and `pl-5` with `px-5`.

We can work with margins in the exact same way, for example:

- we can replace `mt-1` and `mb-1` with `my-1`
- we can replace `mr-4` and `ml-4` with `mx-4`

Our layout, improved with containers

After the above updates, our layout now looks like this:



The layout, part 2, completed

Now we can see the result of the above updates in a live preview[63].

You can download it here[64].

Next, we'll add the the text and images.

Adding images and text to our layout

All that's left to do for this layout is add some elements in a layout grid.

We'll use two movie-themed images from Unsplash.com:

[63] https://www.codingexercises.com/codelabs/2019-10-04-building-bootstrap-layouts-article-7-pt-2/
[64] https://www.codingexercises.com/downloads/2019-10-04/typography-layout-pt-2.zip

- the first one is an image of an old theather[65] by Peter Lewicki[66]
- the second one is a photo of Netflix website on a laptop[67] by Charles PH[68]

Both the images will spread 7 columns of Bootstrap's 12-column grid. The first image will show on the left, with text on the right. The other image will appear on the right, and the text will appear on the left.

Let's add the first image, the Netflix image.

For this, we'll add a new container. We don't want any backgrounds covering the full width of the screen, so we won't be adding the `container-fluid` class:

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

Next, we'll add the row class:

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

Now we can add the columns.

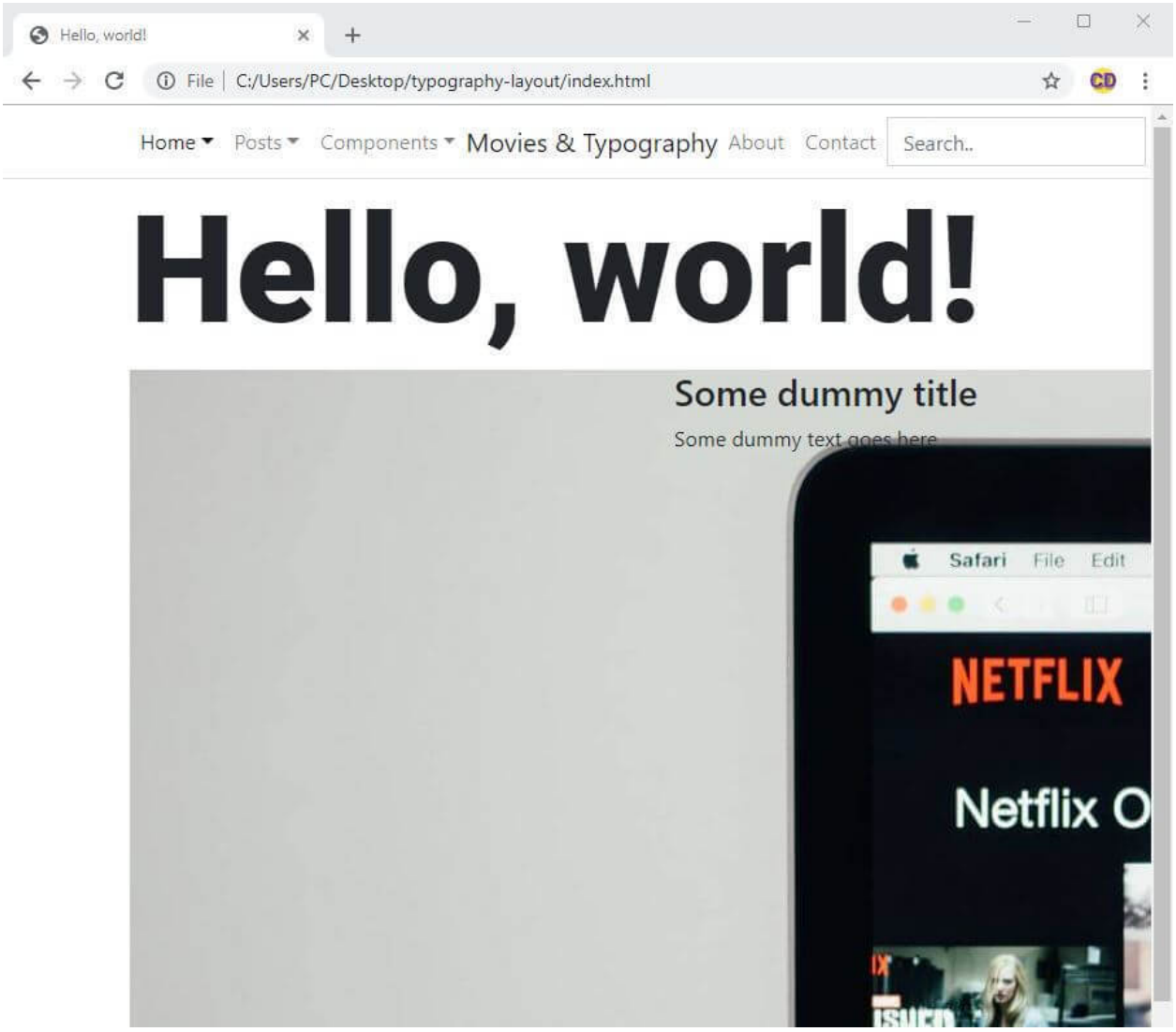
Adding the columns

Let's update our new container with the columns:

```
<div class="container">
    <div class="row">
        <div class="col-sm-12 col-md-7">
            <img src="./netflix.jpg" alt="netflix on a laptop">
        </div>
        <div class="col-sm-12 col-md-5">
            <h2 class="h3">Some dummy title</h2>
            <p>Some dummy text goes here</p>
        </div>
    </div>
</div>
```

[65] https://unsplash.com/photos/Wfh650C1OHU
[66] https://unsplash.com/@peterlewicki
[67] https://unsplash.com/photos/jtmwD4i4v1U
[68] https://unsplash.com/@charlesdeluvio

Here's the screenshot of our layout now:



The layout, part 3, in progress

Ooops, that's not what we expected!

The image is displayed in its full, original size. That's not good.

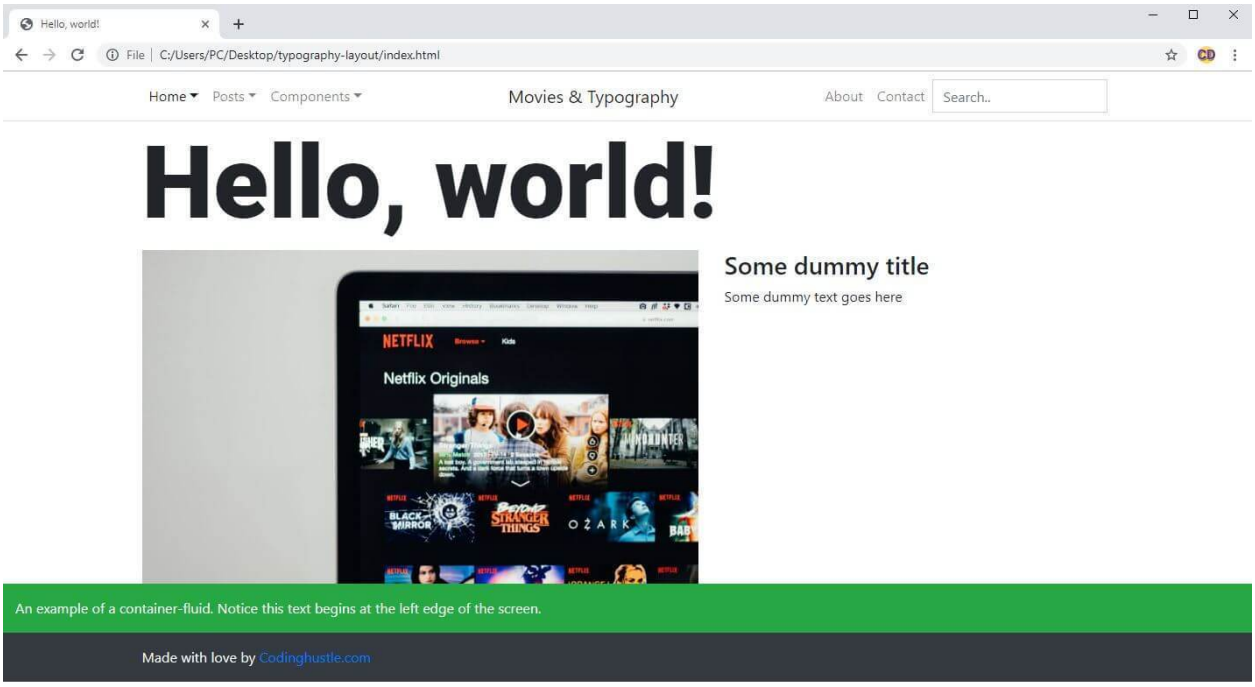
Luckily, it's an easy fix: we just need to add the Bootstrap 4 class of `img-fluid`. This class will make our image responsive[69].

Here's the updated `img` element from the above code:

```
1  <img class="img-fluid" src="./netflix.jpg" alt="netflix on a laptop">
```

That's it, it's all that it takes to get this:

[69] https://getbootstrap.com/docs/4.3/content/images/#responsive-images



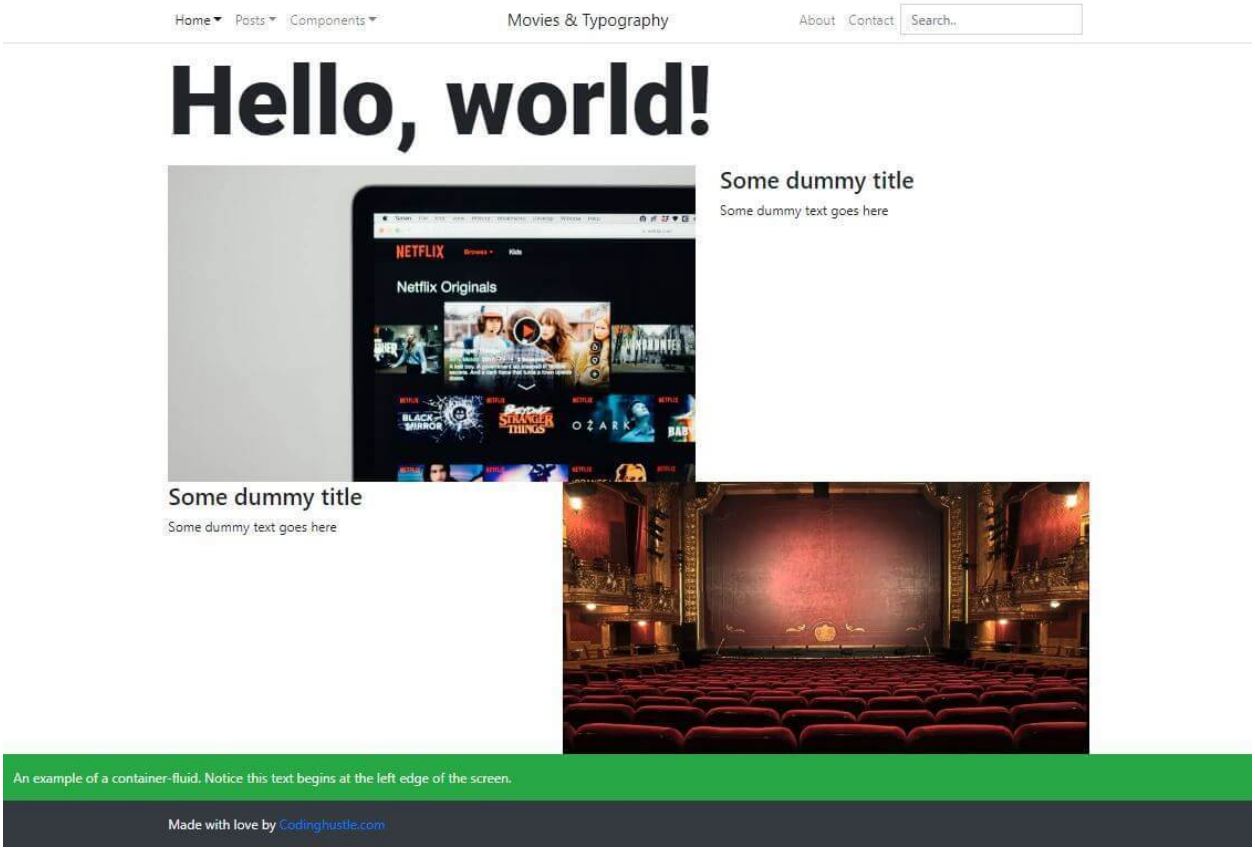
The layout, part 3, in progress, responsive images

Now we can add the other image too, as another row.

Here's the complete update:

```
<div class="container">
    <div class="row">
        <div class="col-sm-12 col-md-7">
            <img class="img-fluid" src="./netflix.jpg" alt="netflix on a laptop">
        </div>
        <div class="col-sm-12 col-md-5">
            <h2 class="h3">Some dummy title</h2>
            <p>Some dummy text goes here</p>
        </div>
    </div>
    <div class="row">
        <div class="col-sm-12 col-md-7">
            <img class="img-fluid" src="./old-theatre.jpg" alt="old theatre">
        </div>
        <div class="col-sm-12 col-md-5">
            <h2 class="h3">Some dummy title</h2>
            <p>Some dummy text goes here</p>
        </div>
    </div>
</div>
```

Here's the screenshot of the update:



The layout, part 3, complete

You can also download the layout as it is now[70], or view it live in the browser[71].

Our layout is almost done; there's only a few minor fixes left.

Polishing up our layout

There are a few things to improve:

1. We could try adding more space between each row in our layout.
2. We'll remove the example fluid container fluid with the `bg-success` contextual color
3. We'll see how to quickly flip the order of columns in our layout
4. We'll make the footer stick to the bottom no matter what!
5. Update the typography (on paragraphs, `h2` elements, and in the footer)

[70] https://www.codingexercises.com/downloads/2019-10-04/typography-layout-pt-3.zip
[71] https://www.codingexercises.com/codelabs/2019-10-04-building-bootstrap-layouts-article-7-pt-3

Let's first add more space between each row.

We've learned about the spacing utility classes in this article, and now we have a chance to use them.

Let's update both rows with the class of `my-5`, like this:

```
<div class="row my-5">
```

That's it for the spacing, now let's just erase the green bar above the footer. This is the code to delete:

```
<div class="container-fluid bg-success text-light pt-3 pb-3">
    An example of a container-fluid. Notice this text begins at the left edge of the\
 screen.
</div>
```

Now we'll see how to quickly flip the order of columns in our layout with a technique known as "source ordering".

Source ordering our column grid

Source ordering is the practice of keeping our source (the HTML) the same, but ordering our layout differently, using CSS.

To achieve this, we can use Bootstrap 4 CSS classes of `order-1` to `order-12`, where lower numbers come first.

Here's the update in the second `row`:

```
<div class="row my-5">
    <div class="col-sm-12 col-md-7 order-1">
        <img class="img-fluid" src="./old-theatre.jpg" alt="old theatre">
    </div>
    <div class="col-sm-12 col-md-5 order-2">
        <h2 class="h3">Some dummy title</h2>
        <p>Some dummy text goes here</p>
    </div>
</div>
```

You might think at this point, why even bother with source ordering? Shouldn't I just order the HTML to get the layout I want?

There's actually two reasons for source ordering:

1. Layouts are responsive and complex

2. SEO

There's sometimes a slight SEO advantage - more importance given - to those HTML tags that appear earlier in the site structure.

Also, layouts can get complex, so you might not want the same visual order of your columns on different breakpoints (i.e on mobiles, tables, laptops, and desktops).

Now I'll show you a cool trick in devtools' JavaScript console.

Testing layout variations using the JavaScript console in the developer tools

To get to it, open the devtools with F12, then press the ESC key to open the console.

After that, just type:

```
document.designMode = 'on'
```

Now you can type directly into the layout! You can even drag images around. The entire browser window becomes similar to Photoshop.

Let's now take care of the footer.

Making the footer stick to the bottom

In our layout, this is very easy to do. We'll just set the position to `fixed` and the bottom to `0`.

We'll also need to set the height of the `<body>` element to 100 percent of viewport height. The viewport height has its own unit of measure in CSS: the `vh` unit. Thus the CSS declaration will be: `min-height: 100vh`.

Here's the full updated CSS:

```
<style>
@import url('https://fonts.googleapis.com/css?family=Roboto+Slab|Roboto:400,900&disp\
lay=swap');
.dropdown-menu, .form-control {
    border-radius: 0
}
.display-biggest {
    font-family: Roboto, sans-serif;
    font-weight: 900;
```

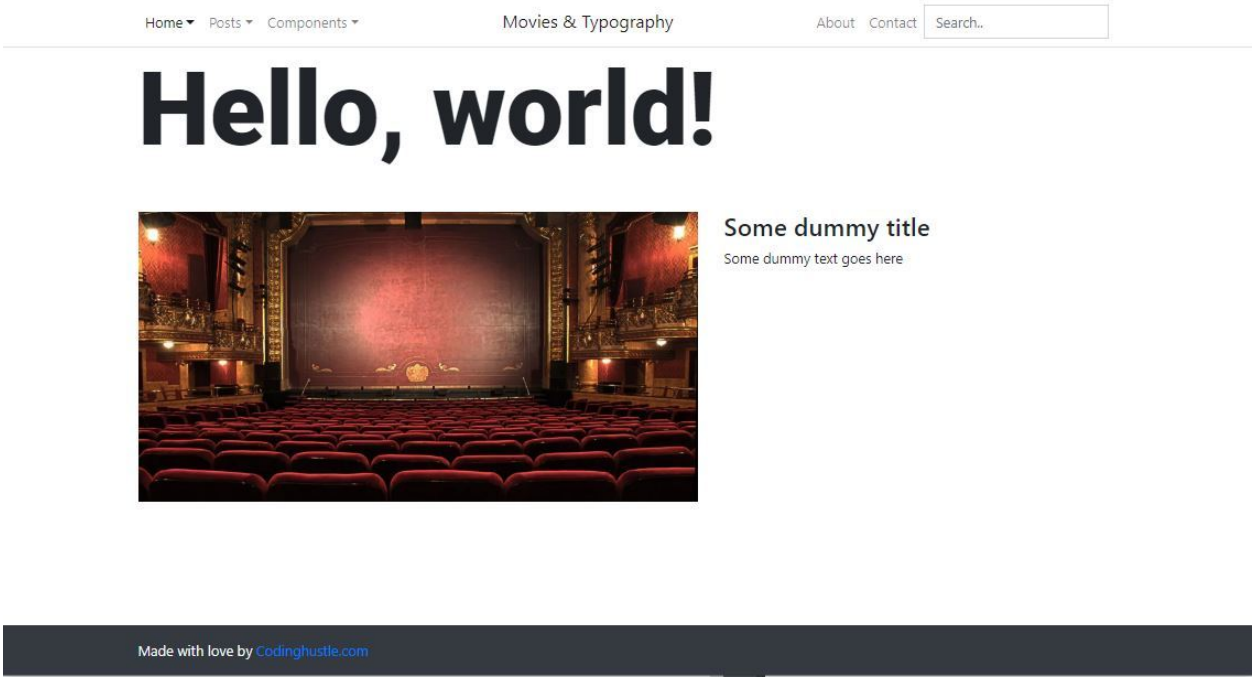
```
    font-size: 116px;
}
.body {
    min-height: 100vh;
}
.pos-fix.b0 {
    position: fixed;
    bottom: 0
}
</style>
```

And we'll update the `<footer>` element accordingly:

```
<footer class="container-fluid bg-dark py-3 pos-fix b0">
    <div class="container text-light">
        Made with love by <a href="https://www.codingexercises.com">codingexercises.\
com</a>
    </div>
</footer>
```

To test if the footer indeed sticks to the bottom, you can open the devtools and click on the top row, then press the DELETE button. This will remove the row from the DOM (Document Object Model) of the page. Note that this is a non-destructive change - meaning it won't save changes, so you can do it without worrying you'll mess up your layout.

Here's a screenshot of testing out if our footer stays in place with one row erased from the DOM:



The layout, part 4, testing

Improving typography

Let's also change the color on the codingexercises link from default browser blue to the `text-light`. We'll also use the `lead` text class to make it a bit more prominent:

```
<footer class="container-fluid bg-dark py-3 pos-fix b0">
    <div class="container text-light">
        Made with love by
        <a class="text-light lead" href="https://www.codingexercises.com">
            codingexercises.com
        </a>
    </div>
</footer>
```

Next, let's make the entire page use the Roboto font:

```
* {
    font-family: Roboto, sans-serif !important;
}
```

The `*` is called the universal selector, and the `!important` flag we used on it overrides everything.

It's best to not use it if you don't have to, but if you do use it, it's likely to override all the other styles on your page. In other words, it will disregard the CSS specificity.

Next, we'll set the font of *Roboto Slab* on h2 headings.

```
h2 {
    font-family: "Roboto Slab", serif;
}
```

You must use quotes around font families whose names contain spaces.

Unfortunately, the above CSS won't work, because we've applied the `!important` flag on the wildcard selector earlier.

To fix the issue, simply erase the `!important` flag from it.

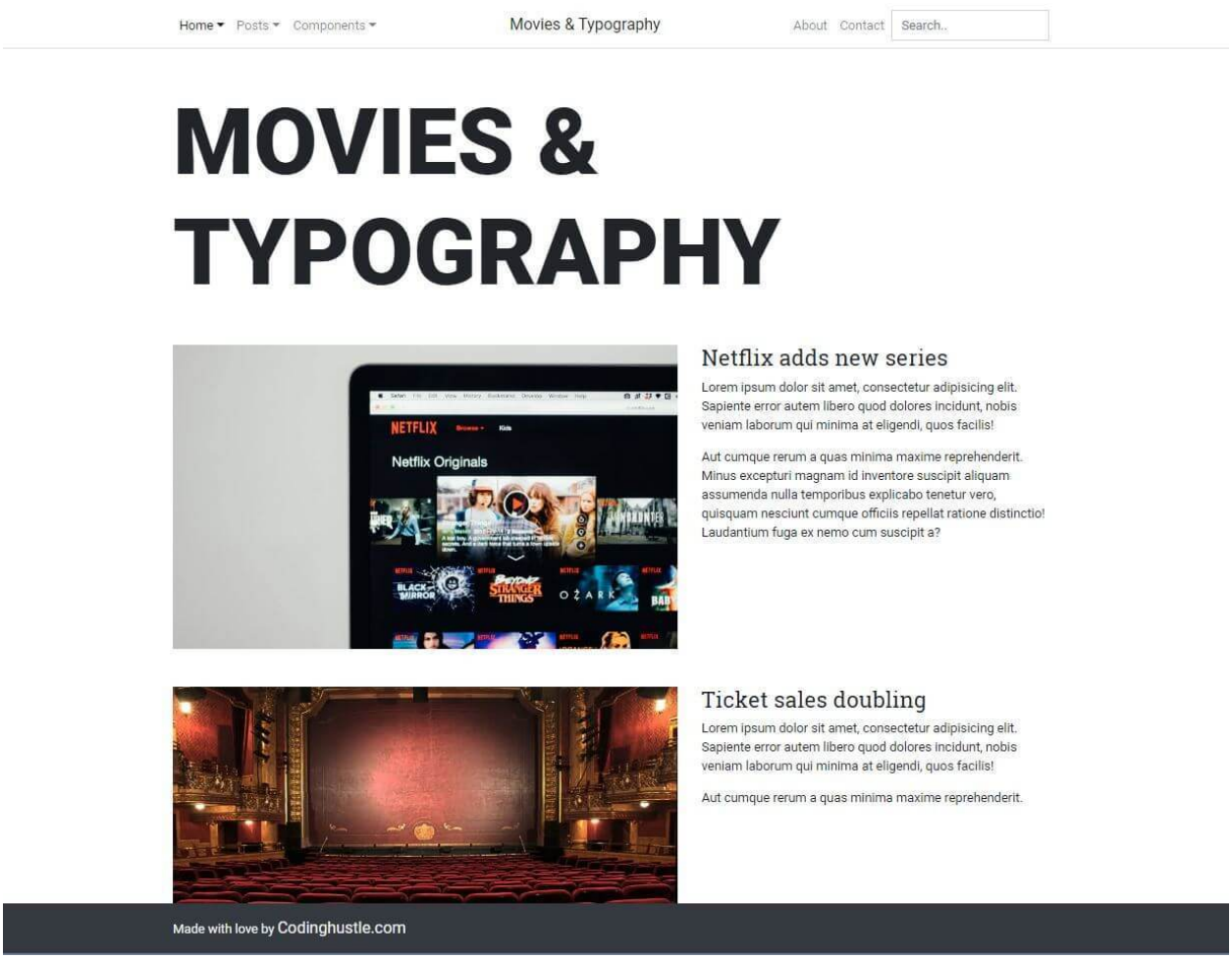
The completed CSS now looks like this:

```
* {
  font-family: Roboto, sans-serif;
}
h2 {
  font-family: "Roboto Slab", serif;
}
.dropdown-menu, .form-control {
  border-radius: 0
}
.display-biggest {
  font-family: Roboto, sans-serif;
  font-weight: 900;
  font-size: 116px;
}
body {
  min-height: 100vh;
}
.pos-fix.b0 {
  position: fixed;
  bottom: 0
}
```

We've also added some more lorem ipsum text under each title, so that our layout looks more like a real web page.

Finally, we've added the text class of `text-uppercase` to the main, `h1` heading, so that the font is all capitalized.

Here's the screenshot of the completed layout:



The completed Movies and Typography layout

You can also download the layout as it is now[72], or view it live in the browser[73].

That's it for this chapter. Next, we'll revise what we learned.

Conclusion

In this chapter, we've learned the following:

1. How to download Google fonts to our layout
2. How to use the starter template from the official Bootstrap docs
3. Use Brackets code editor because it's very beginner-friendly (and it has Live Preview built-in!)

[72] https://www.codingexercises.com/downloads/2019-10-04/typography-layout-pt-4.zip
[73] https://www.codingexercises.com/codelabs/2019-10-04-building-bootstrap-layouts-article-7-pt-4

4. How to add the navbar from another layout into our own layout
5. How to inspect elements in the developer tools
6. How to copy, alter, and delete elements in the developer tools
7. How to make borders square on bootstrap dropdowns and form inputs
8. How to work with styles inside developer tools
9. What is the `&` and why it's used
10. How to use `container` and `container-fluid`
11. How to use spacing utility classes in Bootstrap
12. How to add images and text to our layout
13. What is source ordering and why use it
14. How to make our browser behave like Photoshop (sort of), with the help of `document.designMode = 'on'`
15. How to make our footer stick to the bottom
16. How to use the `*` selector, the `!important` flag, and `text-uppercase` when building Bootstrap-based layouts

In the next chapter, we'll discuss a better way to develop our layouts (by making them modular), we'll use SCSS, and we'll touch on a more efficient way to learn coding.

Chapter 7: Modularize your Bootstrap 4 layouts

In this chapter we'll discuss the options we have for modularizing our layout development.

We can start with a question: what are we still missing in our layout-building process? In other words, where is the space for improvement?

How to improve our layout-building process?

To improve our layout building process, we need to think about where it is currently lacking.

So far in this book, we've covered a number of tips, tricks, and techniques in building layouts with Bootstrap 4.

We've covered SCSS, but we saw that it's a bit hard to set up. And it's a bit clunky to use (when you are just starting out with it).

We've covered Bootstrap 4 components, which are sort-of an implementation of the concept of Don't repeat yourself (DRY development) - because each component always starts from the same code, and some components are "composable" - meaning they are built by combining more than one component.

The question is, can we take the concept of DRY and combine it with the approach to layout components that Bootstrap has?

There's another challenge to solve, and that is the challenge of speed:

- How do we build Bootstrap 4 layouts, faster?
- How do we implement various components, faster?
- How do we learn and improve our skills, faster?

There is no perfect answer to these questions. Or, put differently, modern web technologies give us a plethora of ways to solve these issues.

So what are we to do?

As beginners (which I'm assuming you are), we need to find the balance between getting new knowledge and practicing what we already know.

It's a hard thing to achieve, because these two goals are often in collision with one another.

Ultimately, the choice boils down to a simple cost-benefit analysis: ***Based on the investment, what are my returns?***

That's why I've decided to continue this book with the help of Angular.

Angular is a mature front-end web development framework, backed by Google.

Here are the reasons to use it when learning Bootstrap layouts:

1. It solves the "problems" with SCSS
2. It helps us modularize our layouts (easily!)
3. It helps us keep dry
4. We can do this quickly (with the help of Stackblitz)
5. Consequently, we'll become faster in building layouts
6. Using Angular with Bootstrap 4 will help us be faster to learn and improve our skills the right way

You might ask at this point: why not some other front-end framework?

Because Angular is very easy to start with and has a streamlined approach, meaning, there are usually only a few accepted ways to work with it.

This limits our options, which is good.

While other frameworks ("libraries") are like: "Here are *all* the car parts, now go assemble a car", Angular is more like: "Here are the *major building blocks* of a car, go assemble a car".

Anyway, there's always plenty ways to be opinionated in web development, but that is not productive. Instead, let's get started!

Understanding how Angular works

Angular is complex - but so are other frameworks.

This can make it look like it's a lot harder to learn than it really is.

One sure way to make it easy to understand is to limit the scope of what we are trying to do.

We have only 1 goal right now: to split our large one-file HTML layouts, into several smaller files.

Thus we are not ***really*** trying to understand how the ***entire*** Angular framework works.

What we are doing here is: we're learning how to split our HTML files with the help of Angular.

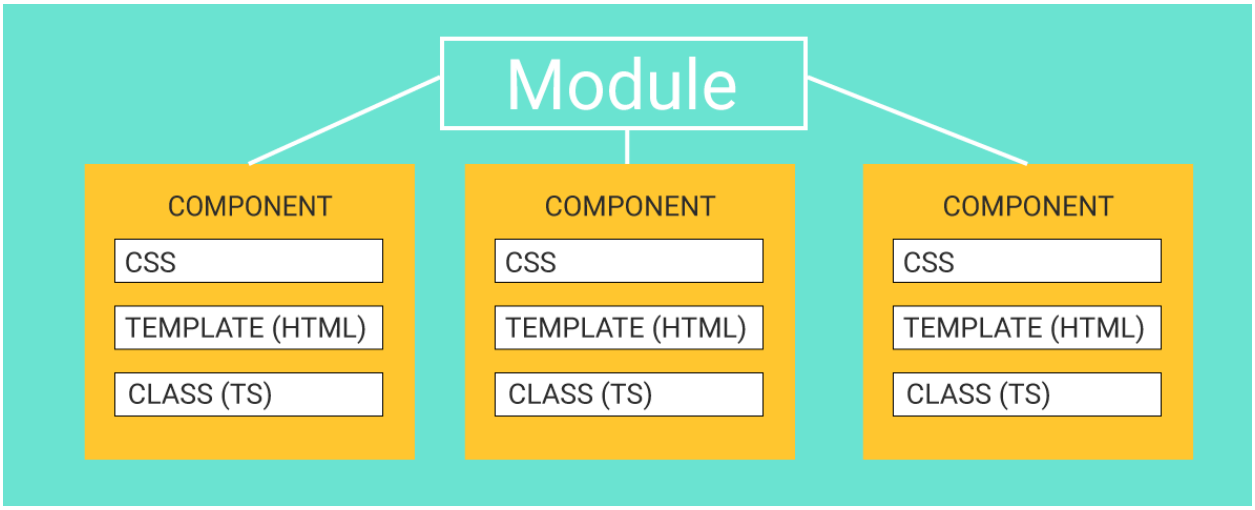
That's all that we're doing in this chapter!

How to split HTML files with Angular

Before we can even get to the HTML part, we will look at how code is organized in Angular:

- ***Modules*** hold components
- ***Components*** hold:
  * the ***styles*** file,
  * the ***HTML template*** file,
  * the ***class*** file

Let's see a diagram of this:



A module holds components

An Angular module holds components together. Inside each component in Angular, we already know everything we need to know about the ***styles*** file and the ***template HTML*** file: the styles file holds all the styles that will apply only to the HTML of their specific component. The template HTML file is just plain HTML - that's how we'll use it.

But what's with this "class (ts)" file?

The class file in an Angular template

The class file is just there to do two things:

1. List the component's HTML and CSS files
2. Add some functionality (interactivity)

While point 1 is easy to comprehend, point 2 might need a bit more explaining.

How does the class file add functionality in an Angular component?

If we were not using a framework, the interactivity would be added with plain JavaScript.

If we say that our website is like a house, we could say that:

- HTML is the house structure (the walls, the doors, the roof, etc)
- CSS is the house coloring and decoration
- JavaScript is the house functionality: opening and closing the doors, turning on the lights, etc.

Angular takes things one step further and uses another language: TypeScript. Let's explain TypeScript and JavaScript relationship by looking at the relationship between SCSS and CSS.

In Book 1 of this book series, we said that SCSS is a superset of CSS.

What that means is that SCSS ***includes*** all the CSS, ***plus some additional features that CSS doesn't have***. SCSS also compiles down to CSS, the process which we described earlier in this book.

It's also important to note that we can write pure CSS inside of SCSS files. ***We never have to use the SCSS functionalities in SCSS files!*** Since SCSS includes the entire CSS functionality, we can stick to using only this subset of SCSS.

Why then use SCSS anyway? Because of the partial include files!

Similar to SCSS and CSS, in Angular we use TypeScript by default. But TypeScript is a superset of JavaScript, and we're fine - for the most part - using plain JavaScript inside TypeScript files.

If all of this is making your head spin, I've got good news for you: We will use almost no TypeScript or JavaScript functionality in this chapter. No opening and closing doors in our HTML house!

It's simply too early in this book series to be that advanced. And there's no need for it at this point.

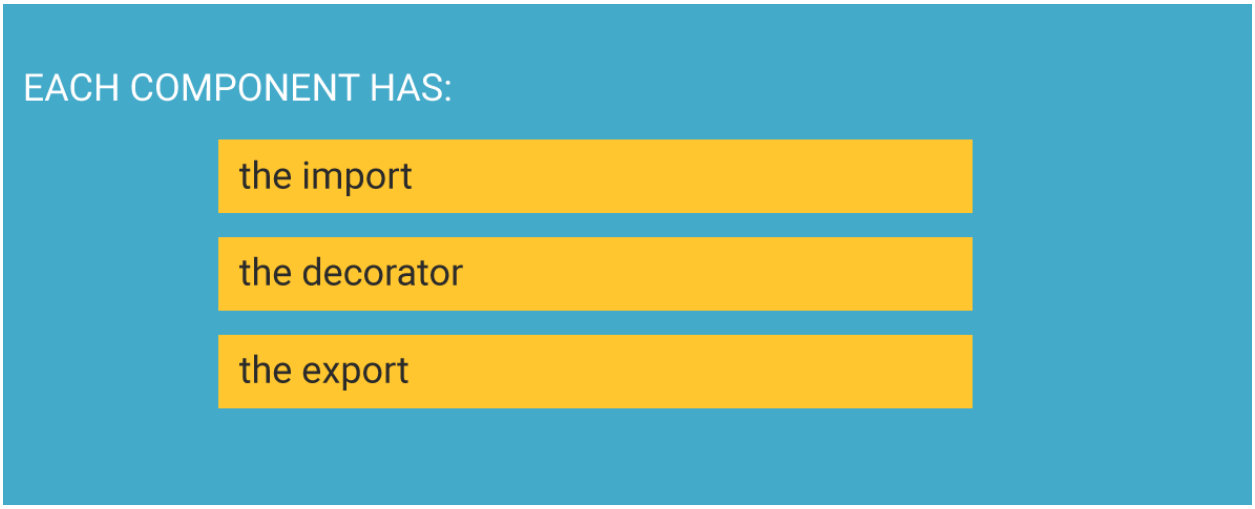
Shameless plug: If you would like to learn about JS, I've written an entire book series of 5 books about it: ***A Better Way to Learn JavaScript***. But that's a bit out of scope of what we're doing here.

What we ***will have to do*** however, is use just the bare necessary minimum of TypeScript inside the class file, just to make it possible to split HTML files into different components.

Thus let's look at this minimal required understanding of the component's class TypeScript file.

Understanding the minimal code of a component's class file

Here's a diagram of what a component's class file looks like:



Each component has the import and the decorator and the export

Each component must have ***the import section***, ***the decorator section***, and ***the export section***.

The **import** section imports all the code that will be used in a specific component. The bare minimum for the import section is this:

```
import { Component } from '@angular/core'
```

The above code means that we are importing the Component class from the `@angular/core` JavaScript module (Node module, to be precise).

The **decorator** section adds options to the imported Component, like this:

```
@Component()
```

The `@Component` is called ***the decorator***. Inside the decorator, we add our component's ***meta data***: the custom HTML selector the component will be given, the location of the component's template HTML file, and the location of the component's CSS file:

```
@Component({
    selector: 'some-kebab-cased-custom-html-element-name',
    templateUrl: './component-name.component.html',
    styleUrls: ['./component-name.component.css']
})
```

Additionally, the ***export*** code is just this:

```
export class ClassNameComponent {
    // additional interactive
    // functionality is set up here
    // we won't use this section
}
```

That's it, we've just explained all the building blocks of a minimum required code for a component to work in Angular.

Finally, a component that's prepared this way needs to be imported into its module. To import a component into a module, we need to do two things:

1. Use the `import` syntax to import the component into a module
2. List the imported component in the declarations array of the module where it's imported to

That's it!

This might look like a lot of theory, but as soon as we build our first one or two Bootstrap layouts in Angular, this will be easy as pie.

Setting up our Angular minimal app

In the past, Angular required a bit more setup, but it is getting ever more approachable.

Currently, one of the easiest and fastest ways to set up an Angular prototype is to use Stackblitz[74].

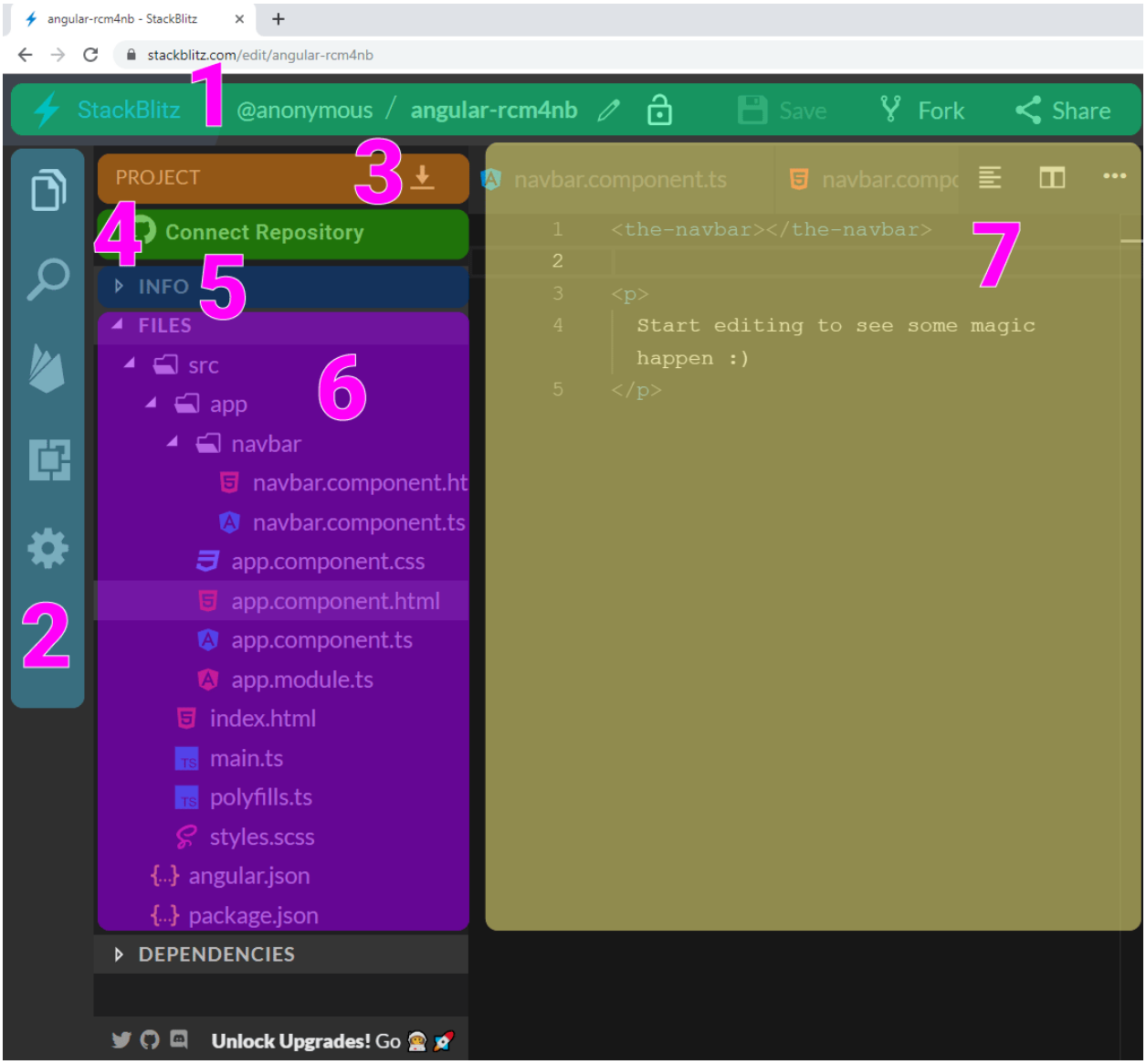
The layout we'll be looking at is the exact same layout we built in the previous article in this article series. It's the *Movies & Typography layout*.

Lets begin by looking at the first, incomplete version of our layout, with only the navbar added[75].

The Stacblitz app is actually an instance of the VS Code editor, running online. That's great, because if you've used VS Code before, you should feel right at home!

If you haven't used it, here's a quick tour:

[74] https://stackblitz.com/
[75] https://stackblitz.com/edit/angular-rcm4nb



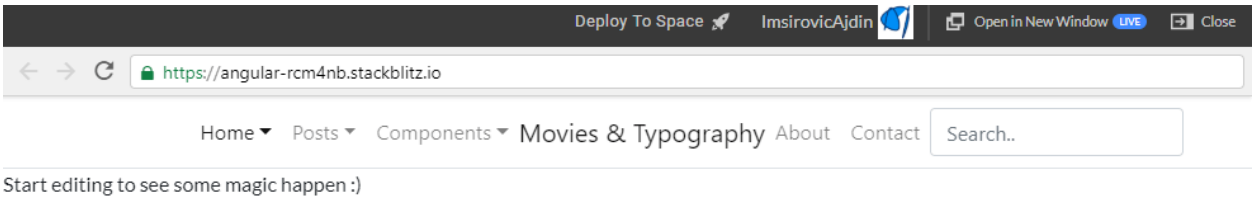
A quick tour of stackblitz

Let's explain each section of Stackblitz above:

1. The bar on top has ***the link*** to the Stackblitz home, then your ***username*** (if you're logged-in), the ***unique name*** for the app online, ***padlock*** for privacy settings, ***fork*** to copy the project in one click, and the ***share options***
2. The sidebar on the left inludes: ***project*** files list, ***search*** functionality, link for ***firebase*** hosting, ***extensions*** section, and the ***settings*** button
3. This sidebar depends on what button from the sidebar (item 2 above) is clicked. If the active sidebar button is clicked, then you will see the details of the project list; under number 3 on the image above, you have a little ***down arrow icon***; that's the download button to easily download your Angular project

4. The ***Connect repository*** is for Github integration (we'll cover Github later, so for now, ignore this button)
5. If you clicked on the ***Project info***, you'd see some basic stats: how many views your stackblitz received, and how many times it was forked. You'll also get the project description: Starter project for Angular apps that exports to the Angular CLI
6. The ***files section*** lists all your project's files, and finally
7. The ***editor groups*** are holds the files you can edit. If you double click any file, it will open here.

In the right half of the screen (on a desktop), you can see the preview of your app - or sometimes you'll see some errors. Either way, you can see the end result of your code in this right half section on Stackblitz:



A screenshot of our app v1 preview on stackblitz

In the above preview, we can see that our *Movies & Typography* layout is starting to come together nicely.

We've added the navbar, and now, before we continue, let's look at the actual code that made this possible.

Inspecting our app's code on Stackblitz

At the beginning of this article, we've covered the basics of how modules and components in Angular work together.

We'll now look at a practical example, by examining how we added our layout's navbar.

The top-level structure of our Stackblitz app currently looks like this:

```
src/
 └── ...
angular.json
package.json
```

In the root of our Stackblitz project, there's the `src` folder and two files:

- `angular.json`, and
- `package.json`

The `package.json` file holds our project's information. This is a big topic, and I've covered it in this in-depth article[76].

However, we won't touch the `package.json` file in this article, so at this point, you really shouldn't go down that rabbit hole.

The `angular.json` file holds our Angular settings, and we can change those settings if we want to.

When you start a new boilerplate Angular app on Stackblitz, these files come with a new installation. Sometimes you need to update the `angular.json` file, which is exactly what I did here, on line 26 of `angular.json`:

```
"src/styles.scss"
```

Why this update? Because there's a file inside the root of the `src` folder, called `styles.css`, which holds the styles that will apply to the entire app, *globally*. By renaming this file to `styles.scss` (provided that you've already updated the `angular.json` file), you're now ready to use SCSS instead of CSS in all your files.

Here's the contents of the `src/styles.scss` file:

```
/* Add application styles & imports to this file! */

@import url(https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css\
);
```

As you can see, we're importing the Bootstrap framework, version 4.3.1, from the Stackpath CDN.

[76] https://www.codingexercises.com/guides/understanding-node-npm-and-javascript-modules

The purpose of a CDN

CDN stands for "Content Delivery Network". A CDN is a global network of servers that hosts popular libraries on these servers. The Bootstrap framework is an example of such a popular library.

But why the global network of servers, and not just one server in one location? Well, that's the whole purpose of a CDN: to host files close to where a user is.

So, for example, if you're located somewhere in the USA, you'll get served the Bootstrap 4.3.1 minified CSS file from, for example, New York, or LA. If you're visiting it from Europe, it might get served to your from the server in London or Frankfurt.

The point of a CDN is speed. A CDN helps serve files quickly to website visitors. There are other things that a CDN does too, but let's keep it at that for now - and get back to examining our project's file structure.

The contents of the src folder

Here are the top-level items inside of the `src` folder:

```
src/
  ├── app/
  │   ├── ...
  │   ├── ...
  │   └── ...
  ├── index.html
  ├── main.ts
  ├── polyfills.ts
  └── styles.scss
```

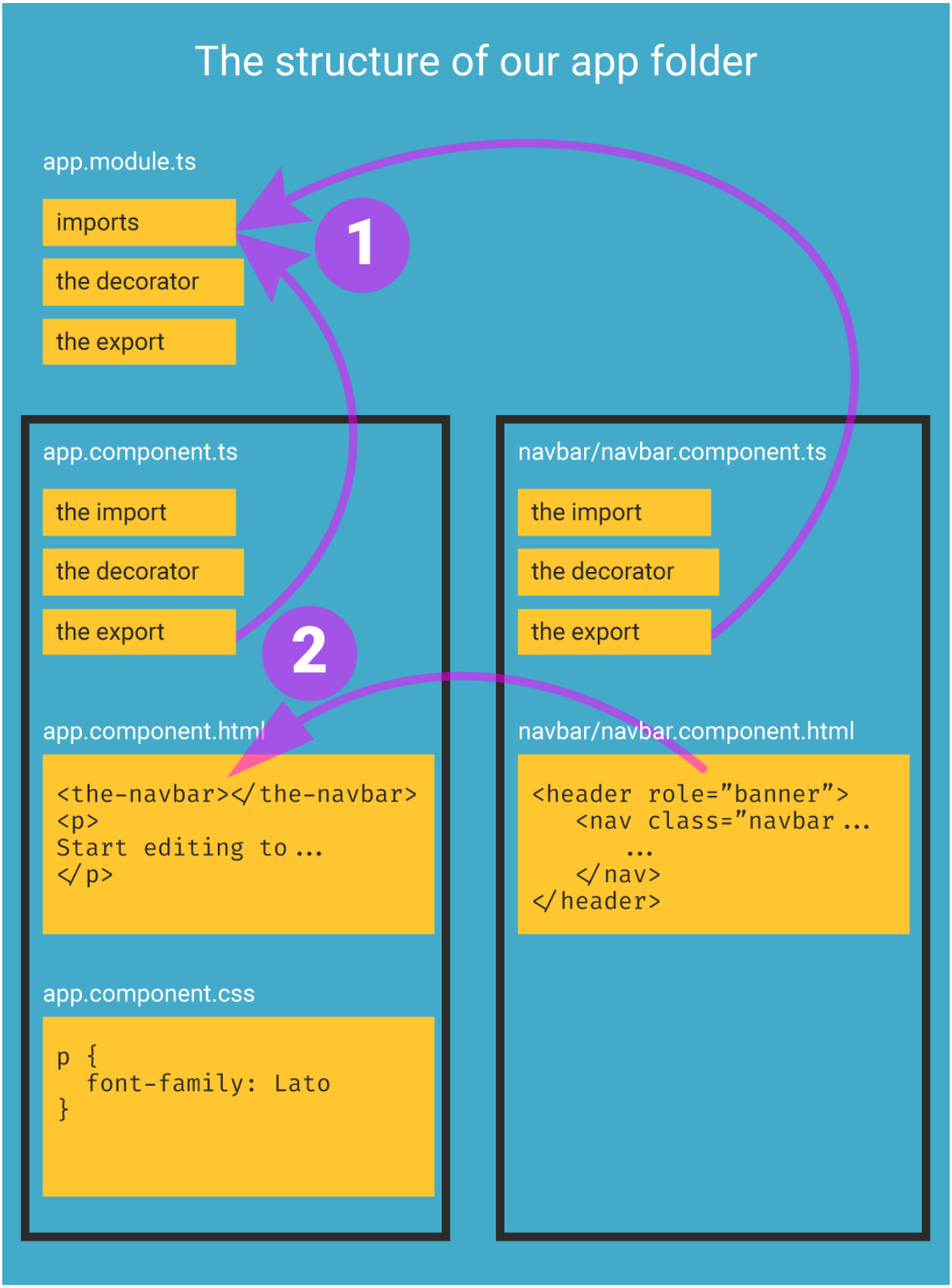
There are five items altogether. Out of these five, only the `styles.scss` file and the `app` folder are of any relevance for us. We can disregard the rest.

Drilling down further, the `app` folder holds these files and folders:

```
app/
  ├── navbar/
  │   ├── navbar.component.html
  │   └── navbar.componetn.ts
  ├── app.component.css
  ├── app.component.html
  ├── app.component.ts
  └── app.module.ts
```

This is where our layout gets built!

Let's now see a diagram of how the `app` folder works:



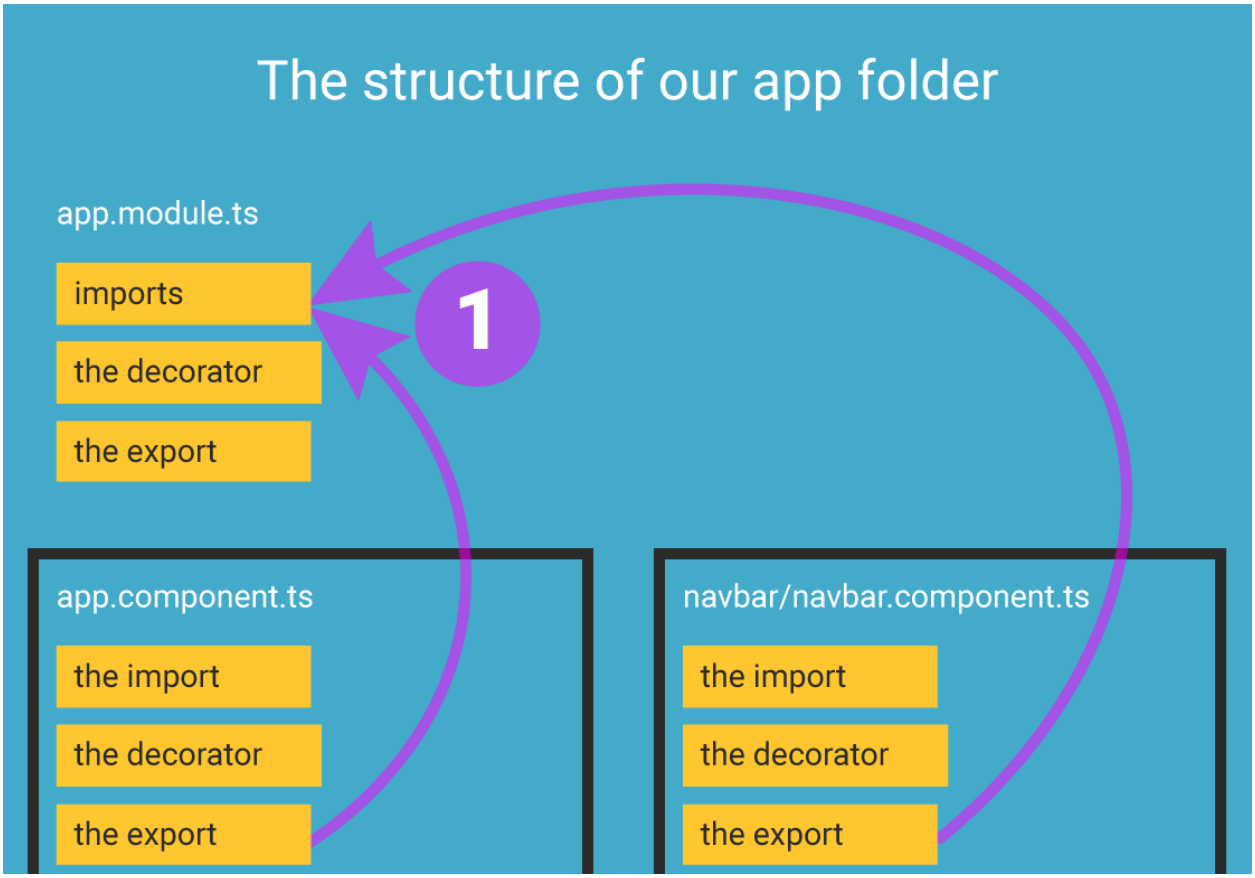
The structure of our app folder

#1: The app module imports all the components

Obviously, the `app.module.ts` file imports all the components. Currently there are two:

1. the app component
2. the navbar component

Let's see that again:

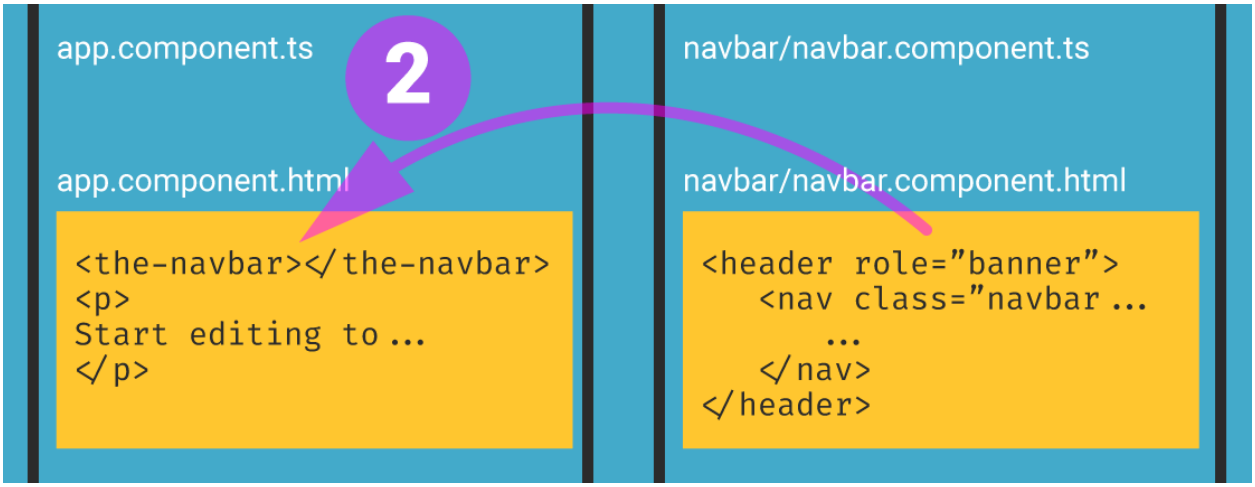


The structure of our app folder part 1

#2: The app.component.html imports all the other HTML files

The `app.component.html` file imports all the other HTML "partial" files, from all the other components.

Currently, there's only one single partial file, located inside the `navbar` folder, so it's the only one that gets imported:



The structure of our app folder part 2

This is a very important point to make, so let's repeat it once again, even if it might sound redundant: In Angular, the `app.module.ts` imports all the exported component classes, and the `app.component.html` imports all the other HTML partial files.

The `app.module.ts` ***imports classes.***
The `app.component.html` ***imports HTML.***

Inspecting the app.module.ts file

Let's now see what's going on inside the `app.module.ts` file:

```
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3  import { FormsModule } from '@angular/forms';
4
5  import { AppComponent } from './app.component';
6  import { NavbarComponent } from './navbar/navbar.component';
7
8  @NgModule({
9    imports:      [ BrowserModule, FormsModule ],
10   declarations: [ AppComponent, NavbarComponent ],
11   bootstrap:    [ AppComponent ]
12 })
13 export class AppModule { }
```

In the code above, on lines 1-3, we import some basic functionality that makes Angular work. No need to focus on that now.

On lines 5-6, we import the `AppComponent` and the `NavbarComponent`. These class names were exported from the `app.component.ts` and `navbar/navbar.component.ts` files, respectively.

On lines 8-12, we're specifying the module's meta data in its `@NgModule` decorator. This is similar to what we already discussed earlier when we saw how the `@Component` decorator works.

Inside the `@NgModule` decorator, on line 9, we use the modules imported in lines 1-3.

Still inside the `@NgModule` decorator, on line 10, we put all the imported classes into the `declarations` array:

```
1  declarations: [ AppComponent, NavbarComponent ]
```

This is the line where we'll be adding more HTML partials as we add each section of our layout in a new component folder.

Finally, line 11 inside the `@NgModule` decorator shows the entry point for the module: it's the `AppComponent`. Notice the bootstrap array:

```
1  bootstrap: [ AppComponent ]
```

It's important to understand that this line has got nothing to do with the Bootstrap framework! It's just a coincidence that Angular is using the same word for a different thing. Here, `bootstrap` means "the entry component for this module".

That's it for the `app.module.ts` file.

Next, let's inspect the `app.component.ts` file.

Inspecting the app.component.ts file

Here's the file:

```
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'my-app',
5    templateUrl: './app.component.html',
6    styleUrls: [ './app.component.css' ]
7  })
8  export class AppComponent  {
9    name = 'Angular';
10 }
```

On line 1, we see the import for the base Component functionality; this is needed for line 3, where we use the `@Component` decorator.

Inside the decorator, on lines 3-7, we specify the meta data: the selector name, the templateUrl, and the styleUrls.

Finally, we export the class of `AppComponent`.

There's nothing special happening here, so let's move along to the next file.

Inspecting app.component.html

Inside this file, we're just importing the one HTML partial that we currently have:

```
1  <the-navbar></the-navbar>
2
3  <p>
4    Start editing to see some magic happen :)
5  </p>
```

We've already explained earlier, in step 2 of the diagram. The navbar component's custom HTML tag is `<the-navbar>`, and we can just add it here like we did.

Inspecting app.component.css

Here's the CSS file:

```
1  p {
2    font-family: Lato;
3  }
```

The CSS file here is completely irrelevant. We don't even have to use it!

We can see what a component looks like ***without any CSS***, by inspecting the contents of the `navbar` folder, and its two files: `navbar.component.html` and `navbar.component.ts`.

Inspecting navbar.component.ts

Here we can see code that we're already familiar with (the import, the decorator, and the export):

```
import { Component } from '@angular/core';

@Component({
  selector: 'the-navbar',
  templateUrl: './navbar.component.html',
  // styleUrls: [ './navbar.component.css' ]
})
export class NavbarComponent  { }
```

Above, we can see the `styleUrls` array commented out. That means that the TypeScript / JavaScript parser will completely ignore it! We could have just deleted it if we wanted to - having it commented out is the same as if we did.

Inspecting navbar.component.html

This file is just plain HTML that we copied from our HTML layout, the one we built in the previous chapter. Since it's just plain HTML, there's nothing special to see here.

With this, we're done inspecting our layout.

We've covered a lot of ground in this chapter.

We discussed the problems we're facing building layouts the "old-school" way.

We found one possible solution, using Angular and Stackblitz, and we listed the reasons why do it.

We also discussed the Stackblitz interface, so that we're able to better use it.

We discussed the very basics of how Angular works too. We covered just enough to be able to separate our layouts into multiple components, and then re-assemble them inside the app module.

Rather then wrapping up this chapter, we'll look at another Stackblitz Angular app: this one will show the completed *Movies & Typography* layout, fully ported to Angular 8.

Inspecting the completed layout in Angular 8

The code for the layout in the Stackblitz editor can be seen at this link[77].

The completed layout's web page can be seen on this Stackblitz url[78].

Let's inspect the app's root:

[77] https://stackblitz.com/edit/angular-nbqv2p
[78] https://angular-nbqv2p.stackblitz.io/

```
src/
angular.json
package.json
```

Obviously, the app's root is the same as what we saw in the previous Stackblitz app.

If we now look at the contents of the src folder, this is the structure we'll find:

```
app/
index.html
main.ts
polyfills.ts
```

Again, exactly the same like in the previous Angular app.

Like we mentioned before, our layout - the app itself - lives in the `app` folder, so let's inspect it:

```
app-component/
blog-posts-component/
footer-component/
heading-component/
navbar-component/
app.module.ts
```

Here, the structure is slightly different from what we saw before. Hopefully, organizing our folders and files this way makes it easier to understand what's going on.

First of all, we've move the component class and the template HTML files for the `AppComponent` into its own subfolder, named `app-component`.

Next, we've added all the sections of the web page as separate components:

- the navbar component
- the heading component
- the blog-posts component, and
- the footer component

As we've learned earlier, the role of the AppComponent's HTML template file is to use the actual custom HTML tag we specified in each of the above components, and thus this file, `app-component/app.component.html`, looks like this:

```
<the-navbar></the-navbar>
<the-heading></the-heading>
<blog-posts></blog-posts>
<the-footer></the-footer>
```

Like we mentioned earlier in this chapter, the `app.module.ts` file imports all the class names that get exported from each individual component class files:

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';

import { AppComponent } from './app-component/app.component';
import { NavbarComponent } from './navbar-component/navbar.component';
import { HeadingComponent } from './heading-component/heading.component';
import { BlogPostsComponent } from './blog-posts-component/blog-posts.component';
import { FooterComponent } from './footer-component/footer.component';

@NgModule({
  imports:      [ BrowserModule, FormsModule ],
  declarations: [ AppComponent, NavbarComponent, HeadingComponent, BlogPostsComponen\
t, FooterComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

Looking at the code above, we can see two groups of imports: the first group includes imports that are compulsory in any Angular app, and the second group imports the actual components we made for our specific Angular app on Stackblitz:

```
import { AppComponent } from './app-component/app.component';
import { NavbarComponent } from './navbar-component/navbar.component';
import { HeadingComponent } from './heading-component/heading.component';
import { BlogPostsComponent } from './blog-posts-component/blog-posts.component';
import { FooterComponent } from './footer-component/footer.component';
```

Next, inside the `@NgModule` decorator, we're declaring all the components our module is using, listed in the `declarations` array:

```
declarations: [
    AppComponent,
    NavbarComponent,
    HeadingComponent,
    BlogPostsComponent,
    FooterComponent
]
```

Notice that the `declarations` array above is formatted slightly differently than in the source code; this is done only to improve the readability, but you can freely format your Angular apps this way, without problems.

Finally, let's just mention one little issue on Stackblitz: currently, in October 2019, there's no way to serve static assets, such as images, in Angular apps on Stackblitz.

While the Stackblitz team might resolve this issue quickly, we are employing a simple work-around: we are hot-linking images directly from the Unsplash.com website, and this is currently working fine.

The only issue here is that the images' width and height ratios are different from those we used in our *Movies & Typography* layout from the previous chapter. This can be easily solved in two ways:

1. We could either format our photos' appearance using CSS, or
2. We could hot-link images from the codingexercises.com website, or another website, as we see fit

Regardless of how we do it, this is an easy fix, so we will not go into the details here.

That's it for this chapter. In the next one, we'll build another Bootstrap layout in Angular.

Chapter 8: Build another Bootstrap layout in Angular

In this chapter we'll cherry-pick Bootstrap components to quickly build a Bootstrap 4 layout with Angular 8.

Up to this point in the book, we've built seven layouts:

1. A basic demo of containers
2. A shoe-themed demo of rows
3. Another shoe-themed site, using components. We saw how to make it responsive with the help of meta tags. This layout is the first one that actually started looking like a real layout.
4. Based on the official Album layout, we've built a custom Album layout.
5. A pricing table layout, based on a premium theme's pricing layout
6. A tweaked Checkout form layout, based on the official one
7. A typography-focused layout

The goal is to build 20 layouts in this series, but we're actually going to see the final missing peace of the puzzle, which will enable us to build many more layouts than 20. Before we do that, let's go back to see what we've covered so far.

We started easy, with containers and contextual colors. We learned about rows and responsive columns.

We then introduced Bootstrap 4 components.

We learned how to use the layouts found on the official Bootstrap examples page to build our first customized Bootstrap 4 layout.

We next saw how to copy features from premium themes and then tweak them to truly make them our own.

We got to know the Koala app for easy SCSS to CSS compilation - and we saw how to quickly update layout examples on the official docs by changing the values of SCSS variables in Bootstrap 4.

We started using the Brackets code editor and combined the techniques learned up to that point to build a completely custom layout, *Movies & Typography*. We learned how to: import Google fonts, revisited containers, talked about mockups, worked with Bootstrap's utility and spacing classes, discussed columns, responsive images, and source ordering.

Finally, in the previous chapter, we **introduced Angular as a way to improve our process**. We saw how to avoid repeating ourselves with Angular. We introduced modularity. We further simplified working with SCSS, using Angular.

In this article we won't build a whole new layout. We'll just take a few components from the official documentation, and whip up a layout.

Here are the steps we'll take:

1. Start building a new Angular app on Stackblitz
2. Split each section of the layout into its' own component, then import them via the app module and app component's html file

Let's get started!

8.1. Start building a new Angular app on Stackblitz

In the previous chapter we covered how to use Stackblitz in detail. We also covered how to split and how to import sections of a website in Angular.

Thus, in this chapter, let's just speed through our new Angular app setup, without too much in-depth discussion. If something is unclear, please refer to the previous chapter.

8.1.1 Setup a new Angular 8 app on Stackblitz

Let's open a brand new Angular 8 app on Stackblitz.

To track the changes to our code more efficiently, we'll use Github.

Since this is a beginner's book, I'm assuming you haven't heard of Github, so I'll keep it really simple. Github is just a way to take snapshots of an app at certain points in app development.

Which points exactly? It's completely up to us! Whenever we feel like it, we can take a snapshot of our app, called a ***commit***. Github is using the Git system, built by Linus Torvalds of Linux.

So, once we take this snapshot of a page (i.e a commit), we can then save that change on Github. The place where all these commits live is called a repository. Thus, a repository on Github will have any number of commits for a given project, ordered chronologically.

This makes perfect sense, because a repository is like a time-machine for certain points in the development of your project.

8.1.2 Using Github on Stackblitz

It's really easy to get started with Github on Stackblitz, because it's integrated right on the Stackblitz interface. In this article however, I'm not suggesting that you even sign up. You can just inspect the commits I've made while building this layout.

Thus, the first commit of this project is the default Angular 8 app on Stackblitz[79]. The following link will open my own repository for this Stackblitz app[80].

This first commit might look a bit overwhelming, because it simply lists all the files added to the app when it was started. As we've learned in the previous article, there's quite a lot of files in a default starter Angular app.

However, every new commit is going to be much smaller, and you'll be able to easily see what happened at each commited update to our app.

We'll see that next, when we remove redundant files from our default starter Angular 8 app.

8.2. Removing redundant files

We'll begin by removing the Hello component.

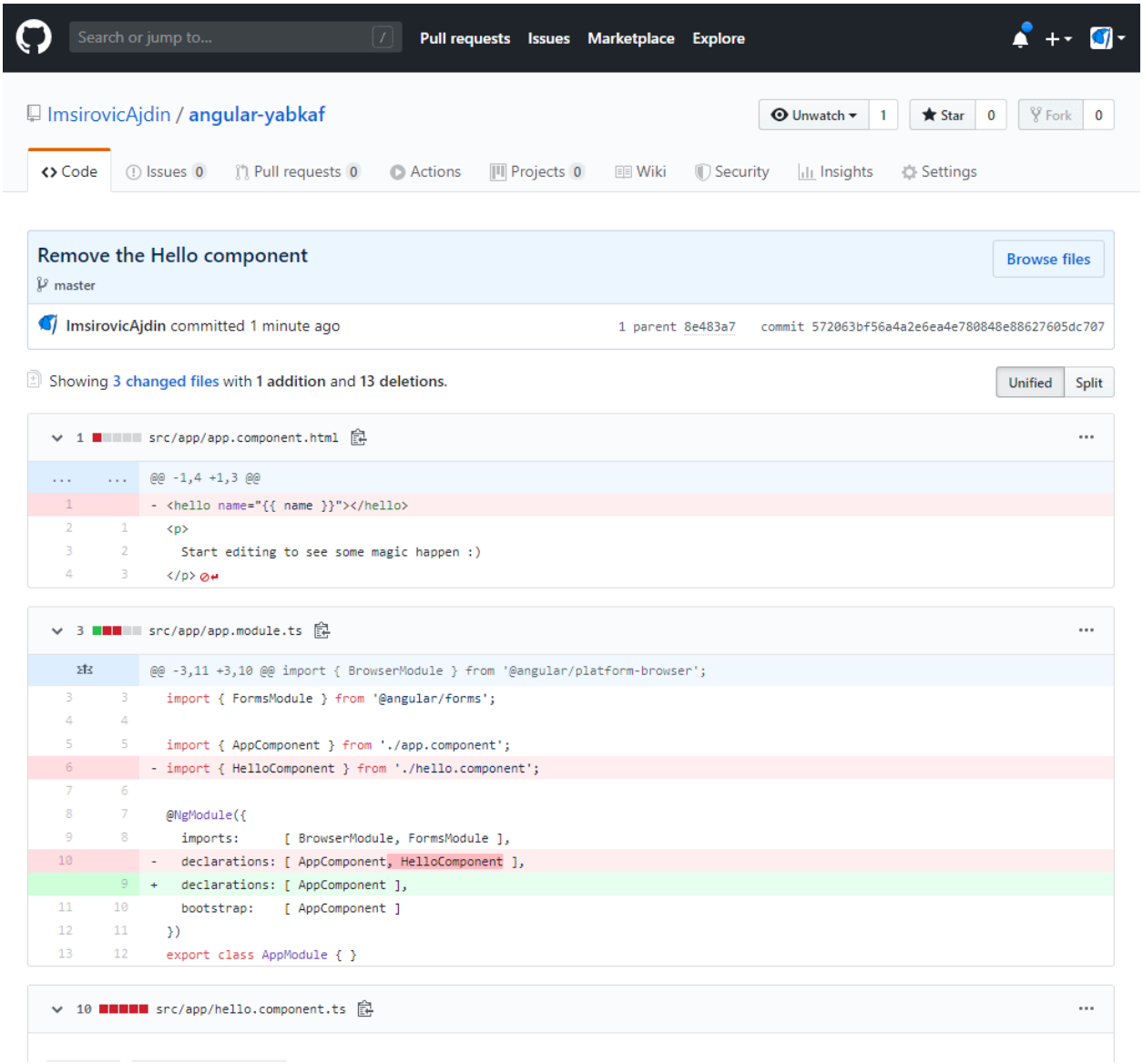
We will:

- delete `hello.component.html`
- remove it from the `declarations` array in `app.module.ts`
- remove its import from `app.module.ts` too
- delete the `<hello>` custom tag from `app.component.html`

Rather than explaining every single line of code where we deleted traces of the Hello component, let's just see the commit for it[81].

Here's a screenshot of the above-linked commit:

[79] https://github.com/ImsirovicAjdin/angular-sykrmr/commit/5ee9ad301aa04a9caa86f232abdac732846fe576
[80] https://github.com/ImsirovicAjdin/angular-sykrmr
[81] https://github.com/ImsirovicAjdin/angular-sykrmr/commit/57a8536862f030177f00adef80c1484da255efb1

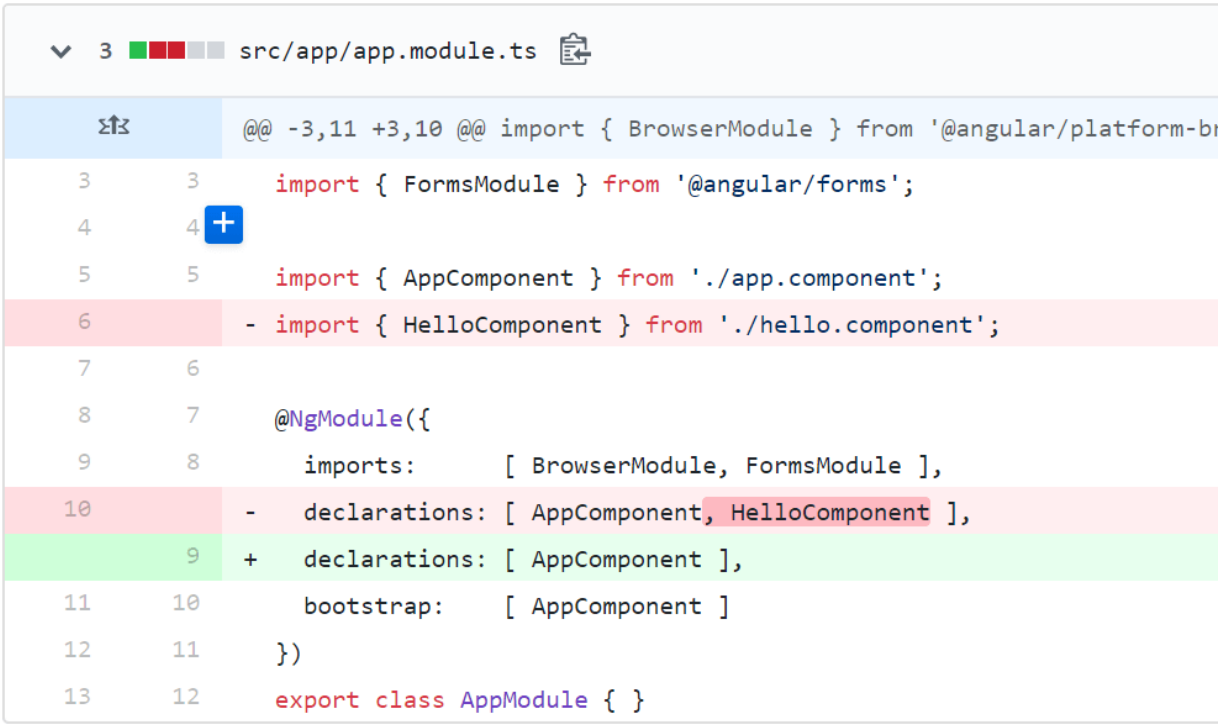


A screenshot of a commit titled Remove the Hello component

Now it's much easier to understand what's happening in our app at this moment in time.

Looking at the above screenshot, it says that there were 3 changed files. The lines and bits of code that were removed are highlighted in red. The additions are highlighted in green.

Looking at the declarations array inside `app.module.ts`, you will see both a removal and an addition:

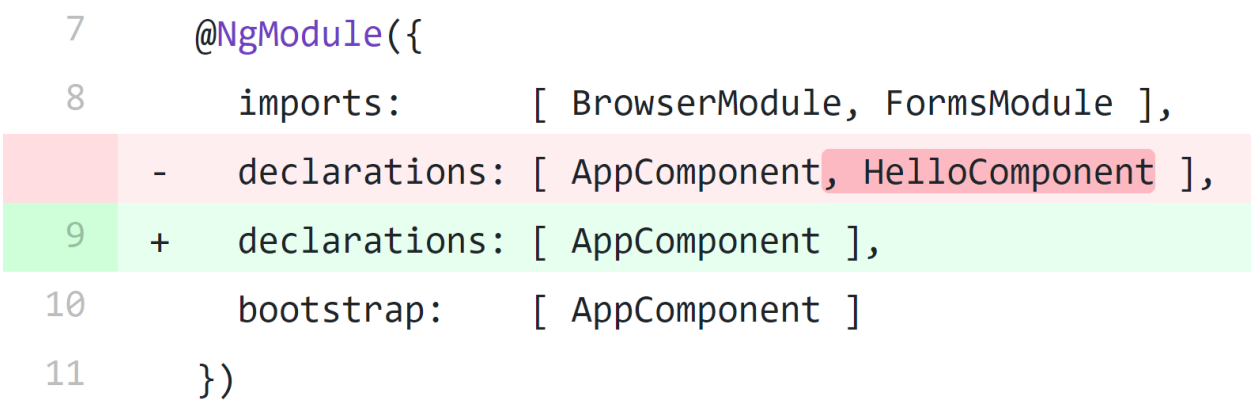


A screenshot of a commit with additions and removals

You might think: "This is wrong, we didn't add anything, we just removed code!", and you'd be right. That's exactly what happened.

But Git doesn't look at it that way.

If you have removed only a part of a line of code rather than the entire line, Git will see it as both an addition and a removal:



A screenshot of a commit with both a removal and an addition

If you look at the above screenshot, it helps to note the line numbers. Or, rather, the line number. Line number 9, to be precise.

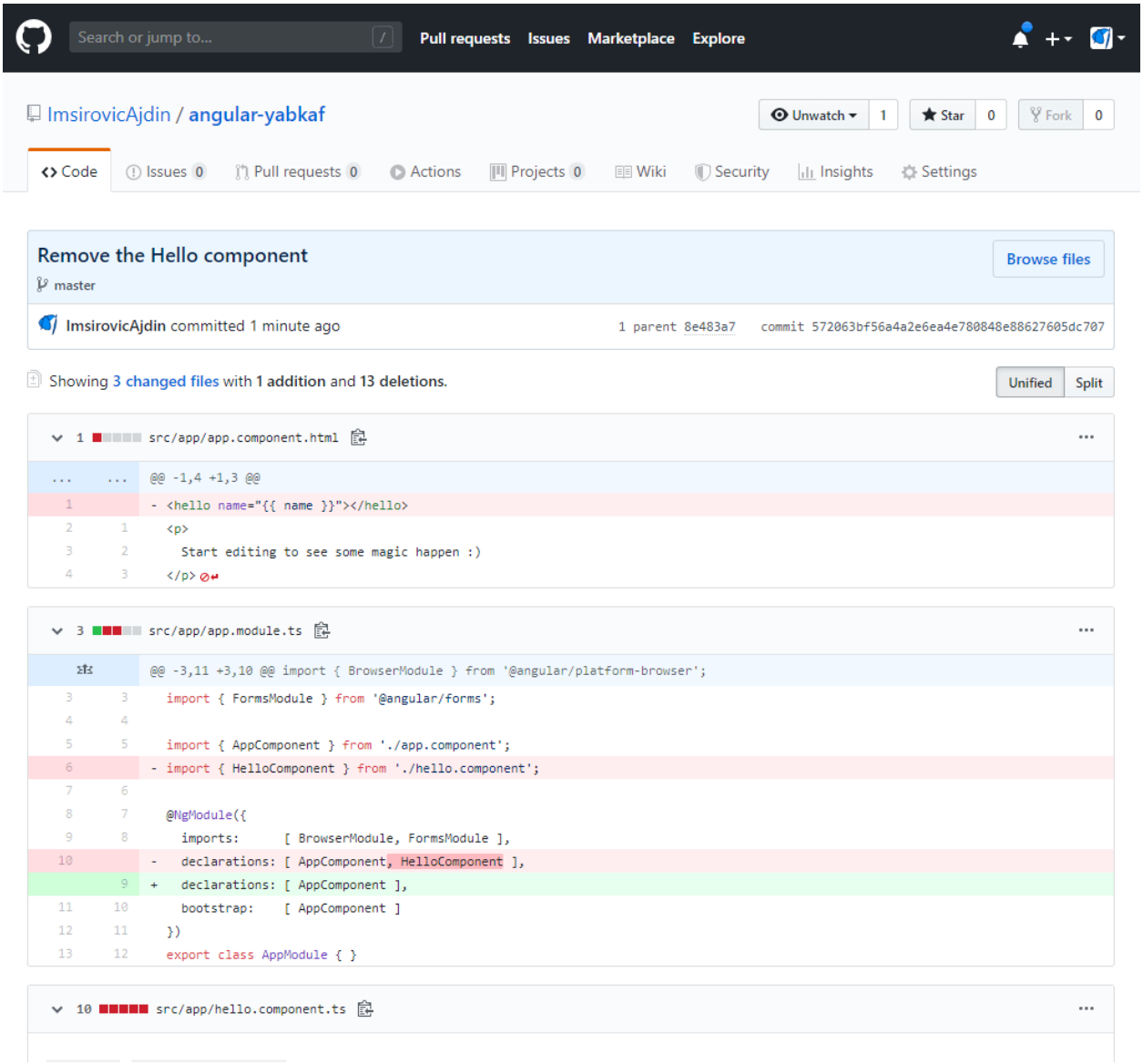
The line number 9 is split on two lines:

- the first, red line, shows the code before the removal (with the removed item highlighted with a darker red background)
- the second, green line, shows the code after the removal

If you think about it, Git's way of looking at deletions might actually make more sense.

The takeaway here is that if you see a red line and then a green line, and there's only one line number for both lines, that means that a bit of this line was removed in the code. Another way to look at it: the green line will always be shorter - by the number of characters that got removed in the commit.

Let's look at the entire commit screenshot again:

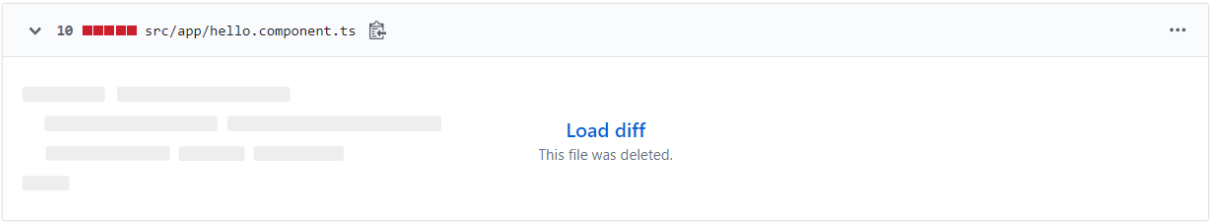


A screenshot of a commit titled Remove the Hello component

Looking at it again, we see there were 3 changed files:

1. the `app.component.html` file, where we removed the line that starts with `<hello>...`
2. the `app.module.ts` file, where we removed the import and the `HelloComponent` entry in the `declarations` array
3. the entire `hello.component.ts` file

As for the third file, the `hello.component.ts` file, you can see it's deleted - it says so right under the "Load diff" link in the file's panel (at the bottom of the commit):



Load diff, this file was deleted

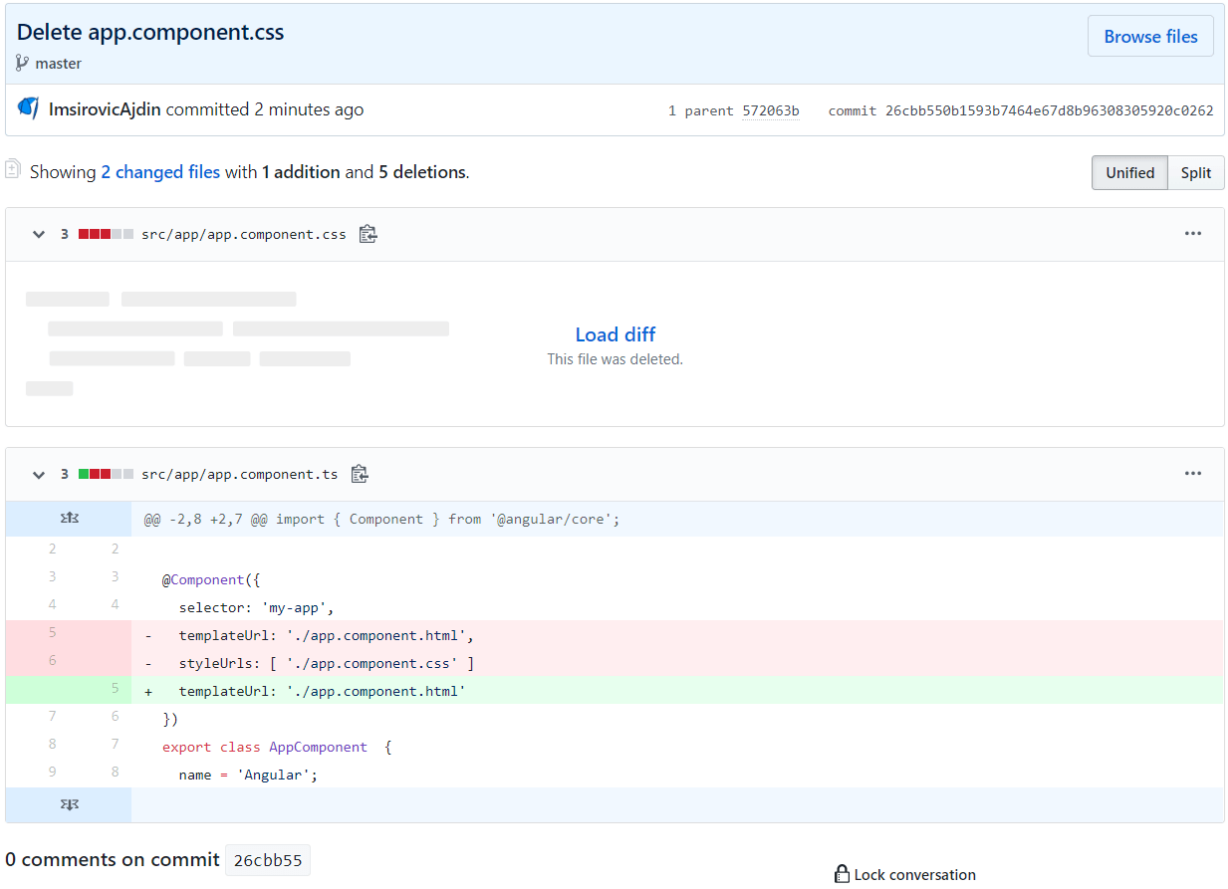
If you click on the "Load diff" link, you'll see the entire deleted file's code. That's sometimes useful, just to inspect the code that was removed, if you're trying to better understand what happened at a certain stage of an app's development.

Great, so we've just seen how Github and Git commits work and can be viewed.

We still have another file to remove: the `app.component.css` file. Here's a commit with this file removed. The commit is titled Delete app.component.css[82].

If you inspect the changes in this commit, you'd see the following:

[82] https://github.com/ImsirovicAjdin/angular-sykrmr/commit/99ee09238164bc1865ddfab395db6d9e9231f6cb



Another screenshot of a commit

When we said we're just going to delete the `app.component.css` file above, you might have thought that that's literally what was going to happen. Well, that is what happened, but we also had to delete the `styleUrls: [./app.component.css']` line from the `@Component` decorator of the `app.component.ts` file.

This shows us the utility of commits in Git: A nicely-worded commit, with only a few changes to our codebase, can greatly improve the understanding we get from what was happening with our app at any given time.

This goes back to what we discussed in the previous article: speed.

Properly worded commits, that don't involve too many changes, are great to speed up our analysis of what was going on at any given time during the coding of our app.

It's also great for learning - such as in this chapter - to quickly see what changes needed to be made to achieve a certain web layout feature.

That's why we can now speed things up, using commits and commit messages as our guides.

8.2.1 Moving app.component.html and app.component.ts to a new folder

Just like in the previous chapter, we'll take our app component and move it into a separate folder[83], so that only the `app.module.ts` stays in the root of the `app` folder.

The new folder that we'll move our app component to is called `app-component`.

8.3. Adding Bootstrap from a CDN

In this section, we're adding Bootstrap from a CDN[84].

8.3.1 Adding the navbar component

Now we'll add the navbar component[85].

8.3.2 Adding the jumbotron component

Let's copy paste the first example from the official Bootstrap docs' jumbotron documentation[86].

We'll make a new component and use the copied HTML in its template file. The new component's commit title is: Add jumbotron component[87].

8.3.3 Adding the cards component

We're also adding a couple of cards components from the section titled Using grid markup[88], on the official docs. The commit title is Add the cards component[89].

8.3.4 Adding the footer component

In the previous chapter, we've seen how to add the footer component[90]. I enourage you to fork our current Angular app[91], and try adding the footer yourself.

Once again, feel free to have a look at our Angular app's repo[92].

[83] https://github.com/ImsirovicAjdin/angular-sykrmr/commit/ad67fe35d967aff92ce1d56184745b5fdab8beaf
[84] https://github.com/ImsirovicAjdin/angular-sykrmr/commit/97f57912129a497e4601402e0a9c974d1e544bf8
[85] https://github.com/ImsirovicAjdin/angular-sykrmr/commit/921e080209d8d22ab20a24eccb662f97cd950eeb
[86] https://getbootstrap.com/docs/4.3/components/jumbotron/
[87] https://github.com/ImsirovicAjdin/angular-sykrmr/commit/db7417d798886e38a865c493a5c009a139484723
[88] https://getbootstrap.com/docs/4.3/components/card/#using-grid-markup
[89] https://github.com/ImsirovicAjdin/angular-sykrmr/commit/ee84940b9641280832edbb92352bcc21a2c80a05
[90] https://www.codingexercises.com/guides/building-bootstrap-layouts-part-8#inspecting-the-completed-layout-in-angular-8
[91] https://stackblitz.com/edit/angular-sykrmr
[92] https://github.com/ImsirovicAjdin/angular-sykrmr

In this chapter, we've introduced Git and Github. We've also practiced quickly prototyping a Bootstrap layout by copy-pasting components from the official docs.

In the next chapter, we'll go back to a more old-school way of building Bootstrap layouts. We'll build one locally, and this time we'll use Git on the desktop.

Chapter 9: Build a Bootstrap 4 layout and track it with Git

In this chapter we'll build a Bootstrap layout locally and track it with Github for Desktop.

In this chapter we'll see how to use Github and Github Desktop while we build a brand new layout locally in Brackets.

Note that this chapter is Windows-specific, but if you are using a Mac I am pretty sure you can follow the same, or very similar approach to the one outlined here. If you're using Linux, you can install Github Desktop following the steps here[93].

First we'll look at registering on Github.

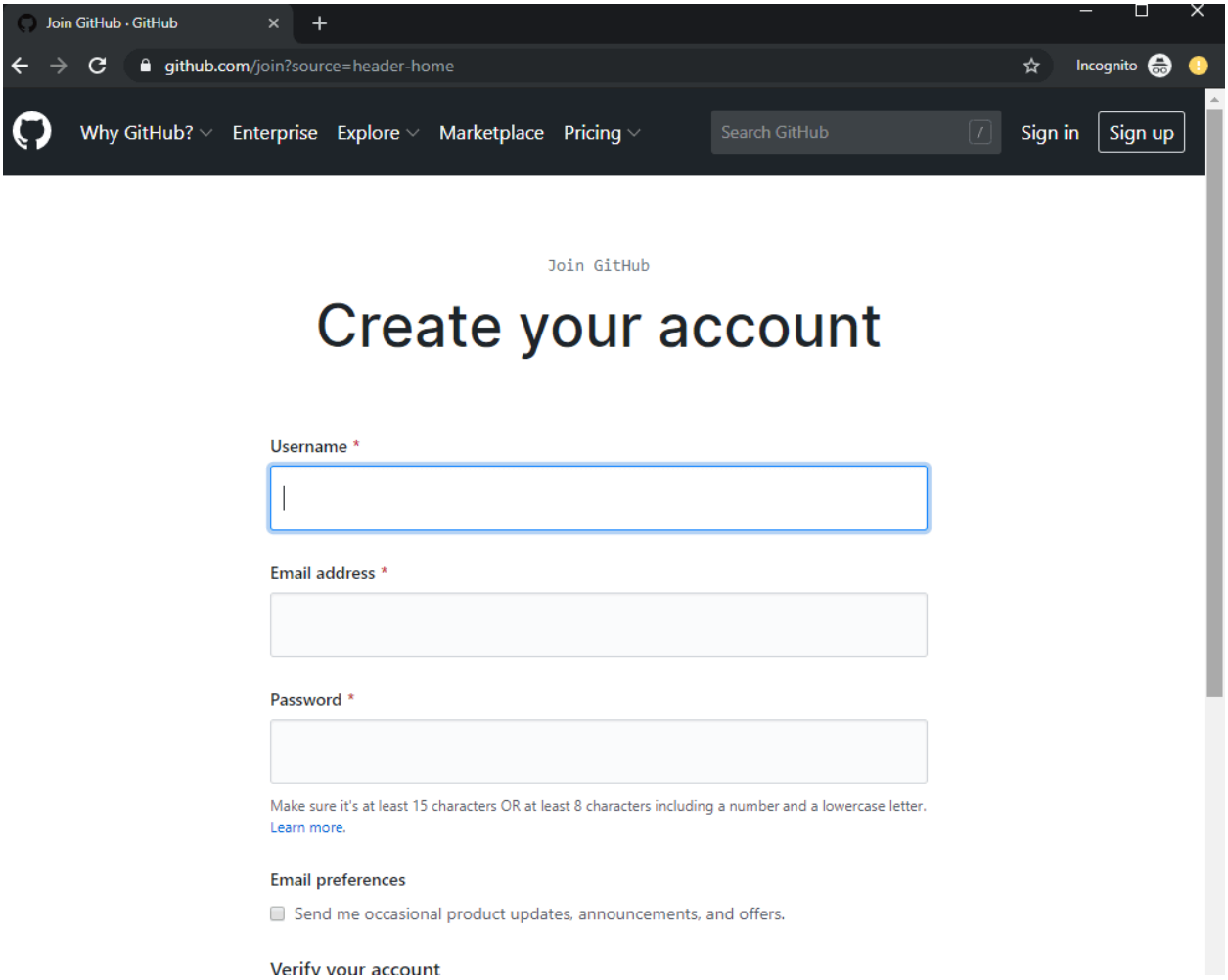
9.1 Register a new account on Github

Github was acquired by Microsoft for $7.5 billion on October 26, 2018. Github was founded in February 2018, running on Ruby on Rails.

If there ever was a success story, Github seems to be it.

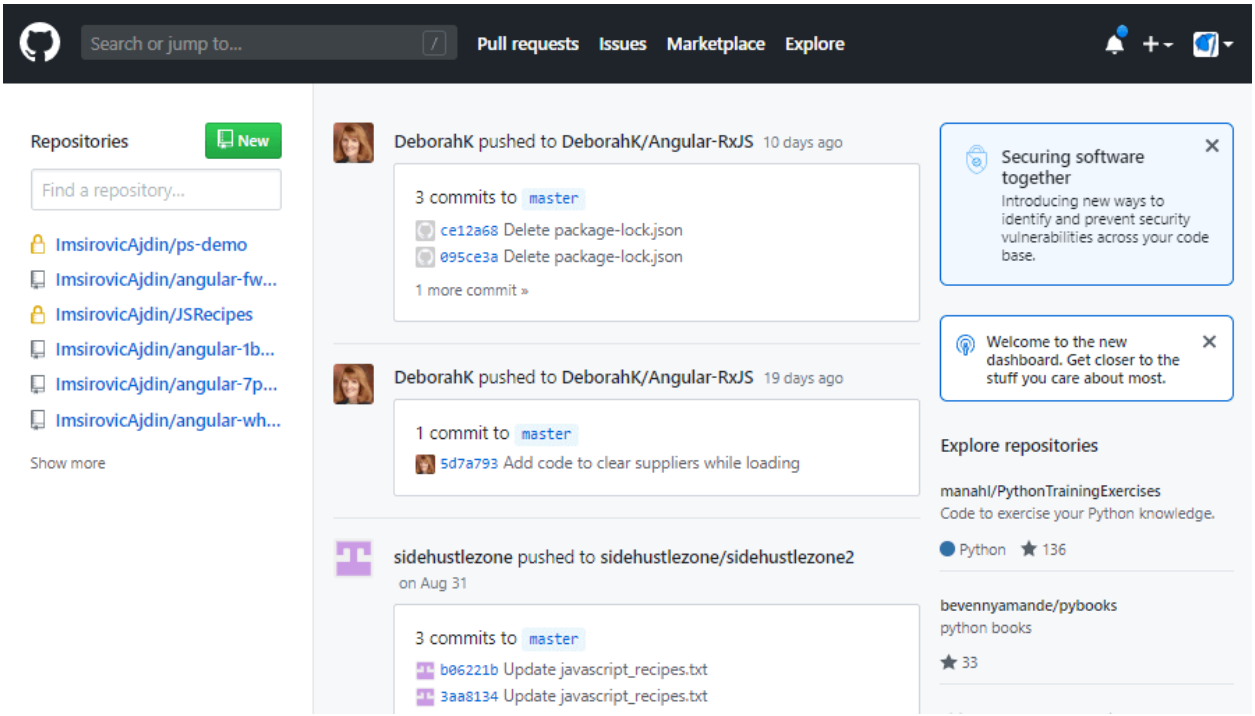
Let's register an account on Github and start discovering why it's so popular.

[93] https://gist.github.com/berkorbay/6feda478a00b0432d13f1fc0a50467f1



Github registration screen

Once you've registered and logged into Github, you'll see a dashboard, similar to this:



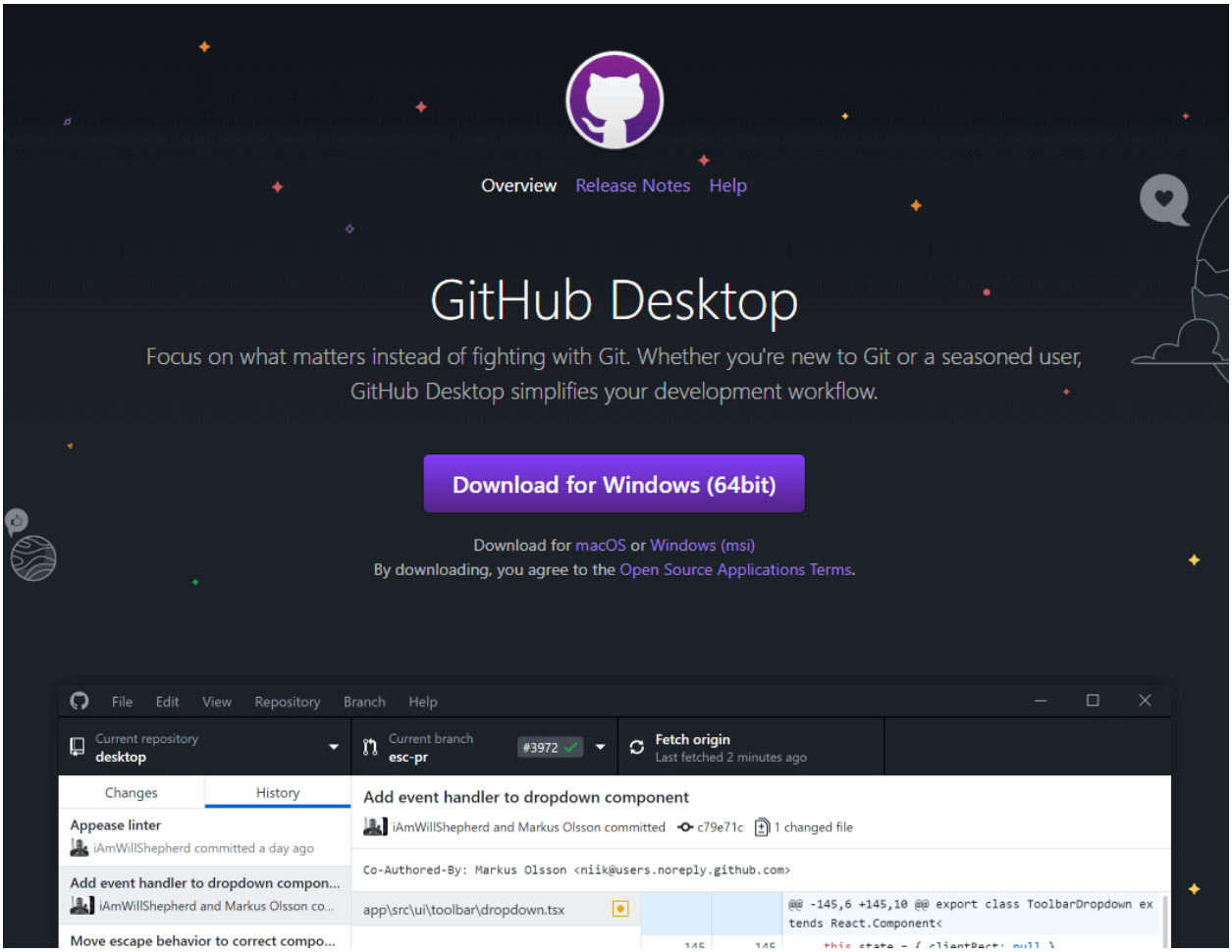
Github dashboard

9.2 Start tracking your code with Github Desktop

To track your code using Github, you *could* start a new repository straight from the Github dashboard. There are two quick ways to start a repository on Github: Press the "new" green button in the top left of the above screenshot, or click on the profile image in the top right, and then press Your repositories, which will open a screen with all your repositories listed. You can then find the "new" green button in the top right of the repository list header.

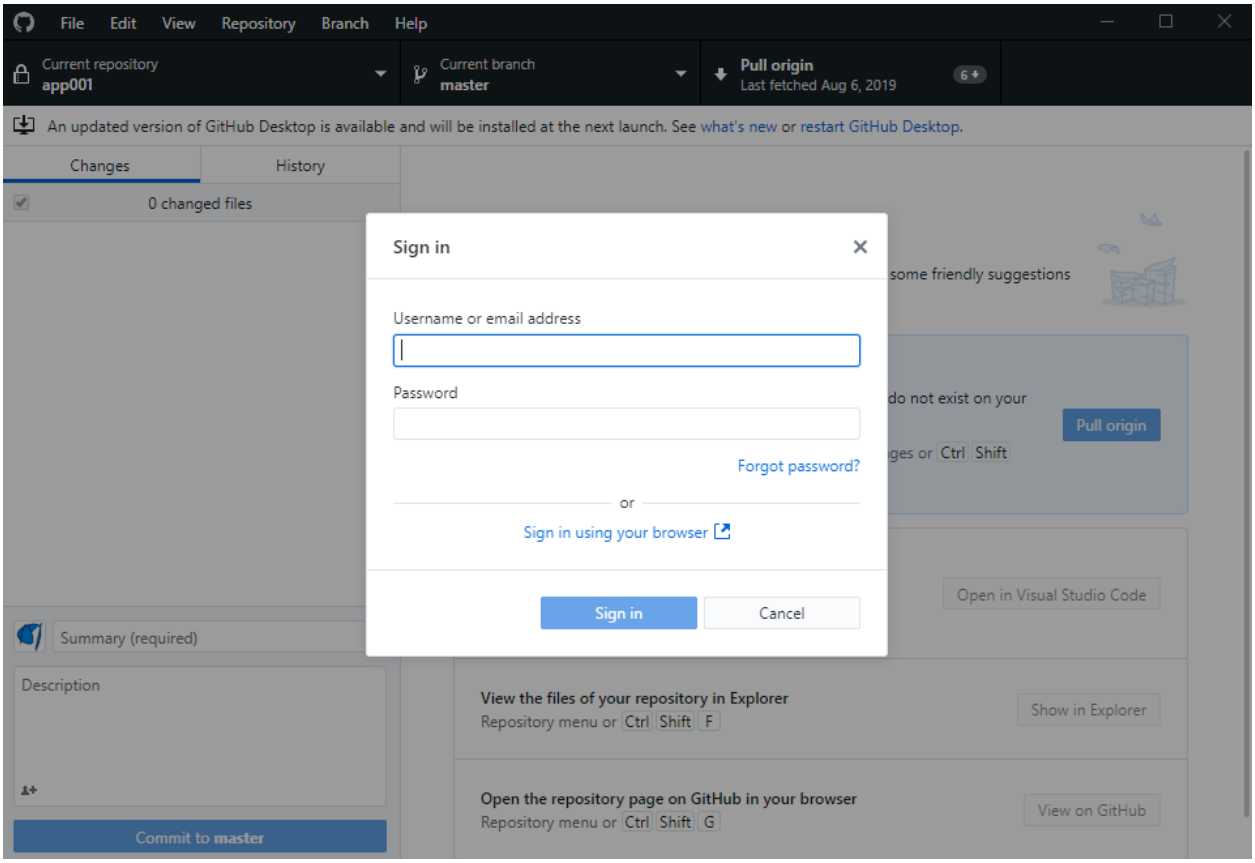
Alternatively, you can download Github Desktop[94], and start a repository using this app. Using Github Desktop is the easiest way to start using Github on your computer, so that's what we're going to do in this chapter.

[94] https://desktop.github.com/



Github desktop homepage

Once you've downloaded and started Github Desktop, it's going to ask you to log in.



Github desktop login

You can locate the sign in any time by clicking the `File` menu, then the `Options` sub-menu item. In the `Options` window, on the `Accounts` Tab, you have the information about the currently signed-in user. You can also sign out on this screen, so another user can sign in (if, for example, you're sharing a computer with another Github user).

9.2.1 Choosing a folder to track

Add a folder on your file system where you'll keep your Bootstrap 4 layout.

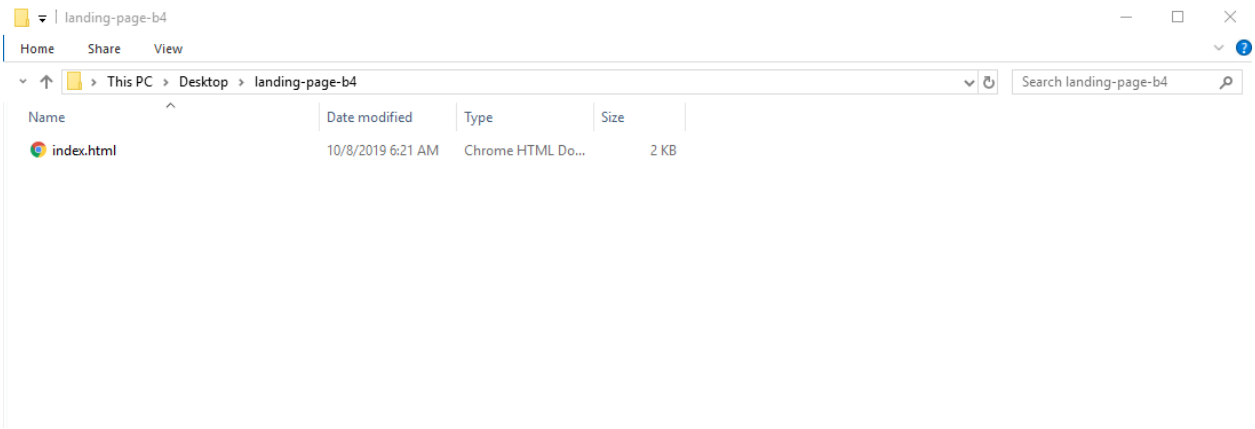
We'll call this folder "landing-page-b4".

Let's also add an `index.html` file inside of it, and add this file the code from the starter template[95].

Feel free to add the code to index.html using the Brackets code editor[96].

So, here is the landing-page-b4 folder in my file explorer:

[95] https://getbootstrap.com/docs/4.3/getting-started/introduction/#starter-template
[96] https://www.codingexercises.com/guides/building-bootstrap-layouts-part-7#choosing-the-code-editor

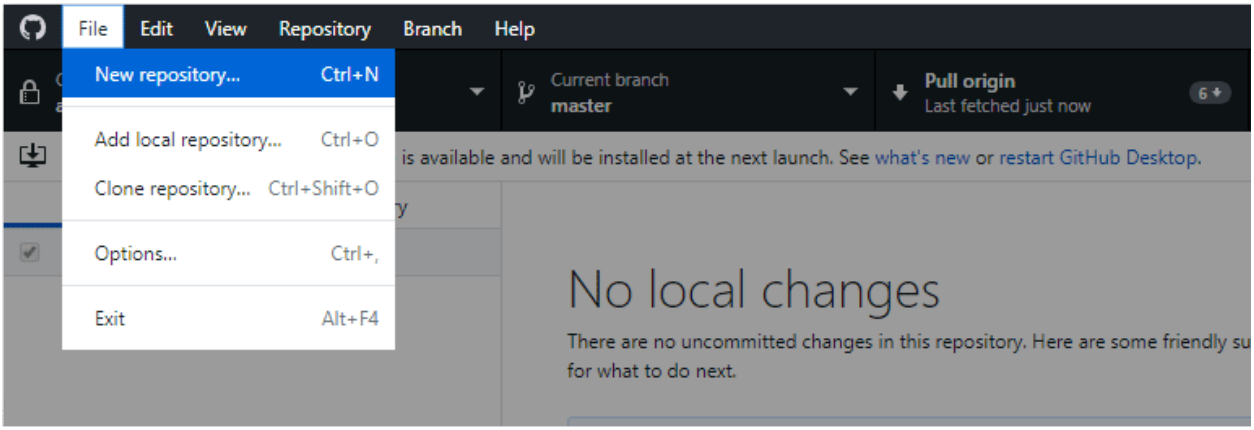


Screenshot of a folder in file explorer

9.2.2 Adding the folder to Github Desktop

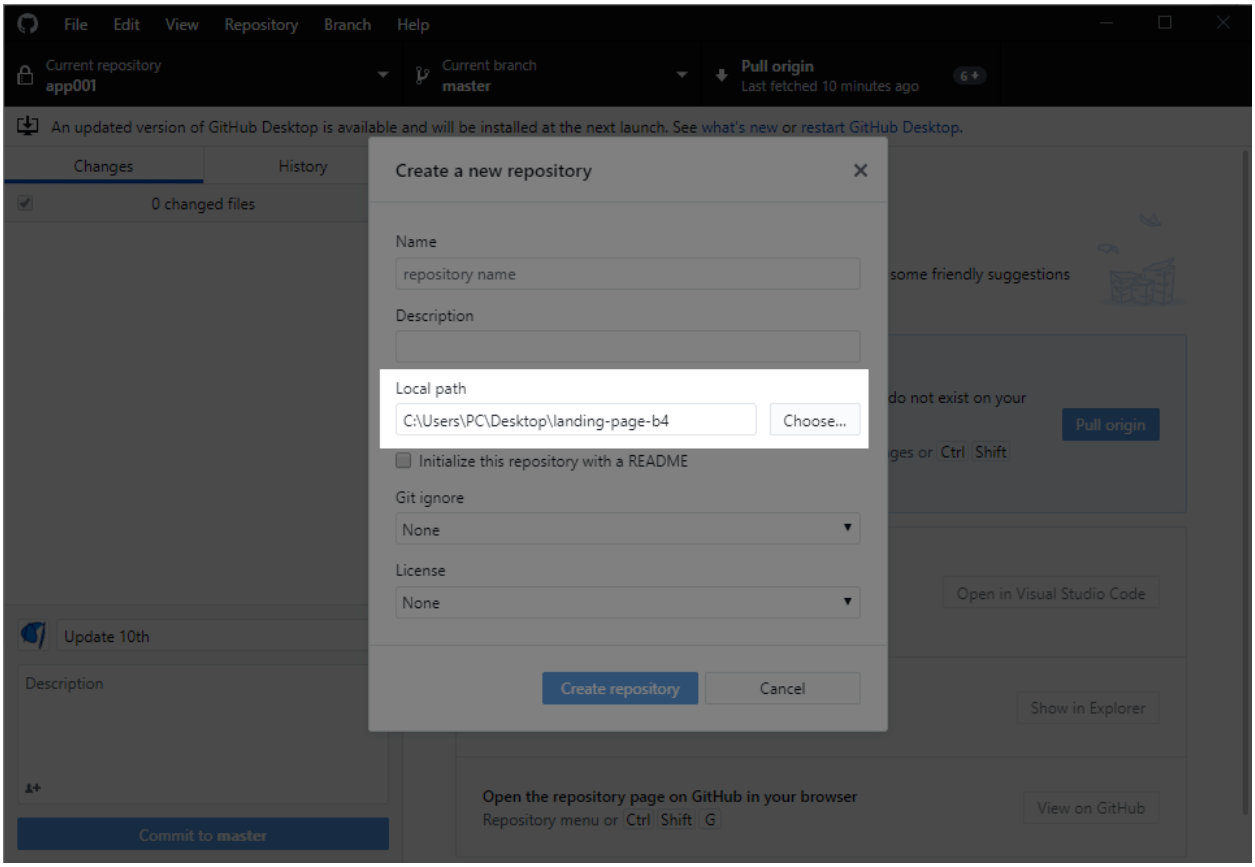
Now let's add the folder to Github Desktop.

Click `File` on the top menu, then `New Repository`.



Screenshot of adding a new repository from existing folder in GitHub desktop

Next, choose the local path for your new repo, pointing it to the folder that holds the Bootstrap 4 layuot we're building.



Choose local path for your new repo

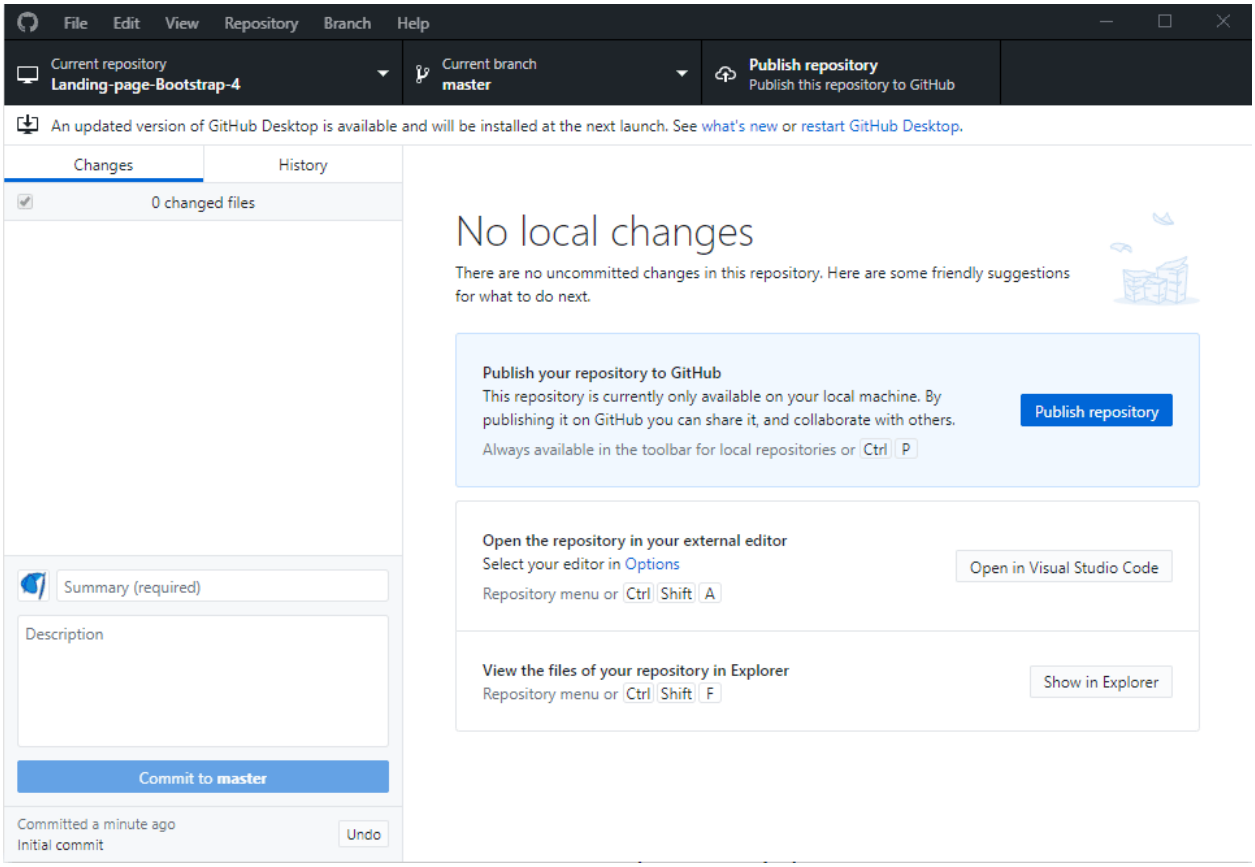
We should also give our repository a name.

Let's call it ***Landing-Page-Bootstrap-4***.

Now we'll press the ***Create repository*** button, and you're ready to publish it.

9.2.3 Publishing our repository

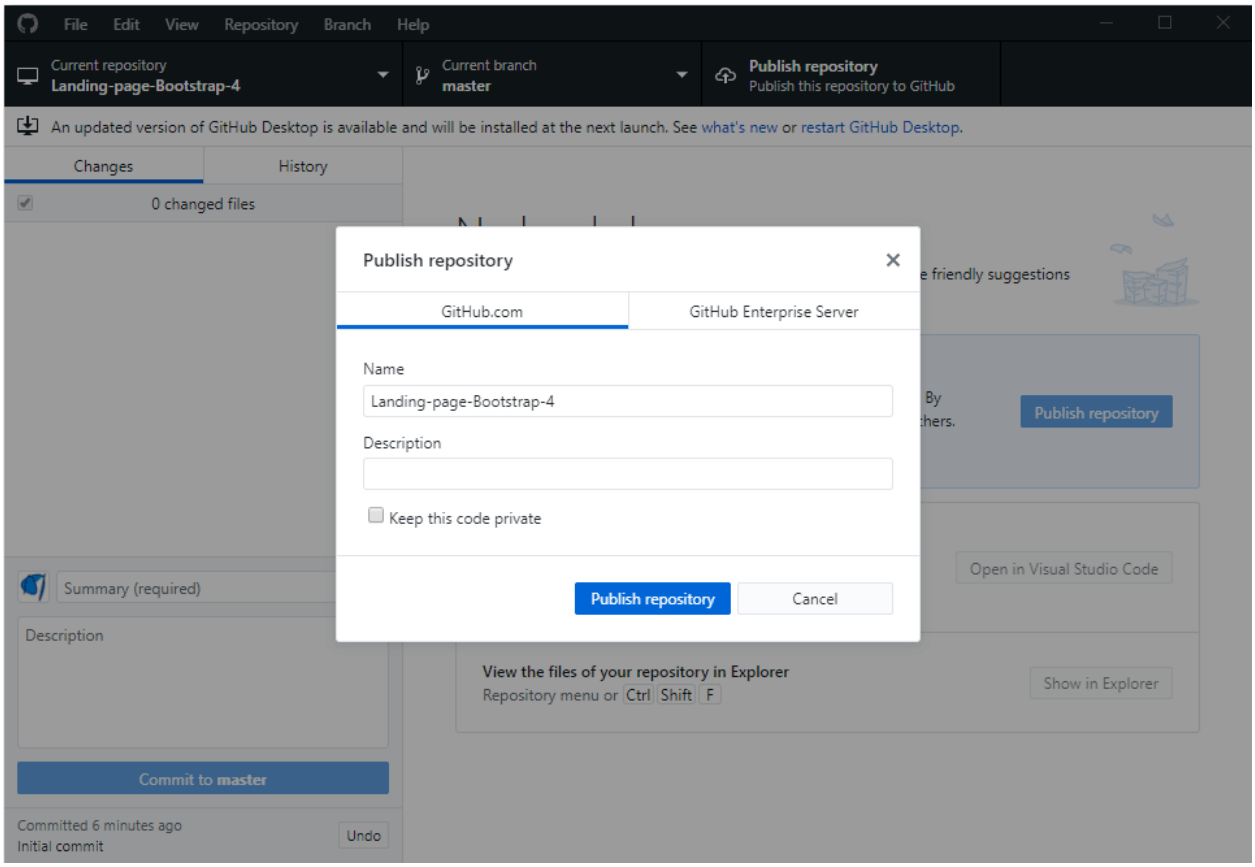
To publish our repository to Github from Github Desktop, at this point, we can just click the ***Publish repository*** button.



Publishing new repository to Github from Github Desktop

A modal window will appear, and here we can change our repository's privacy settings.

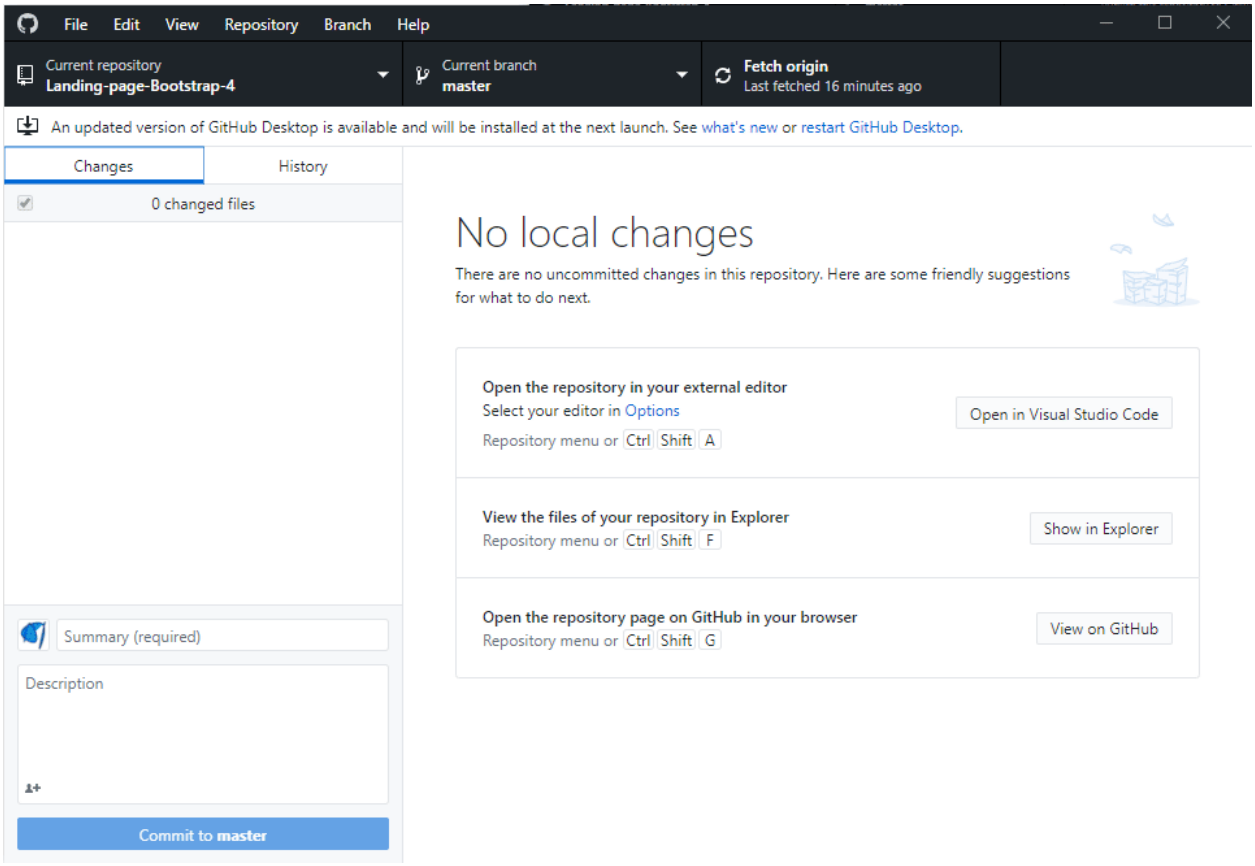
If we want to show our code to the world, we'll need to make sure to uncheck the ***Keep this code private*** option:



To share our code, we uncheck the 'keep this code private' option

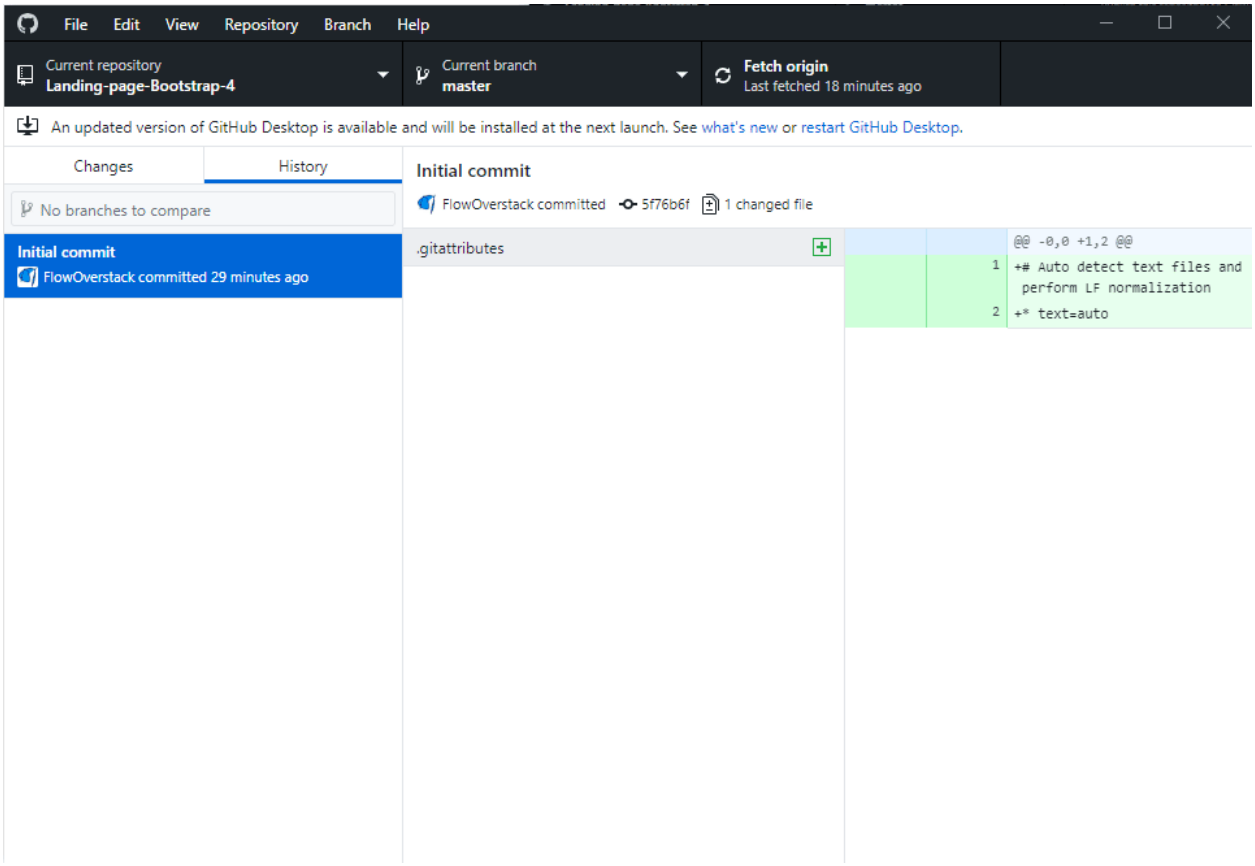
If we want to make it visible for everyone, we don't have to uncheck anything; we just confirm we want to publish the repo by clicking the ***Publish repository*** button again.

Once we've published our repo, we'll see the following screen.



The no local changes screen in Github Desktop

If we click on the history tab, we'll see the ***Initial commit*** with which this repository was published to Github.



The history tab in Github Desktop showing our initial commit

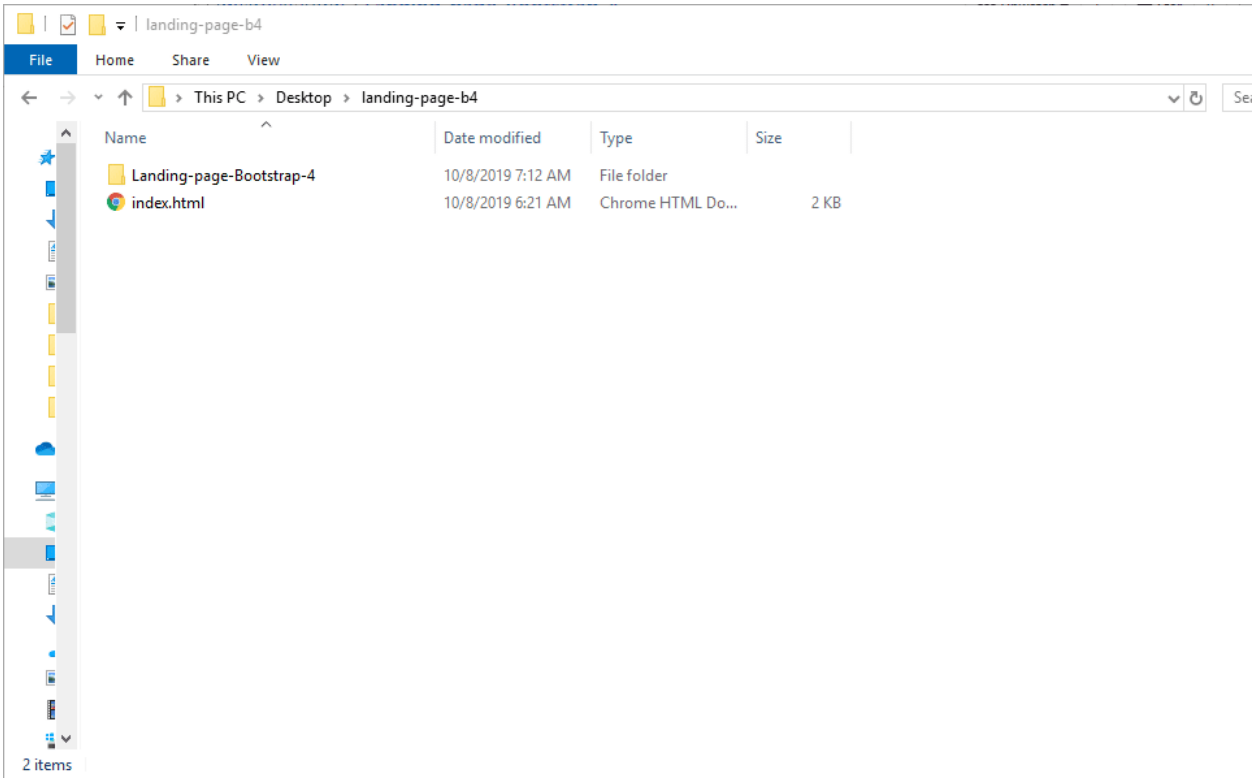
Let's now go back to Github to see the our repository there[97].

We see that our commit has only this weird `.gitattributes` file. This file is irrelevant now so we won't go into the details of why it's there.

But, where's our `index.html` file?

As it turns out, I've made an error. Have a look at this screenshot of my local landing-page-b4 folder:

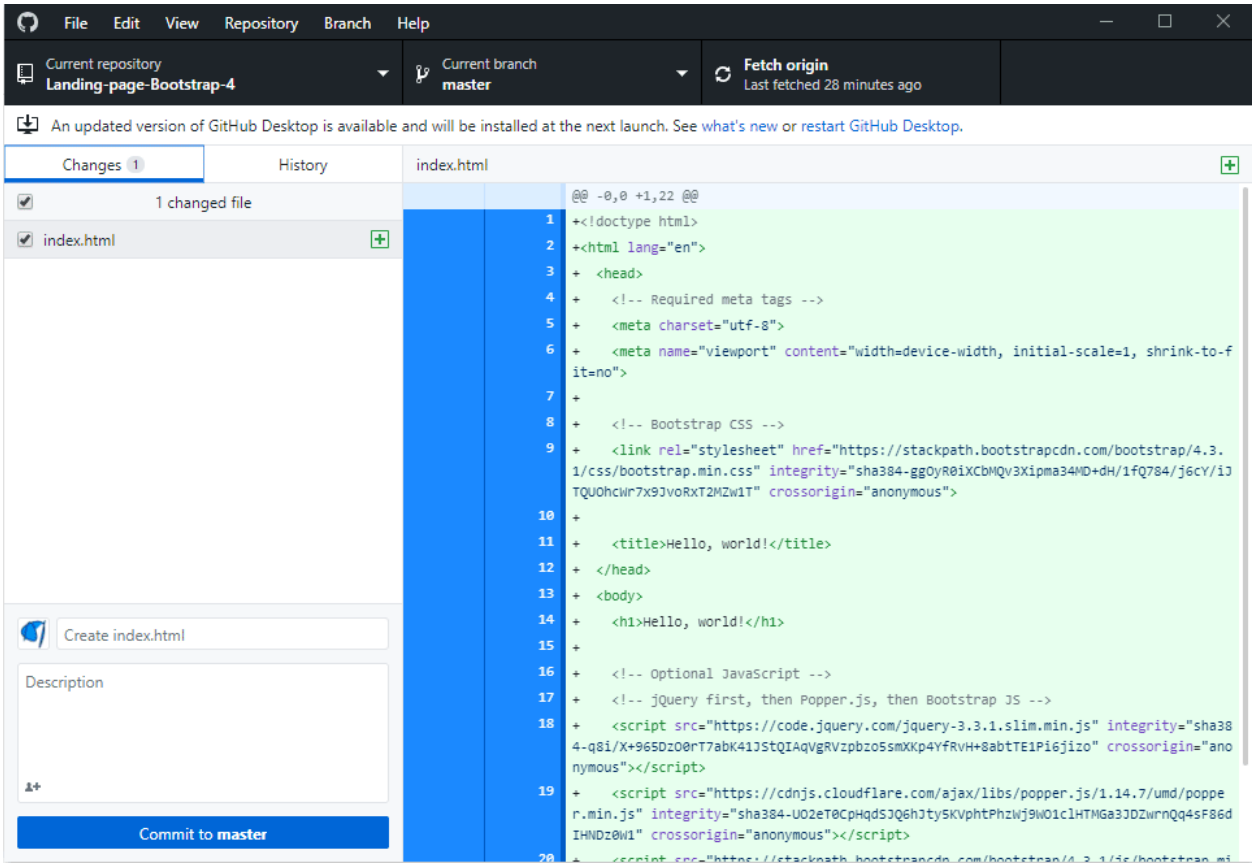
[97] https://github.com/ImsirovicAjdin/Landing-page-Bootstrap-4



Landing-page-Bootstrap-4 folder inside landing-page-b4 folder

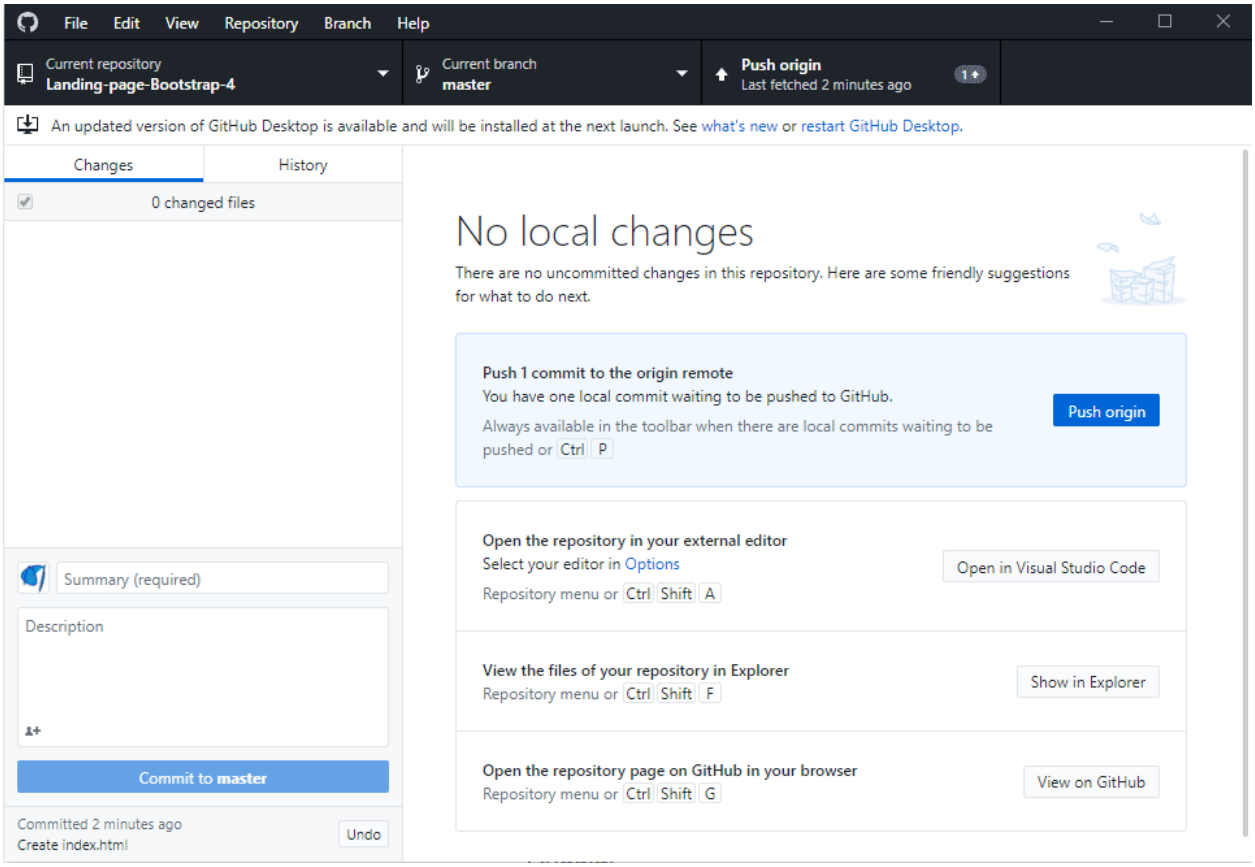
This is an easy fix: we'll simply cut and paste the `index.html` file into the `Landing-page-Bootstrap-4` folder.

Since Git is watching the changes to `Landing-page-Bootstrap-4` folder, it's going to notice the addition of `index.html`:



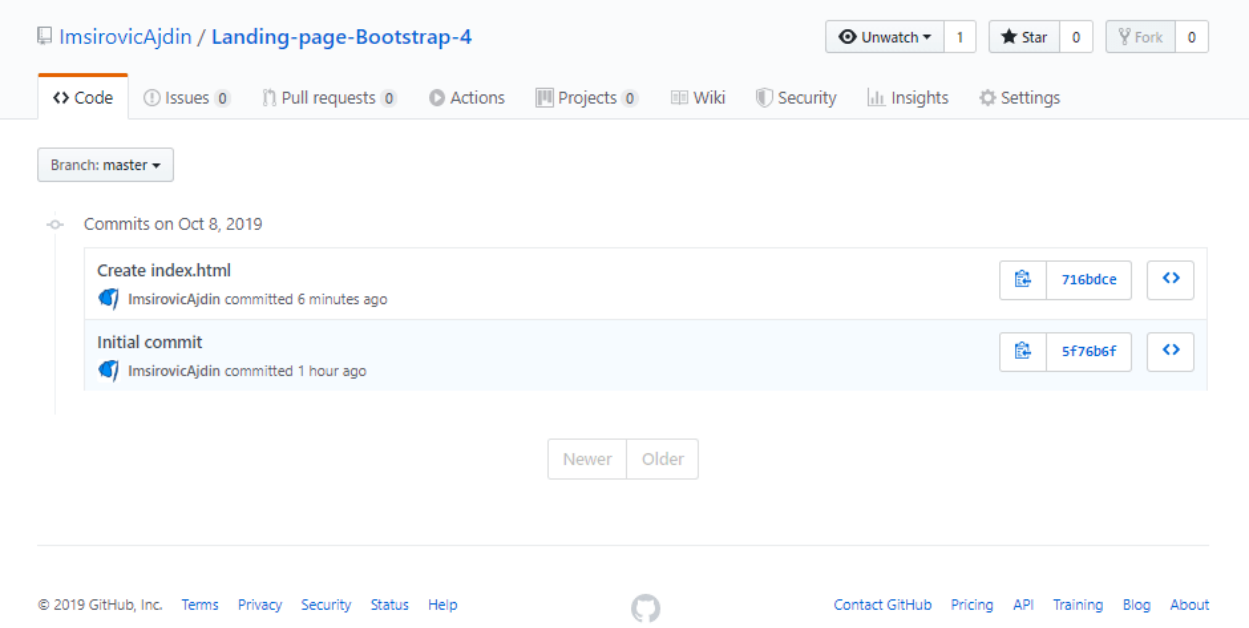
Changes are being tracked inside Github Desktop

Let's commit these changes now, by pressing the ***Commit to master*** button in the very bottom-left of the Github Desktop window. Once we do that, the ***Push origin*** button will appear. That means that we've commited the change on our local machine, and now we're pushing that change to the remote server (the Github server), referred to as the "origin".



After a commit, Github Desktop is ready to push origin

Let's see this update on the Github website.



The second commit showing on the Github website

If we click into the second commit message, Create index.html[98], we'll see the index file we've added.

Now we can reopen the `index.html` file inside the Brackets editor to continue working on it.

9.3 Building the landing page

In the previous chapter, we've discussed the benefits of using Git.

As we mentioned, one of the main benefits is speed.

Once you build a layout while tracking it with Git, you can "time-travel" to any point in time of the layout development.

For this "time-travelling" to be efficient, there is one rule of Git, and that is: commit small and often.

If you keep your commits small and your commit messages clear, that can do wonders for you once you revisit a repository, say, three months from now.

The added benefit of sticking to this approach is: if another person looks at your code, they'll find it easier to understand what's going on.

We'll see this in practice now, when we build the landing page.

Each new addition to the landing page will be saved as a separate commit. We'll add proper commit messages, so that once our layout is complete, we can look at the completed repository with all the commit messages.

[98] https://github.com/ImsirovicAjdin/Landing-page-Bootstrap-4/commit/716bdceaccb960bacace975b002096bec9ea06d1

Thus, we will instantly have a birds-eye view of what steps were needed to be taken to have completed the layout.

Let's get started!

9.3.1 Adding the header

We'll begin from the top of the page. Let's find a header we like, by browsing through the official premium themes[99].

I find the Incline Landing Page Set[100] interesting, so let's look at it closer[101].

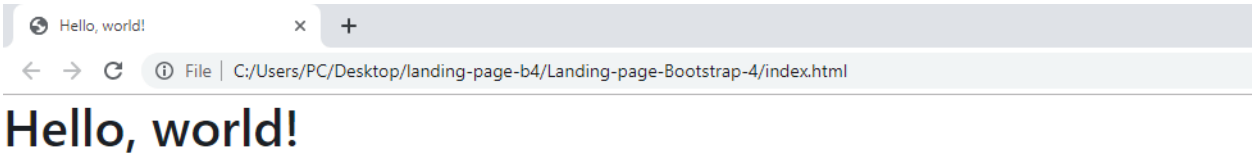
We will copy the code from the theme's navbar.

Earlier in this book, we've already seen how to copy website's code using devtools in Chrome.

We'll go through a similar process to copy our chosen theme's navbar. Next, we'll paste it into the `index.html` file in our local repository, right under the opening `<body>` tag.

Finally, we'll commit the changes with the following commit message: *Add navbar HTML structure*. This commit message implies we haven't added the styles yet.

Here's the screenshot of the layout in browser, after the update.



Screenshot of website after adding the navbar's HTML structure

9.3.1.1 Adding temporary styles to the navbar

To make the header visible we'll add some temporary CSS[102].

First, we'll move the `h1` element down by adding an inline style to it:

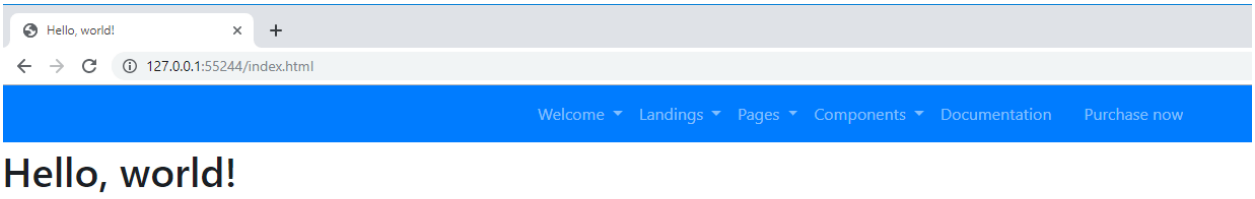
[99] https://themes.getbootstrap.com/product-category/landing-corporate/page/2/
[100] https://themes.getbootstrap.com/product/incline-landing-page-set/
[101] https://themes.getbootstrap.com/preview/?theme_id=3569&show_new=
[102] https://github.com/ImsirovicAjdin/Landing-page-Bootstrap-4/commit/1dfbebda62bac2dc7ddd148b2da0c963a9d74484

```
<h1 style="position: absolute; top: 60px">Hello World!</h1>
```

Next we'll add the contextual color classes on the `nav` element:

```
<nav class="bg-primary text-light navbar navbar-expand-xl navbar-dark  navbar-toggla\
ble  fixed-top">
```

This minor update makes the navbar visible:

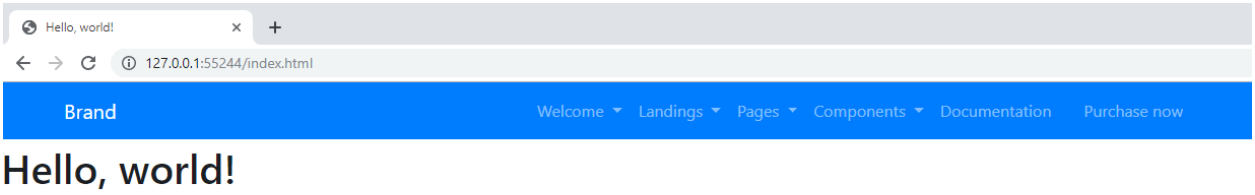


Screenshot of website after adding some temporary styles to the navbar's HTML structure

9.3.1.2 Replace the SVG navbar-brand image

Next, we'll erase the SVG inside the `a` tag with the class of `navbar-brand`, and replace it with text: `Brand`.

The commit for this update is titled: Replace navbar-brand image[103].



Screenshot of website after replacing navbar-brand

Next, we'll add the jumbotron area.

9.3.1.3 Adding the background image

In this section, we'll be adding the large photo in the background.

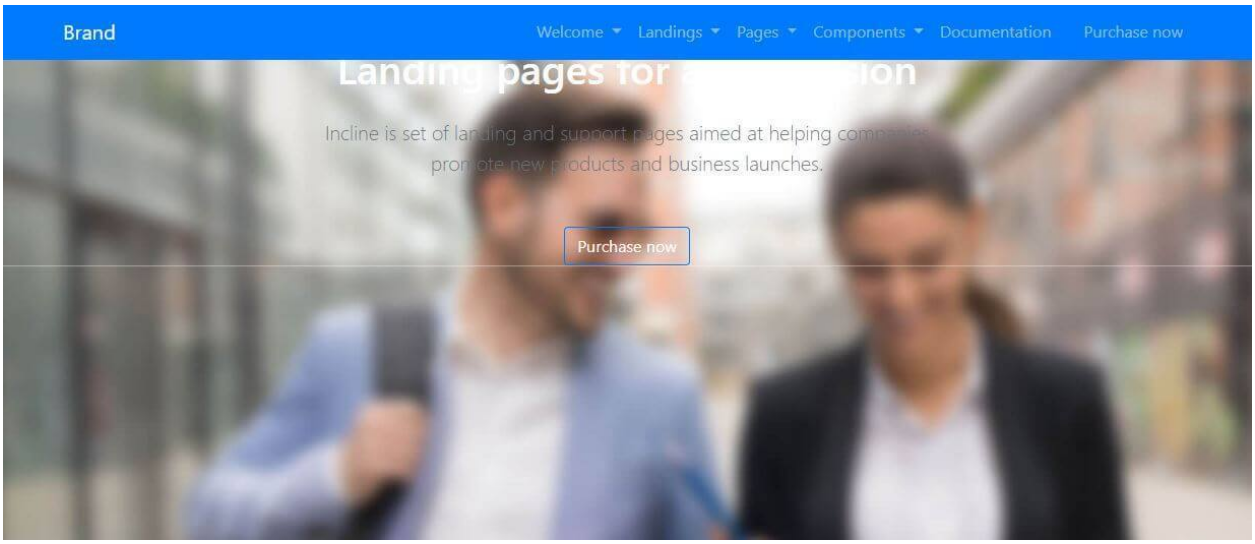
[103] https://github.com/ImsirovicAjdin/Landing-page-Bootstrap-4/commit/215faa6a75028edc89d15dfa5d76d339ac2a957a

Next, again, we'll use devtools to find the HTML element to copy[104].

Besides the HTML for the background image, this time I've also used the styles panel to find the CSS used to get the effect of the image covering the whole screen, and I've copied the CSS styles too. They are inline on the element for now. Once we've updated the layout completely, we can move these inline styles to their proper CSS classes. Dealing with this now would just slow things down and make this article longer.

We'll name this commit: Add the background image[105].

Right now, our layout looks like this:



Website screenshot after adding the jumbotron image

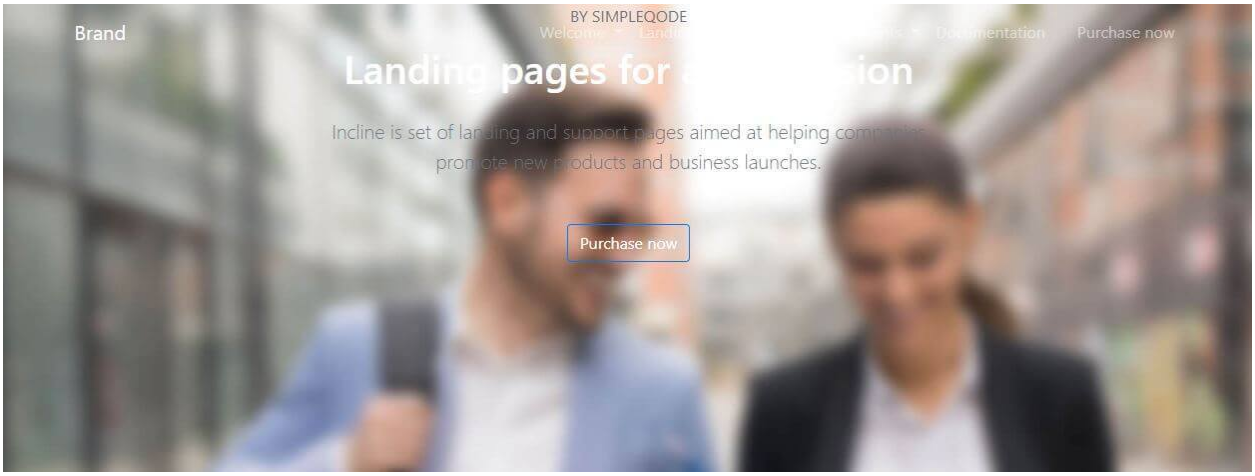
9.3.1.4 Removing navbar's contextual colors

Now we can get rid of the temporary contextual colors on the navbar, and remove the Hello World `h1` element.

This commit is titled: .

The updated layout is still a work in progress:

[104] https://www.codingexercises.com/guides/building-bootstrap-layouts-part-7#adding-the-navbar
[105] https://github.com/ImsirovicAjdin/Landing-page-Bootstrap-4/commit/da1db7dcbfc171f1fb9d471918c9fb42bd27a99a



Website screenshot after adding the jumbotron image

To fix it, we're going to really have to dig into the dev tools and find the styles applied.

9.3.2 Improving the styles

This part of the layout-building process is a bit like detective work.

We're looking at the source theme and we're piecing together the CSS to apply and the CSS to ignore.

We're inspecting the *Elements* panel and deciding which HTML is relevant and which HTML can be skipped.

We're trying to match the HTML to the CSS that achieves the effect that we want.

If this process feels hard or tedious, try to gamify it for yourself.

You could:

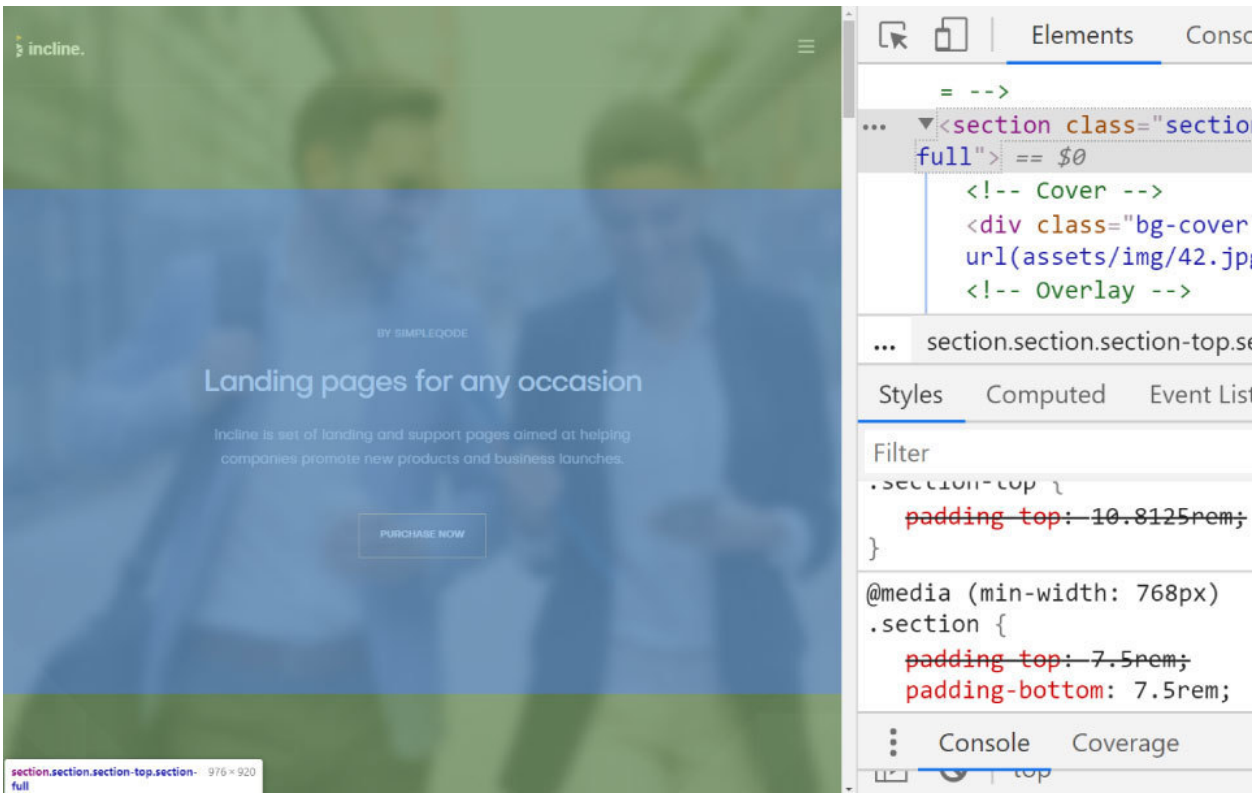
1. Race against the clock - see how many elements you can copy from the source theme in 10 minutes
2. Take a notebook and jot down the techniques you haven't seen before; for example, ask yourself, "why did the developer use this approach when adding the background image?"

Regardless of how you try to make it easy, keep in mind that learning a new skill is hard, and that sometimes you'll struggle. Push through it, learn from it, and soon you'll have your own little repository of CSS tips and tricks. This will make your own website layout development much easier.

Anyway, let's get back to the subject. We'll improve the styles by finding out how the main text gets placed in the vertical center of the screen.

9.3.3 Vertically centering the main text

Here's a screenshot of the image and the devtools. We're hovering the mouse over the wrapping `section` element and the developer tools highlights its padding and shows the styles affecting it:



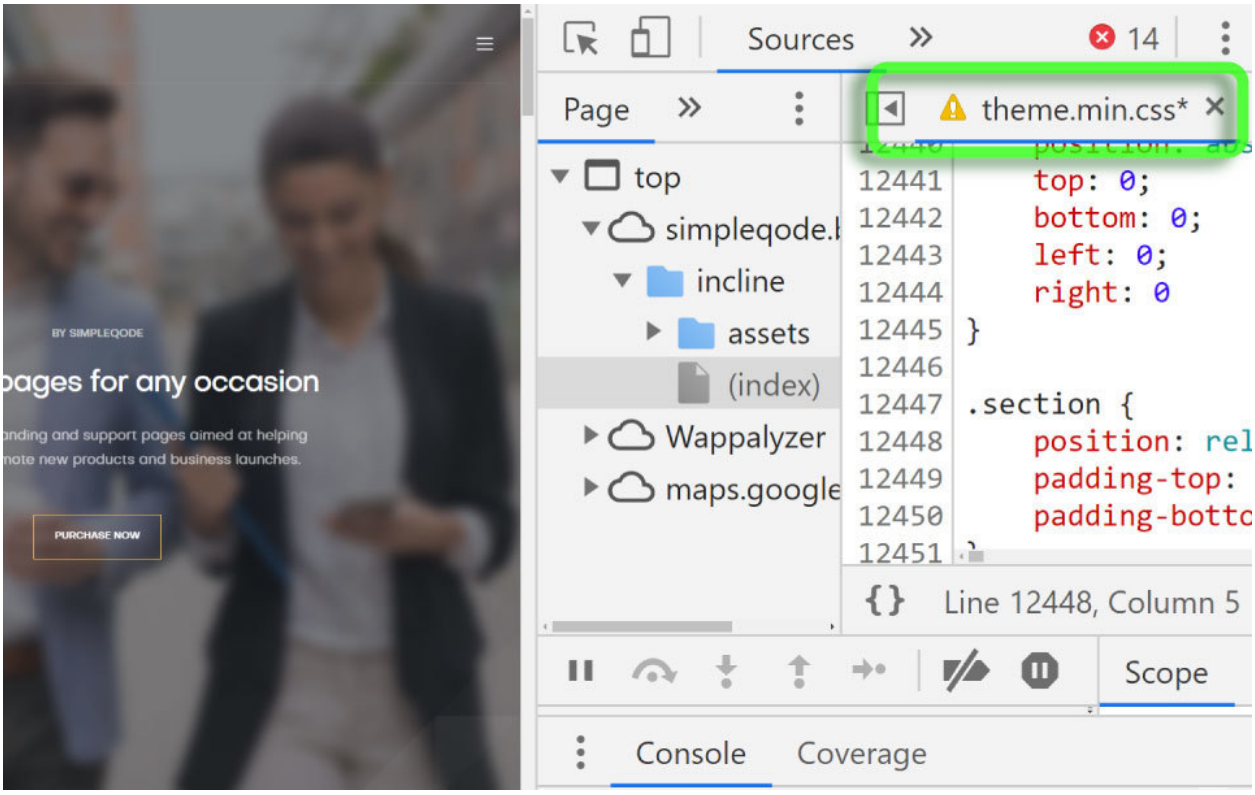
Vertically centering the text

In the bottom-right corner of the above image, we can see that the text gets centered by adding the `padding-bottom` CSS property with the value of `7.5rem`, in a media query that is set to work at `min-width` of `768px`.

The entire CSS code that affects the padding of this `section` element looks like this:

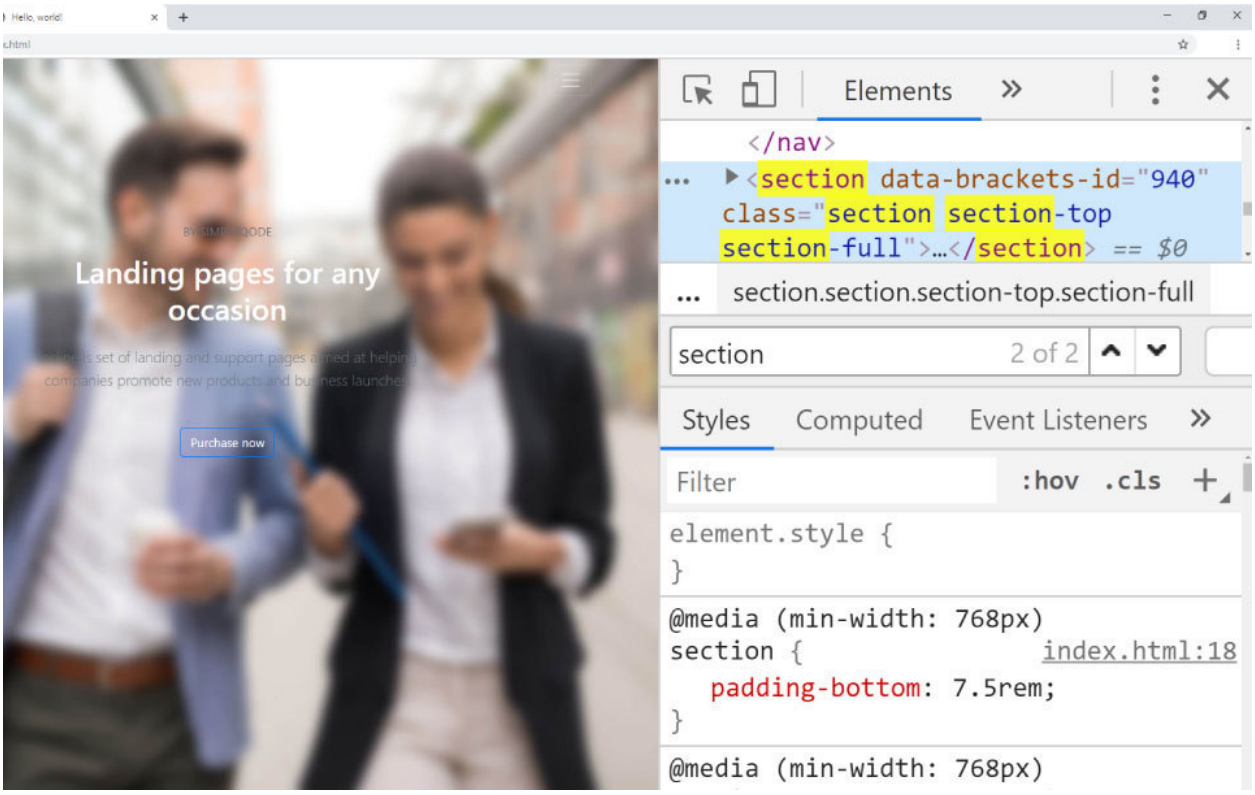
```
@media(min-width: 768px) {
    section {
        padding-top: 13.3125rem;
    }
    section {
        padding-bottom: 7.5rem;
    }
}
```

An alternative approach to the one above would be to simply copy the entire `theme.min.css`, then slowly remove the CSS that isn't used.



A screen showing the minified theme CSS

For now, let's just add the media query to our layout's `style` tag, and see the result in the browser.



Main text centered with padding

9.4 An image is worth a thousand words

Let's now replace the image we copied from the source theme.

First, let's find a nice image on Unsplash.com[106].

Next, let's right click this image and open it in a new window, then copy the URL from the address bar.

Now, back in our own theme's browser window, let's open the developer tools and with the developer tools' Elements panel in focus, press the `CTRL f` keyboard combination.

This will open up the search utility: we can search for any string (any combination of characters). We'll type in "image".

Great, we've got only two occurences of the word "image" on the page.

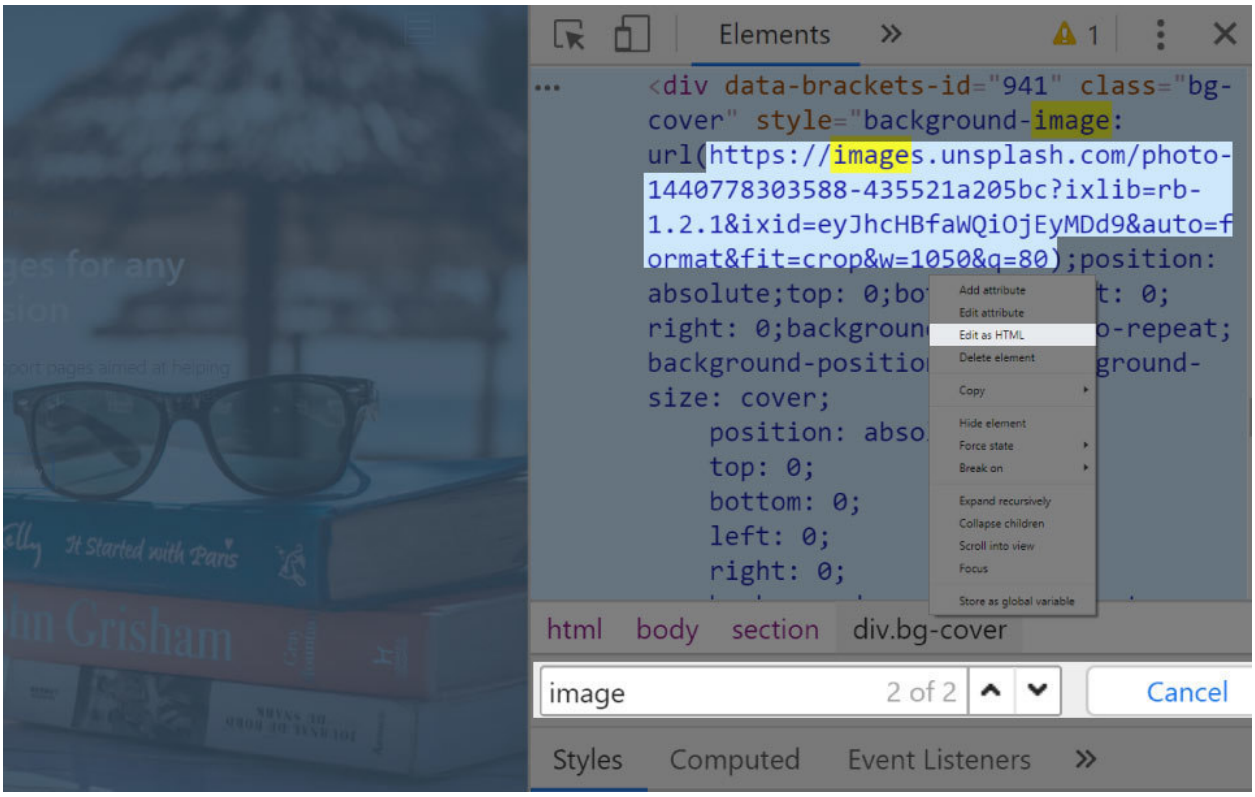
What I was trying to do is to easily find the style that reads: `background-image: url` inside the HTML of the page, because I've copied it over from the source theme earlier.

Now I'm trying to add an image from unsplash live inside the browser, so that I can try different images before I find the one that I like and that I think will work with my theme.

[106] https://unsplash.com/photos/UoqAR2pOxMo

Like they say, an image is worth a thousand words - so I was trying out different images, live in the browser, until I found this vacation-themed image that I liked.

Now all that I need to do is right click onto the part of the Elements panel that shows my searched-for `background-image: url`, and click the "Edit as HTML" command.



Replacing the background image live in devtools

Now that I've replaced the background image live in devtools, I can save it inside my layout's HTML.

The commit is titled: Replace the background image[107].

Next, let's add some additional sections to our webpage.

9.5 Adding additional sections to our page

To speed things up, we'll refer to the examples from the official documentation.

This time, we'll copy a part of the carousel example layout[108] from the Bootstrap 4 official examples.

9.5.1 Adding the main element

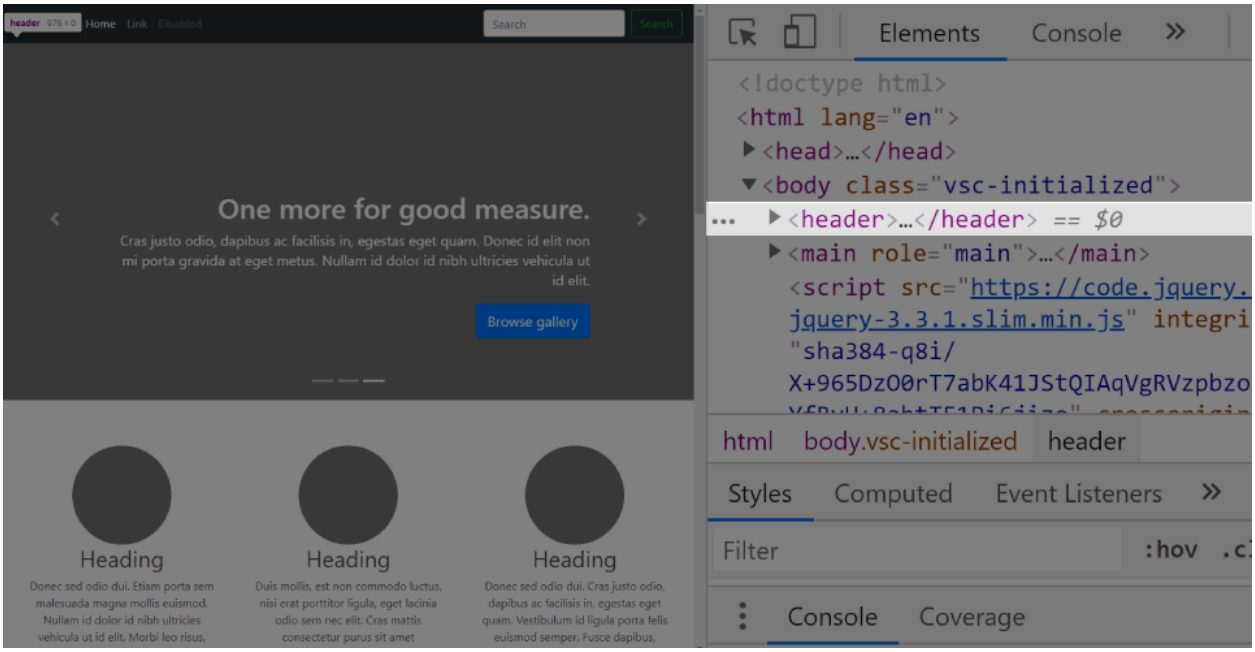
This time, we'll use the developer tools slightly differently.

[107] https://github.com/ImsirovicAjdin/Landing-page-Bootstrap-4/commit/2a7f3baf643235e1422983b90aa77fbb530cfac6
[108] https://getbootstrap.com/docs/4.3/examples/carousel/

Basically, we'll add the entire carousel example layout[109], minus its navbar and carousel.

How do we do this? With devtools, of course.

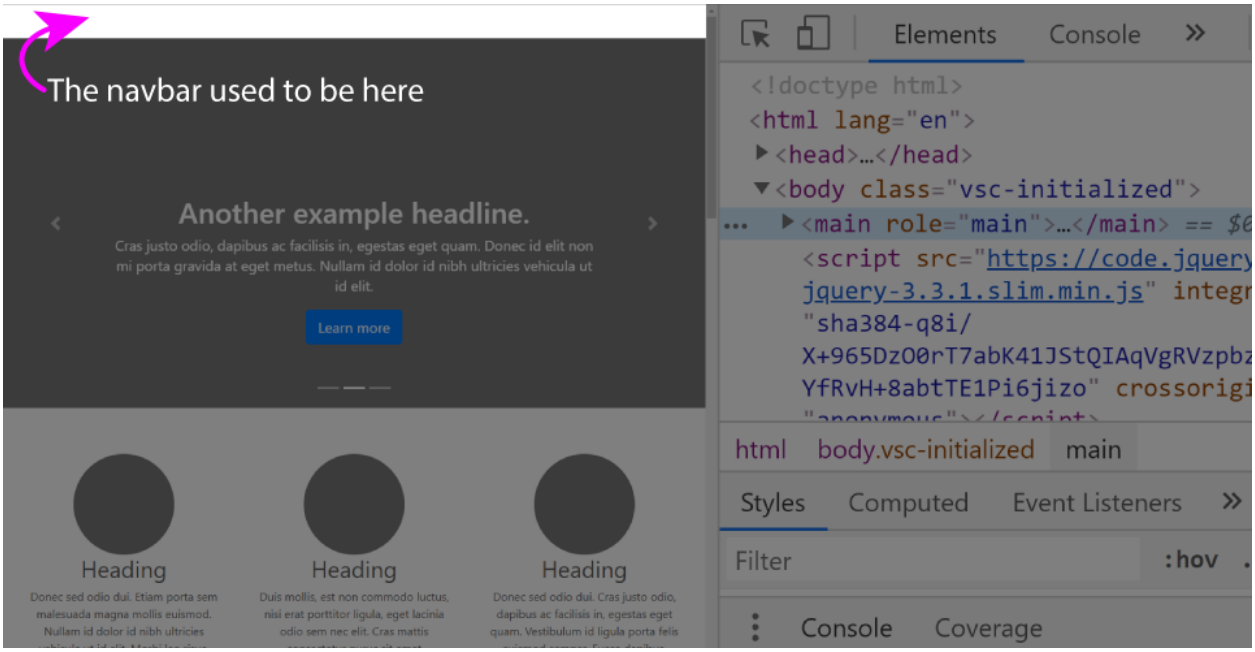
Open the devtools, then click on the *Elements* panel to get the HTML structure in focus. Next, locate the header, and click on the `<header>` element inside the devtools.



Click on the header element inside of devtools

Now simply press the `DEL` key on your keyboard; this will remove the `<header>` element from the website.

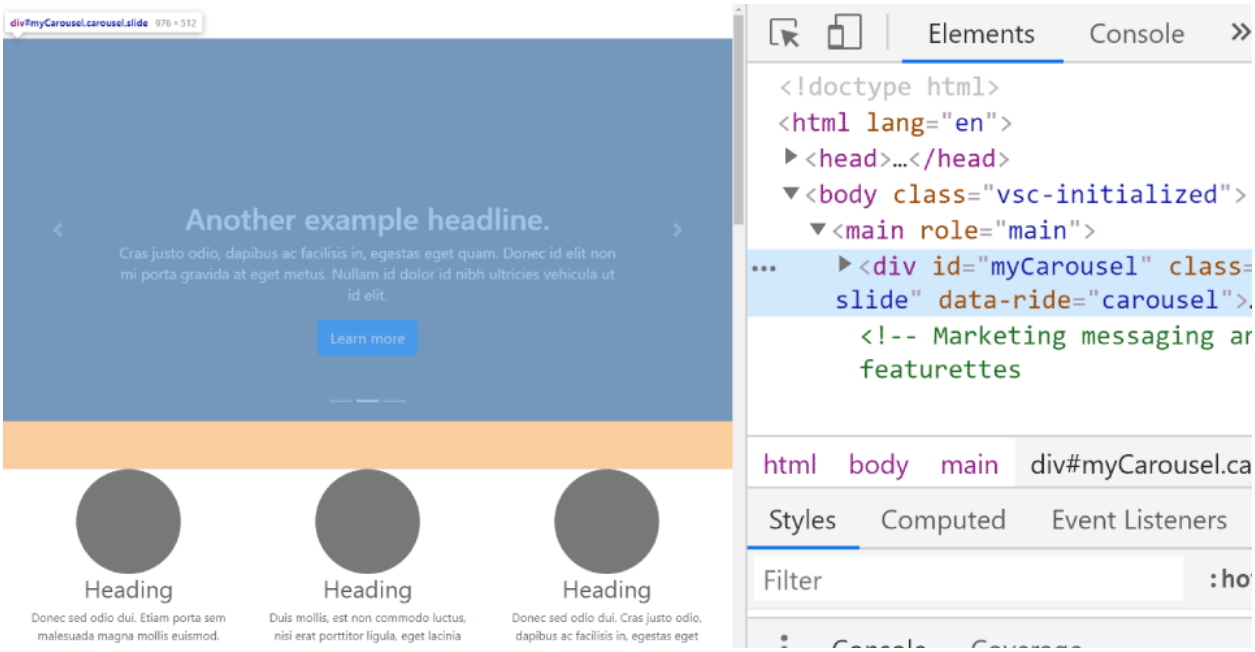
[109] https://getbootstrap.com/docs/4.3/examples/carousel/



The deleted navbar used to be here

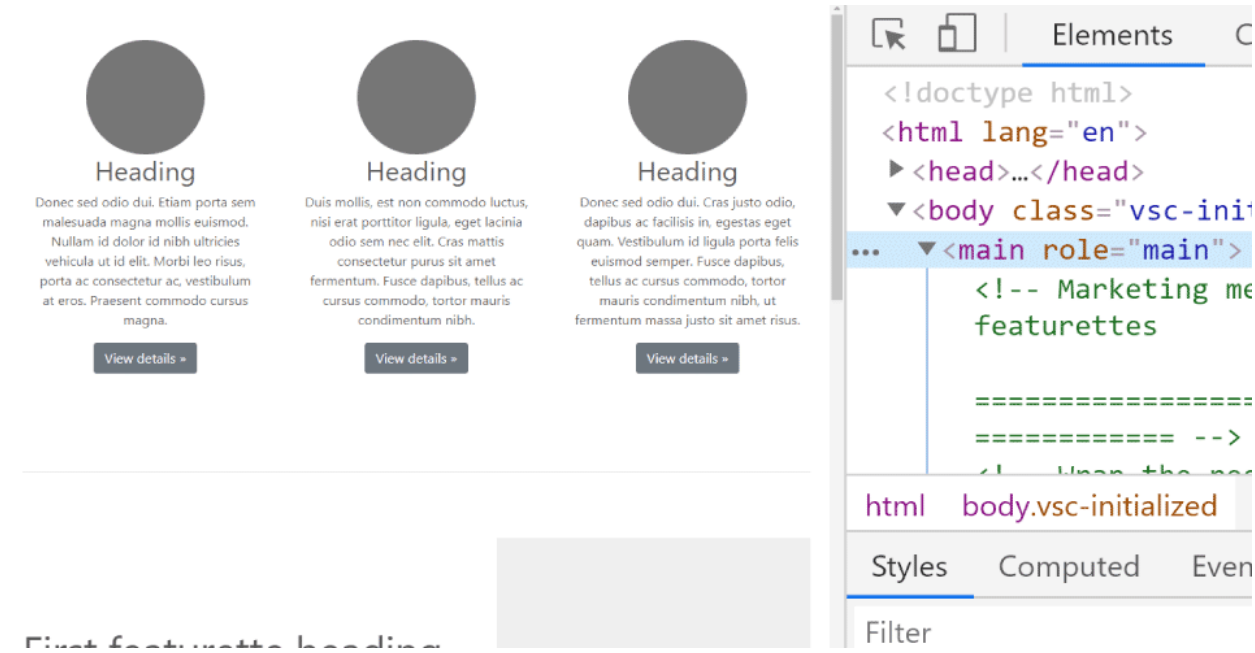
Note that this deletion is a non-destructive change (meaning that this change will not be saved). It doesn't matter if you're deleting an element from a website on the internet or on your own file system, the updates you make to the HTML structure of a website using the devtools **will not be saved on the server**.

Now we'll click the little triangle left of the `<main>` tag, and we'll click on the `div` with the `id` attribute set to `myCarousel`.



The myCarousel div is selected in devtools

Now we'll press the `DEL` button again, which will again update the live website.

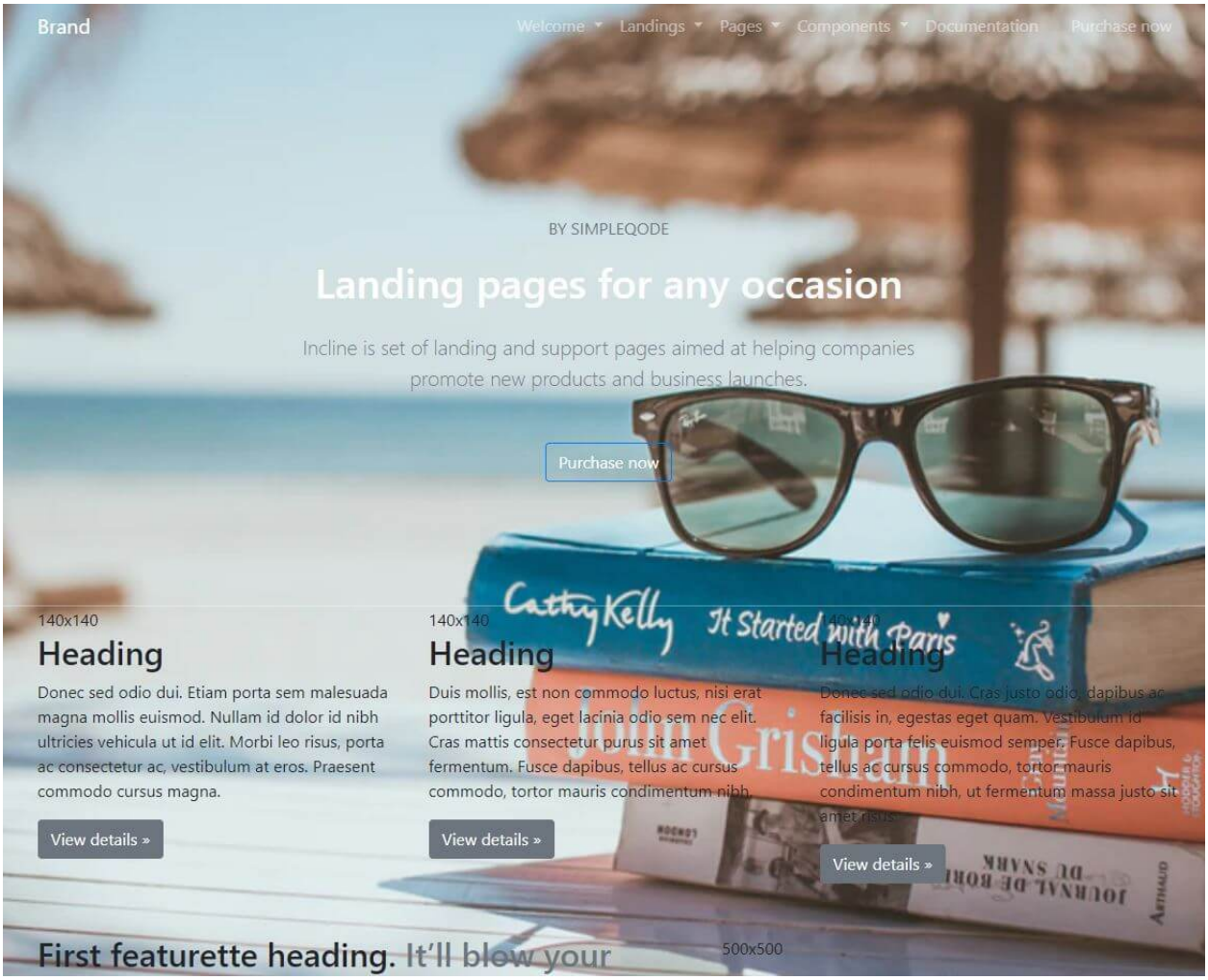


The website with myCarousel div removed

Now we can just copy the `<main>` tag, by right-clicking on it, then clicking "Copy", and then "Copy element".

We'll paste it into our theme, after the section with the background image.

Let's view our own theme now:



Our theme with the main element added

The layout is still far from perfect, but since we've added a large chunk of HTML, let's commit it with this commit message: Add the main tag to layout[110].

Why did we delete elements from the devtools in the first place?

Because I wanted to show you a technique to use to quickly single out an element to copy.

I find this a quick and easy way to work if I need to copy something from a complex layout. It helps me quickly get rid of the noise.

Should you use this approach?

Yes, if it makes sense to you. If it doesn't, at least you know it's possible - you might use it some other time in a different context.

[110] https://github.com/ImsirovicAjdin/Landing-page-Bootstrap-4/commit/6b4bc7457df25c3345c5565605b1b43852ed4edb

9.5.2 Formatting our HTML code in Brackets

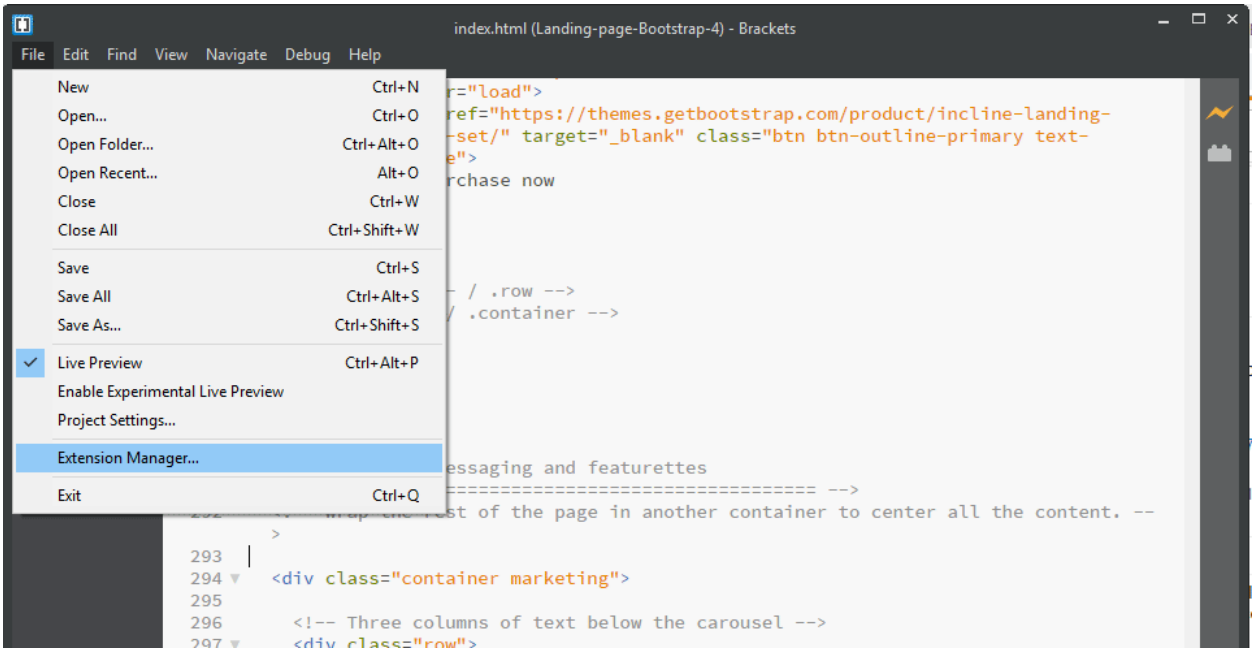
At this point, our theme's structure is finished. There are many things we still need to improve, but they're mostly style-related.

However, there is one thing regarding the HTML structure that is still missing. With all this copying and pasting, our HTML code is not properly formatted: the indentation is off.

Rather than fixing this line by line, we can simply install the *Beautify extension.*

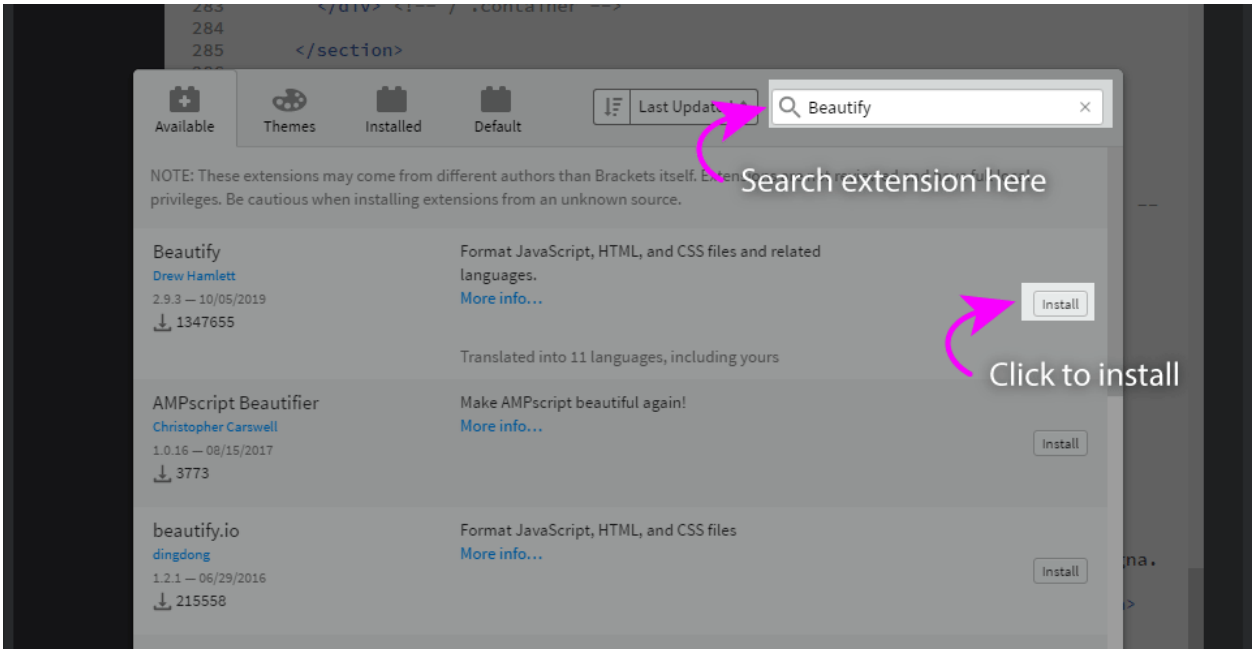
9.5.3 Install the Beautify extension to Brackets

To install an extension in Brackets, we need to click **File > Extension manager**.



Navigating to extension manager in Brackets

Next, we need to type the name of the extension into the search input field, and click the little install button in the right-hand side.



Search for and install a brackets extension

On Windows, to beautify your HTML file, press the `CTRL ALT B` keyboard combination.

Let's commit this change with this commit message: Reformat layout's code[111].

Next, let's improve the styling.

9.5.4 Improving the styling

Let's move all the content inside the `<main>` element under the big image.

This fix is simple: we'll just wrap the `<nav>` and the `<section>` element inside a `<div>` with a class of `container-fluid`, and we'll give it an inline style of height: `100vh`.

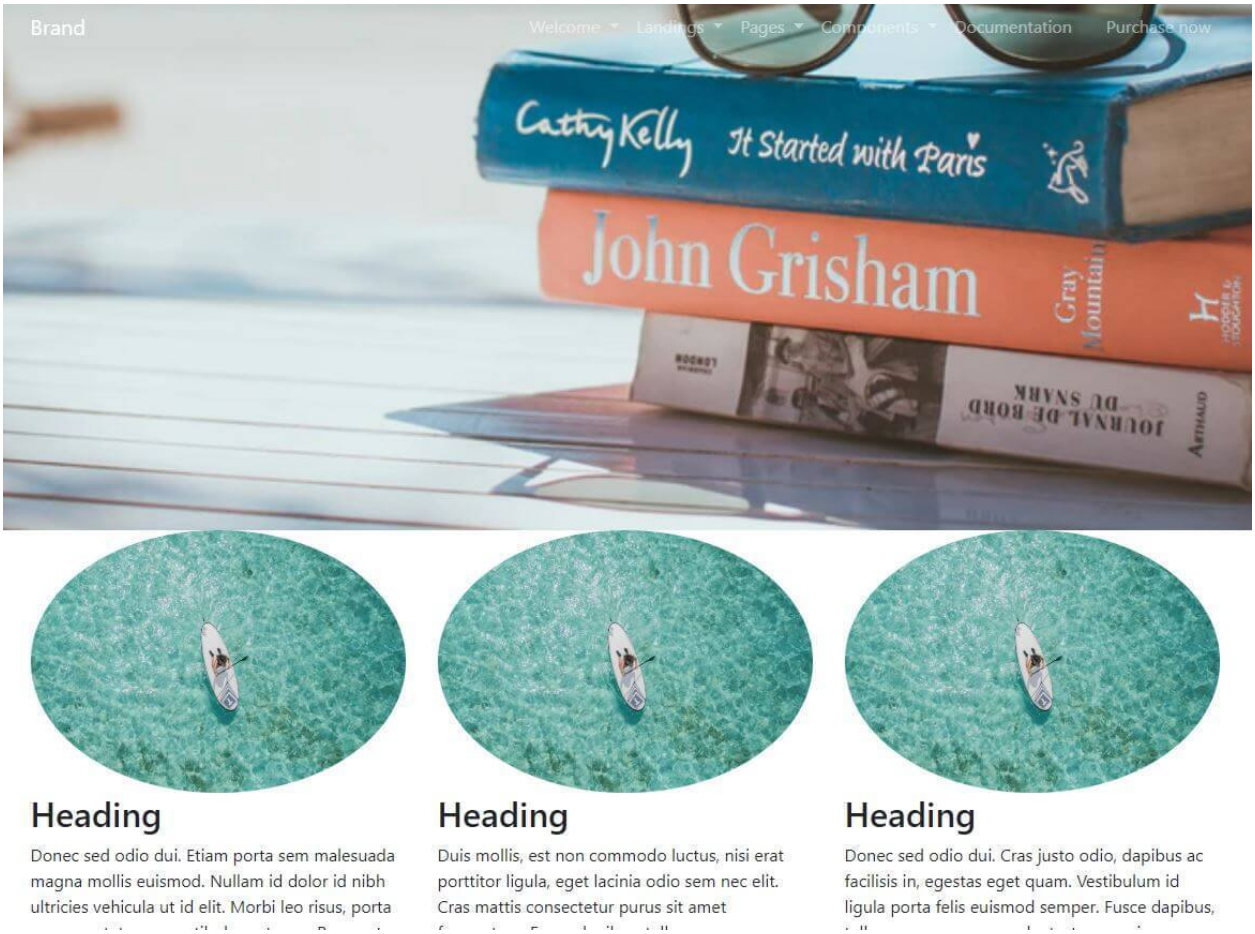
The commit message for this update is: Fix the elements' overlap[112].

9.5.5 Add the missing images

Next, we'll add the missing rounded images[113].

After adding the rounded images, we actually get elliptical images instead of round ones.

[111] https://github.com/ImsirovicAjdin/Landing-page-Bootstrap-4/commit/ced60b8dc258bbd8025376dcec73e152e53804e4
[112] https://github.com/ImsirovicAjdin/Landing-page-Bootstrap-4/commit/00b6acd7691a1ea9a0465e3ab3aa383d1f043971
[113] https://github.com/ImsirovicAjdin/Landing-page-Bootstrap-4/commit/6200e3d7fb3cef9e05288c82a68fae9402d3ebd3



Add rounded images

The `rounded-circle` Bootstrap 4 class will take images that have the same width and height (in a 1:1 ratio), and add CSS property of `border-radius` of `50%`, thus giving them such a large rounded border that the image looks like a circle.

However, this only works on square images. If the width to height ratio is different than 1:1, the circle will become an ellipse, as we can see in the above screenshot.

Luckily, somebody already found a fix for this issue[114].

Of course, we also have another option, and that is, to tweak the layout and make it a bit more our own. Thus, rather than applying the `rounded-circle` class to our images, we'll apply another Bootstrap 4 CSS image manipulation class: `rounded`. This CSS class, when applied to an image, will give it nice rounded corners.

This looks like a good time to learn about some other ways to affect how images look in Bootstrap 4[115].

[114] https://jonathannicol.com/blog/2014/06/16/centre-crop-thumbnails-with-css/
[115] https://getbootstrap.com/docs/4.3/content/images/

The message for the commit that does just that is: Add rounded corners to images[116]. Note that in this commit we've also added a title above the images, and some padding to separate it from the large image one section above. Note also that we are using the `<h2>` tag in the title above the images, but we're applying the Bootstrap CSS class of `h1` to it, like this:

```
1  <h2 class="h1">
```

This is a quick and easy way to make our text look a lot more prominent. You can apply this technique to other HTML elements, not just headings!

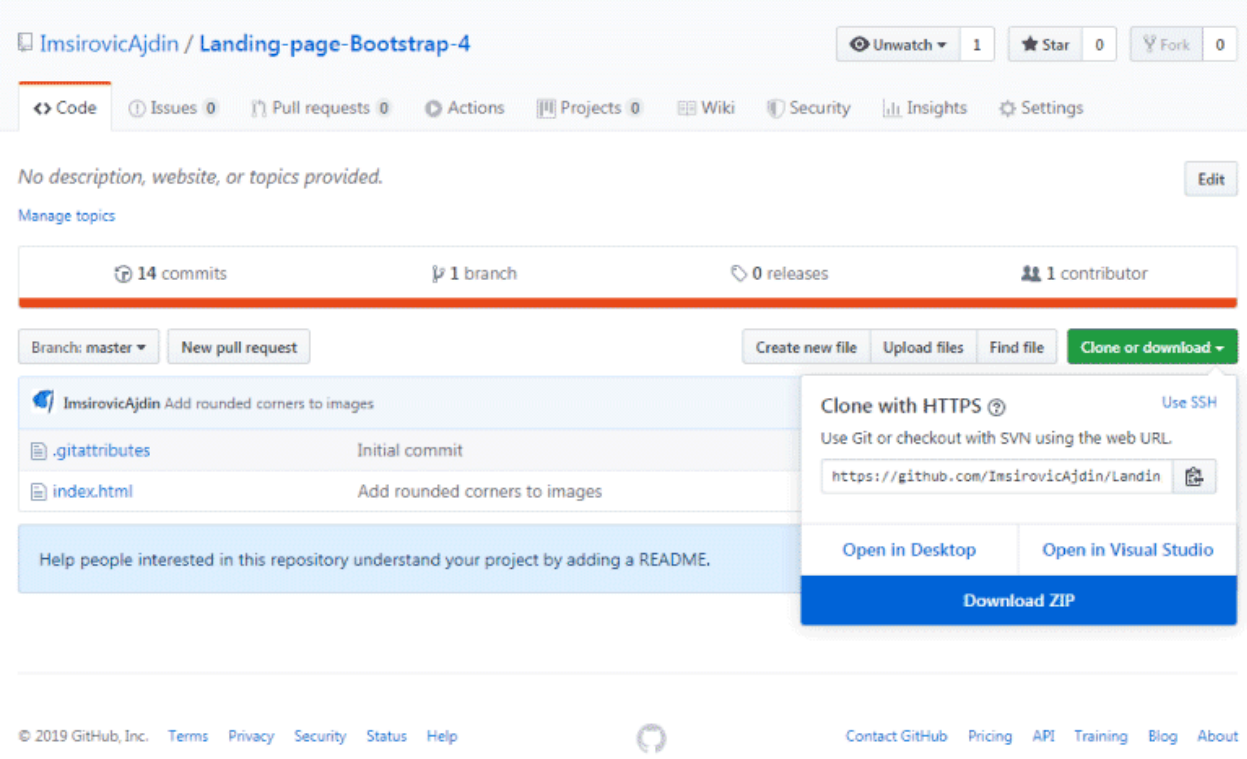
Our layout at this point looks like this:

Add images with round corners

Note that you can download the repository for this layout at any time, by visiting the layout's repository on Github.

Next, press the *Clone or download* button, and then click the *Download zip* at the bottom of the dropdown.

[116] https://github.com/ImsirovicAjdin/Landing-page-Bootstrap-4/commit/be963f1fa6b8dff89e92f7dcb636ea3c51d6fdfa



Download the repository at any time

You could also simply clone the project. Follow this wonderful guide from Github to clone the project using Github Desktop[117].

Next, we'll update the images on the rest of the layout.

9.6 Update the images on the rest of the layout

This is an easy update: We'll just copy one image tag from the images we added earlier, then we'll simply paste it into each of the three divs that have the `col-md-5` class.

This update's commit message is: Update images on layout[118].

Next, we'll update the navbar so that its background updates when we scroll down the page.

For that, we'll be using the scrollspy component from Bootstrap 4[119].

9.7 Making the navbar change background color on scroll

This requires a bit of jQuery. Let's add a `<script>` tag just above the closing `</body>` tag:

[117] https://help.github.com/en/desktop/contributing-to-projects/cloning-a-repository-from-github-desktop
[118] https://github.com/ImsirovicAjdin/Landing-page-Bootstrap-4/commit/be5608adce8cf1f0fb2e8f902cefb98e6fa81fe2
[119] https://getbootstrap.com/docs/4.3/components/scrollspy/

```
<script>

    $(document).scroll(function () {
        $('.navbar').toggleClass('scrolled', $(this).scrollTop() > $('.navbar').heig\
ht());
    })

</script>
```

We'll also need to update our `<style>` tag, just above the closing `</head>` tag:

```
.navbar.scrolled {
    background: hsla(210, 58%, 47%, 0.8) !important;
    transition: 0.25s;
}
```

This update is saved in a commit message titled: Add scrollspy to navbar[120].

With this update, our layout now has 16 commits, and we can now see the full history[121] of how our theme was built.

In the next chapter, we'll build an AirBnB clone layout.

[120] https://github.com/ImsirovicAjdin/Landing-page-Bootstrap-4/commit/6a36413557d5625cb1a375e99c04db345d943663
[121] https://github.com/ImsirovicAjdin/Landing-page-Bootstrap-4/commits/master

Chapter 10: Build an AirBnB clone layout in Bootstrap 4

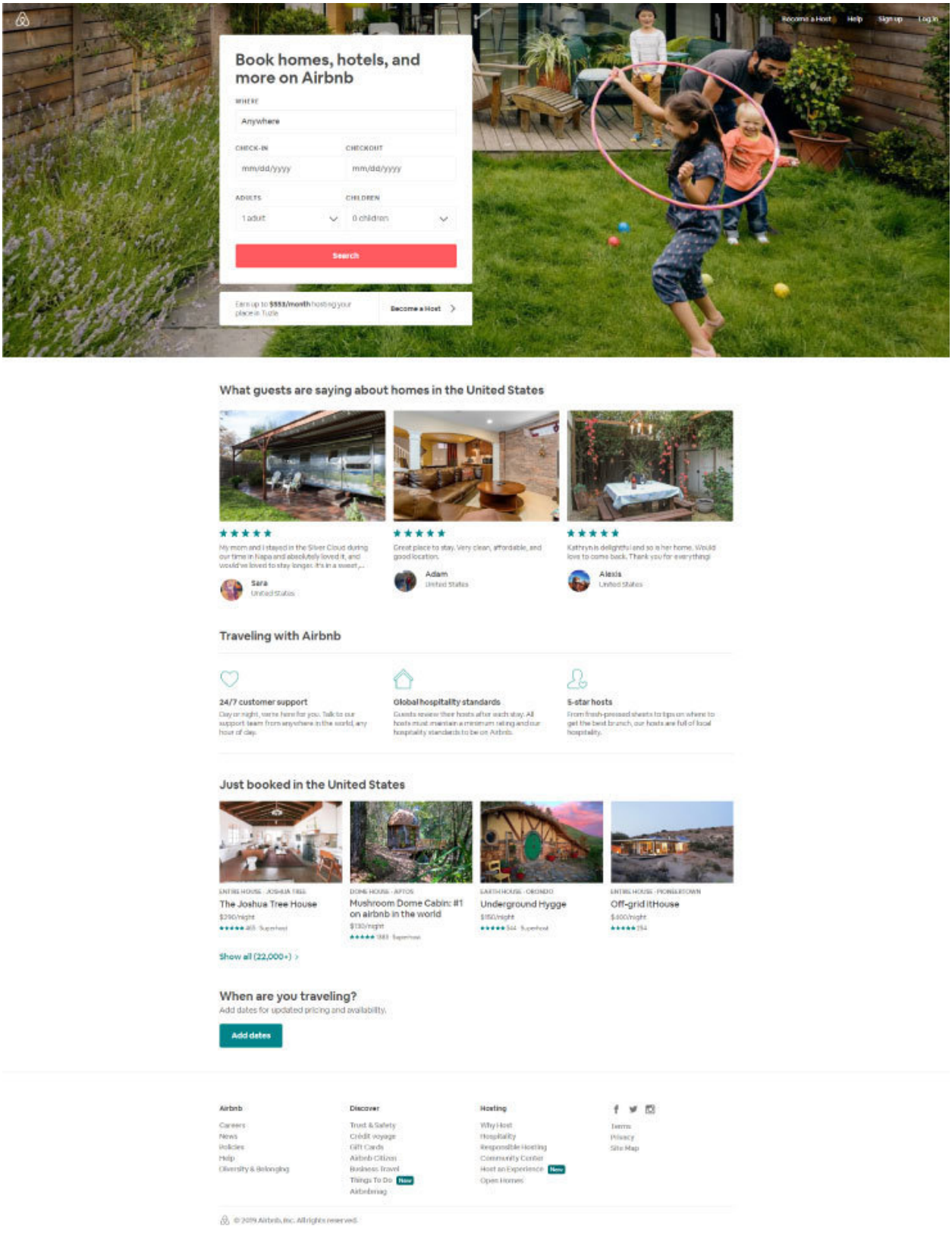
In this chapter, we'll clone an AirBnB layout in Bootstrap 4, and we'll track it with Git.

We'll begin by planning the structure of our layout.

To do that, we'll examine the AirBnB homepage screenshot and decide which Bootstrap components we'll be using.

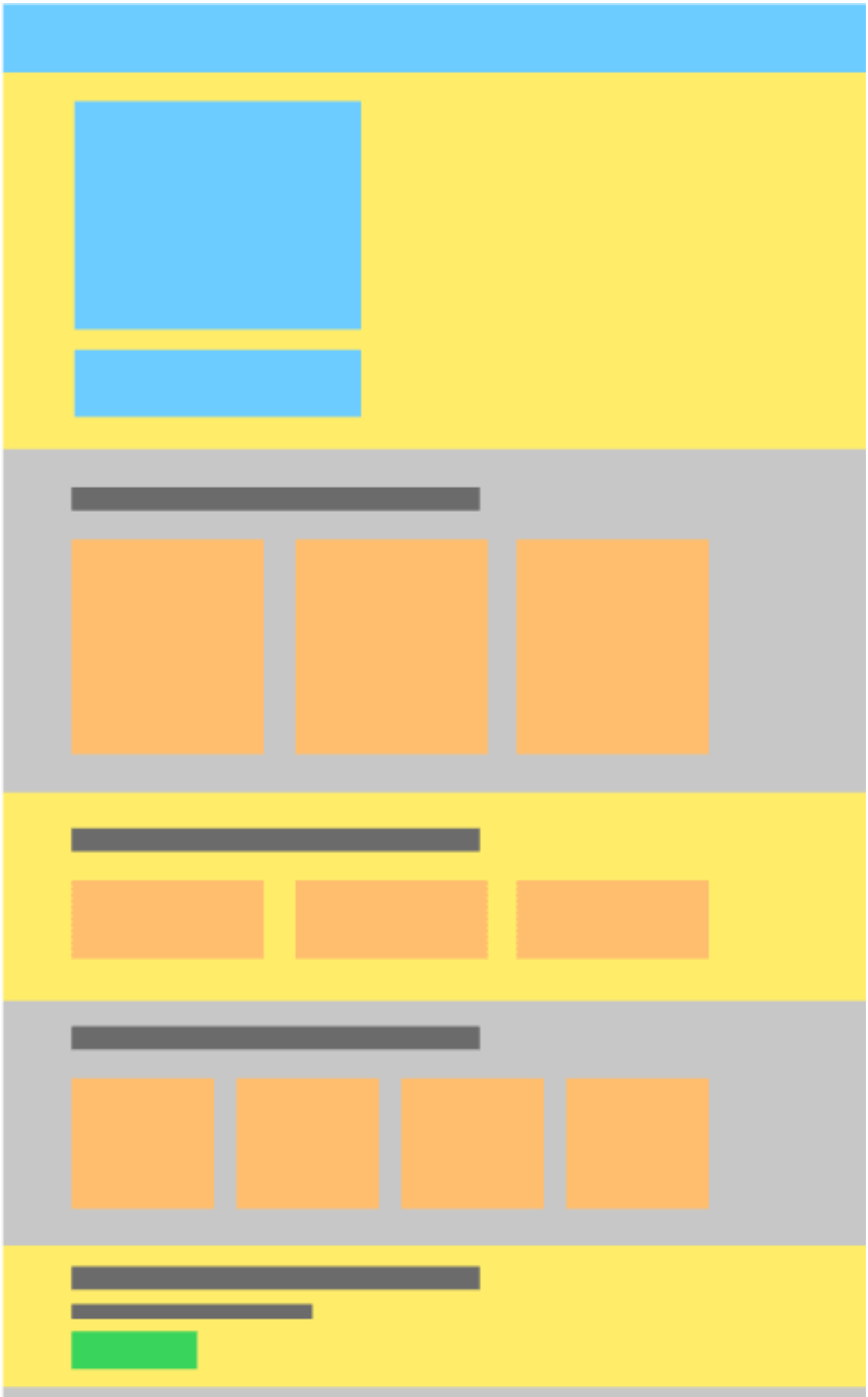
10.1 Planning our layout's structure

Here is the full-height screenshot of the AirBnB homepage at the time this chapter was written:



AirBnB homepage screenshot

Here is the above screenshot, converted into a mockup:



As we can see, this layout actually has 6 `container-fluid` divs, each on top of another.

Thus, let's split the layout's mockup and re-build it, one `container-fluid` div at a time.

Just like in the previous chapter in this series, we'll also be tracking our progress with Git and Github desktop.

Let's begin!

10.2 Add a new repository to Github

We'll start by adding a brand new folder which we'll call `airbnb-clone-layout`.

As we've already explained in the previous chapter, we'll add this new folder to Github Desktop.

This folder will be our new repository.

Finally, as explained in detail in the previous article, we'll publish our repository to Github.

Here's the new Airbnb-clone-layout repo on Github[122].

We've already explained how to start a new project in Brackets, so we'll skip this part of our setup.

10.3 Add the starter Bootstrap template to `index.html`

We'll add an empty `index.html` file into our `Airbnb-clone-layout` folder.

We'll also copy the starter Bootstrap template[123] from the official docs.

Once we've pasted-in the starter template we can commit it with this message: Starter Hello World added[124].

10.4 Adding all the `container-fluid` divs

Next, let's add all the wrapping divs for our project, as follows: inside the `body` element, we'll add the 6 wrapping divs.

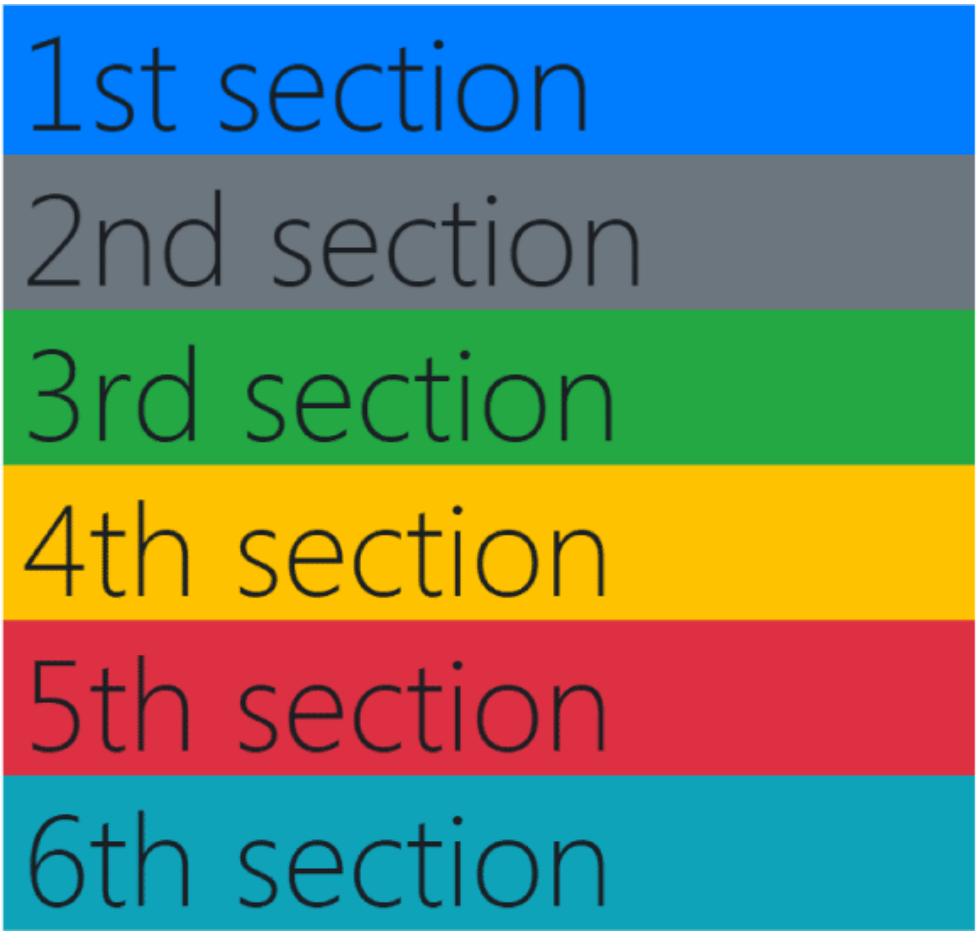
[122] https://github.com/ImsirovicAjdin/Airbnb-clone-layout
[123] https://getbootstrap.com/docs/4.3/getting-started/introduction/#starter-template
[124] https://github.com/ImsirovicAjdin/Airbnb-clone-layout/commit/3e96c59de4db2908db2d777f0db75b9cc3d6f656

```
<div class="container-fluid container bg-primary display-1">
    1st section
</div>
<div class="container-fluid container bg-secondary display-1">
    2nd section
</div>
<div class="container-fluid container bg-success display-1">
    3rd section
</div>
<div class="container-fluid container bg-warning display-1">
    4th section
</div>
<div class="container-fluid container bg-danger display-1">
    5th section
</div>
<div class="container-fluid container bg-info display-1">
    6th section
</div>
```

This update is saved in the commit titled Add container-fluid divs[125].

Here's the screenshot for this update:

[125] https://github.com/ImsirovicAjdin/Airbnb-clone-layout/commit/b2607f2c22ee67e91e7697851f7316f746b4d691



Layout after add container-fluid divs commit

Note that we do *have* `container-fluid` classes on each wrapping div, but we don't have any contextual backgrounds on them. Instead, we've also added inner divs and set both the `container` and the background classes on these divs.

Next, we'll update the first section, the one with the large background image.

10.5 Adding the large background image area

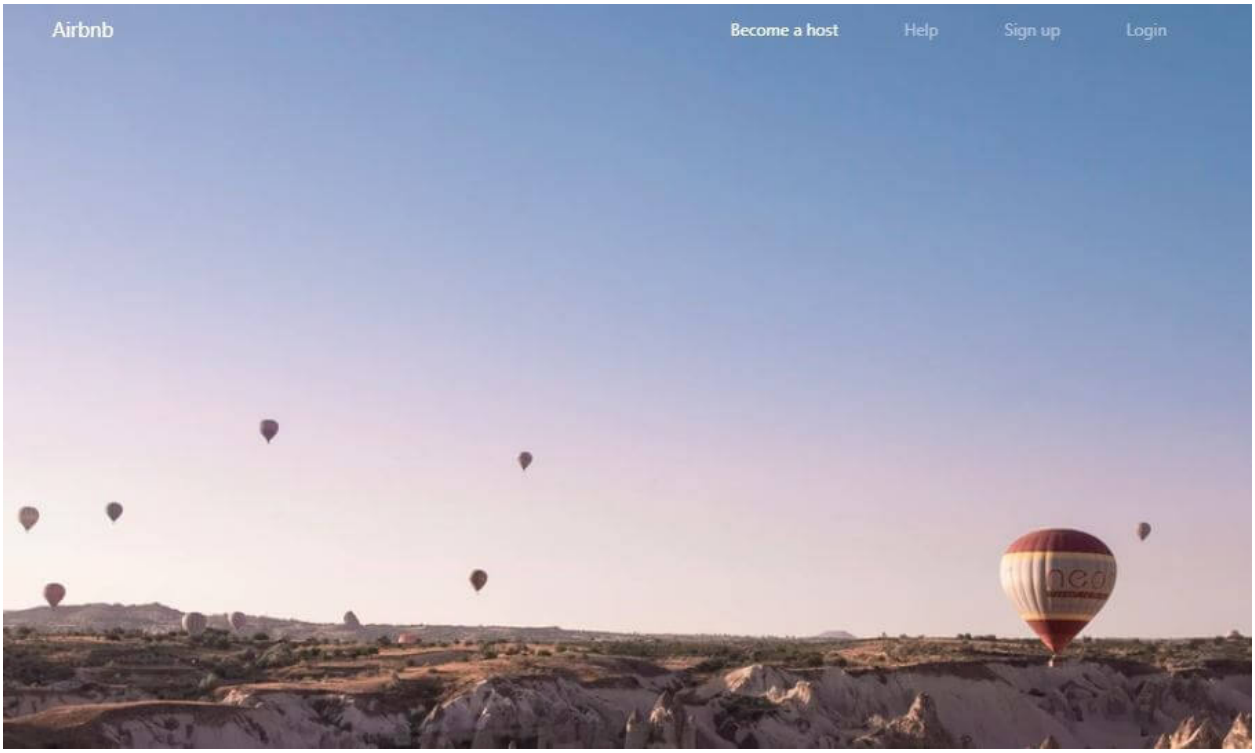
There are several ways we could add the large background image area.

Why not just copy the way it's done on the actual AirBnB page? Even better, this first section looks a lot like what we've already built in the previous layout. So let's just copy it!

This update's commit message is: Add big image section[126].

This is the screenshot of our big image addition:

[126] https://github.com/ImsirovicAjdin/Airbnb-clone-layout/commit/ae71a1b5609dd55f3d5c484f94f2c3b64d1f7ed9



Layout after add big image background commit

Next, inside the big image section, we'll need to add the card with some input fields.

10.6 Adding the card with input fields

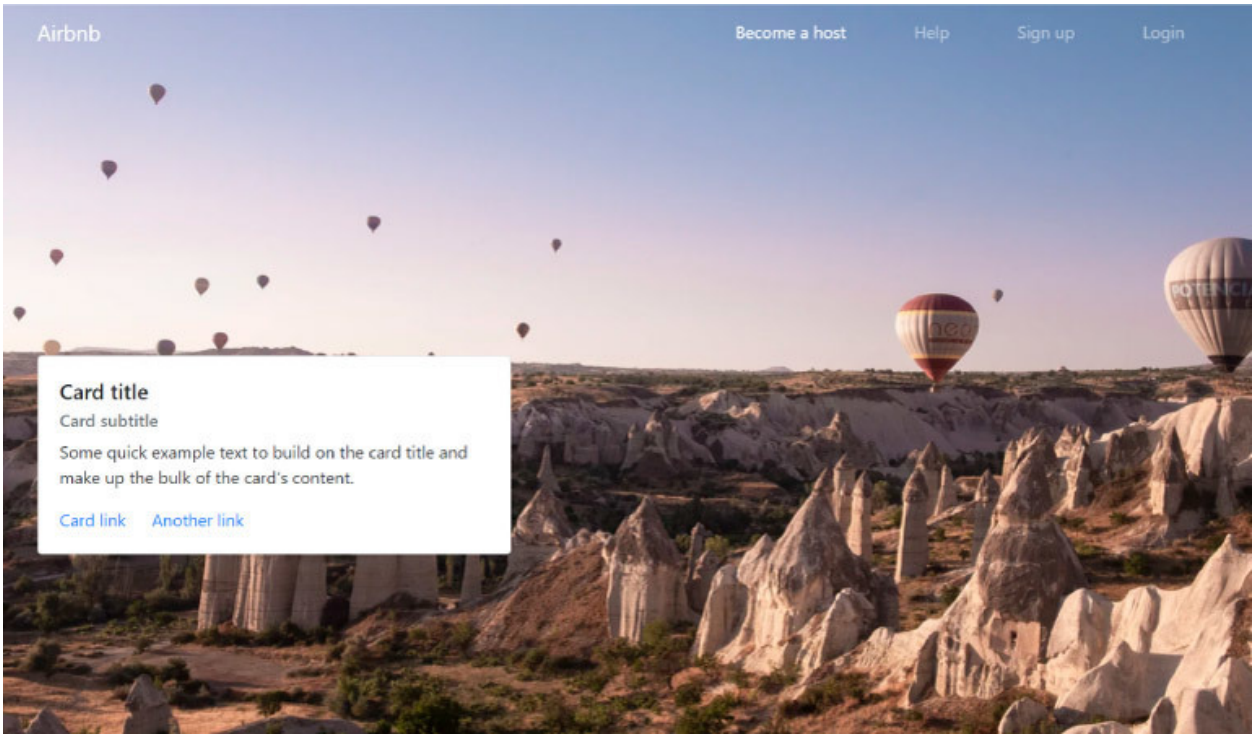
Let's copy a card from the official docs[127].

Next, we'll paste it into our theme, in a `col-sm-4`, inside a `row`, inside a `container`, under the navbar in the first `container-fluid`.

This addition will be saved in this commit message: Add a card to first section[128].

After this update, the layout looks like this:

[127] https://getbootstrap.com/docs/4.3/components/card/#titles-text-and-links
[128] https://github.com/ImsirovicAjdin/Airbnb-clone-layout/commit/6115f193b3c8962d5dde4e7cfb3ae8641fd96cd2



Layout after commit titled "Add a card to first section"

Next, let's add the actual heading and cards like we have in the source AirBnb homepage.

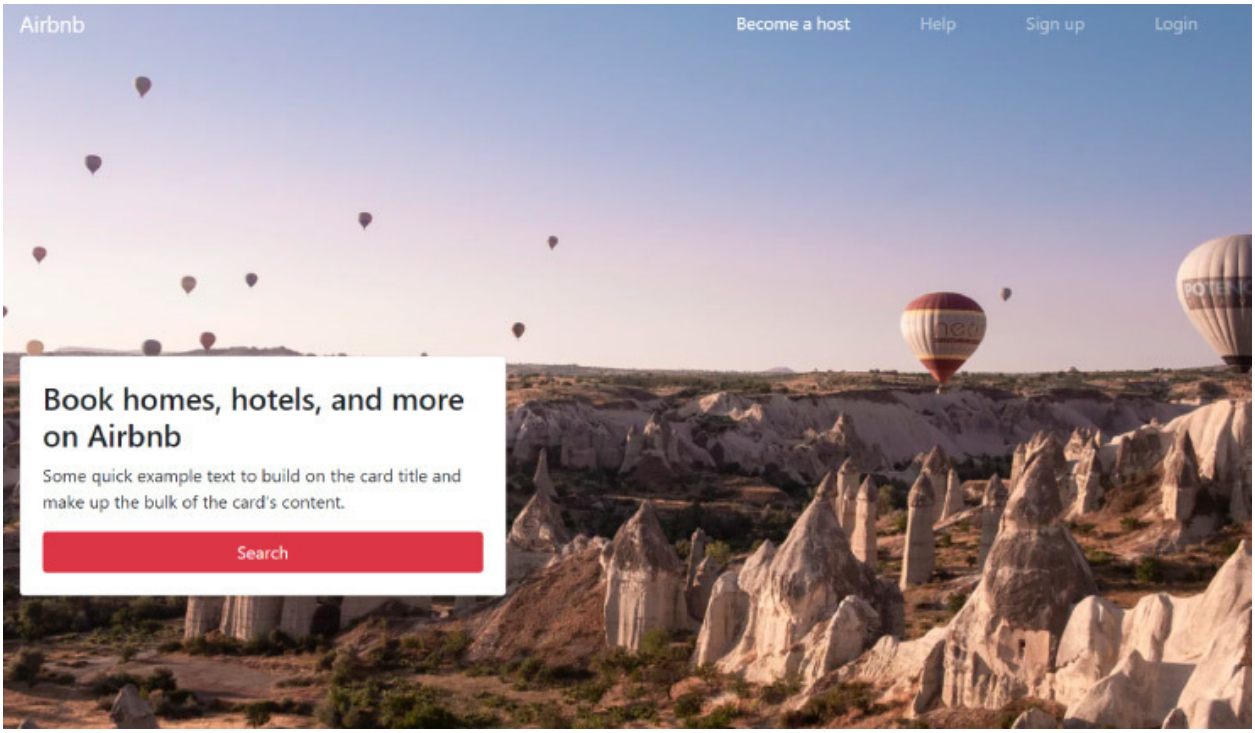
First, we'll add the title: "Book homes, hotels, and more on Airbnb".

In the same commit, we could add the button at the bottom of the card that reads: "Search". For now, we'll give it the class of `bg-danger`.

This commit is titled Add the h1 and the button[129].

Our updated layout now looks as follows:

[129]https://github.com/ImsirovicAjdin/Airbnb-clone-layout/commit/82b98bad108f16af6a7ddff586302be1b135929a



Layout after commit titled "Add the h1 and the button"

Now we can finally add some input fields, five in total:

- A regular text input with "Anywhere" placeholder and a ***WHERE*** label
- Two datepickers with ***CHECK-IN*** and ***CHECKOUT*** labels
- Two dropdowns to choose the number of adults and children, respectively

This shouldn't be too hard to add with the help of the Bootstrap docs on the form inputs[130].

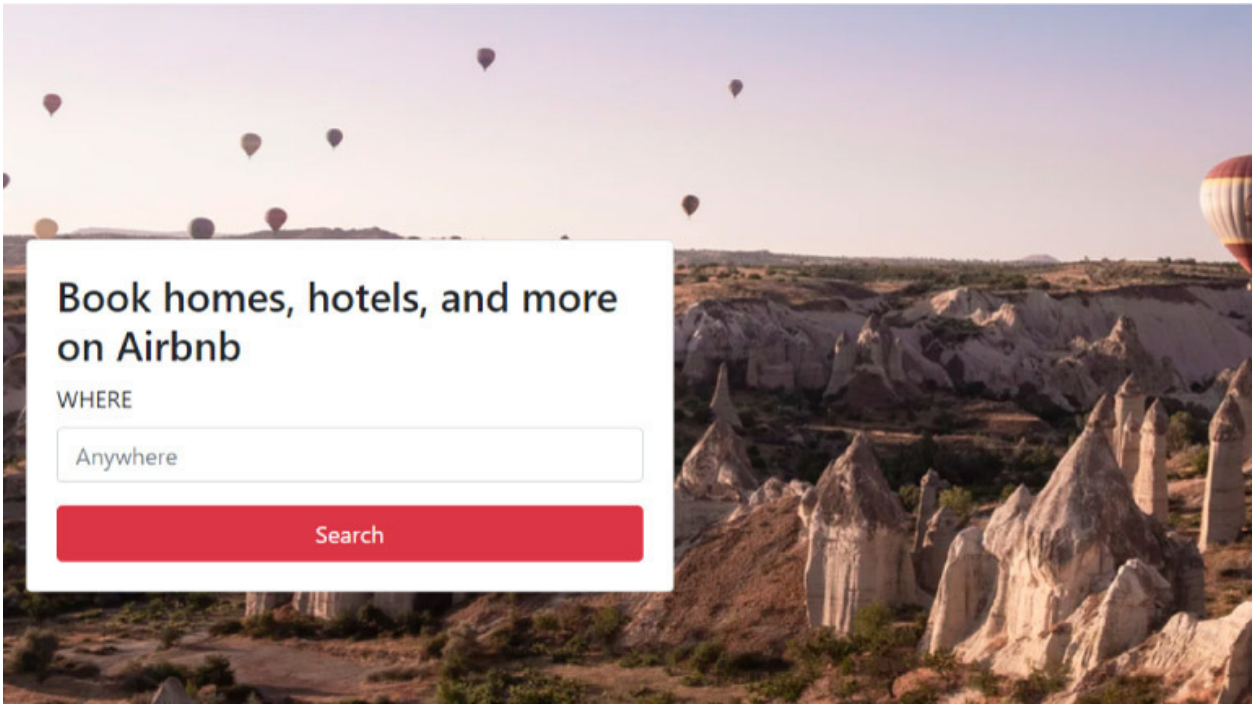
The copied-over code will look like this:

```
<form>
  <div class="form-group">
    <label for="locationInput">WHERE</label>
    <input type="email" class="form-control" id="locationInput" aria-describedby="lo\
cationInputHelp" placeholder="Anywhere">
    <small id="locationInputHelp" class="form-text text-muted sr-only">Please type i\
n your desired destination.</small>
  </div>
</form>
```

The commit at this point is titled: Add the first input[131].

[130] https://getbootstrap.com/docs/4.3/components/forms/
[131] https://github.com/ImsirovicAjdin/Airbnb-clone-layout/commit/ba9e91be2cb2b69926b9c2893b59de84c04ad86a

Here's the screenshot of the layout as it is at this point:



Layout after commit titled "Add the first input"

Next, we're adding two datepickers next to one another, on the same row.

10.7 Adding the two datepickers

There is an interesting website called Gijgo, with a daterange picker example[132]. We'll be using the example code from there and integrate it into our theme.

This is the code we copied from the example:

```
<link
href="https://maxcdn.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css"
rel="stylesheet"
integrity="sha384-wvfXpqpZZVQGK6TAh5PVlGOfQNHSoD2xbE+QkPxCAFlNEevoEH3Sl0sibVcOQVnN"
crossorigin="anonymous">
<script
    src="https://unpkg.com/gijgo@1.9.13/js/gijgo.min.js"
    type="text/javascript">
</script>
<link
```

[132] https://gijgo.com/datepicker/example/daterangepicker

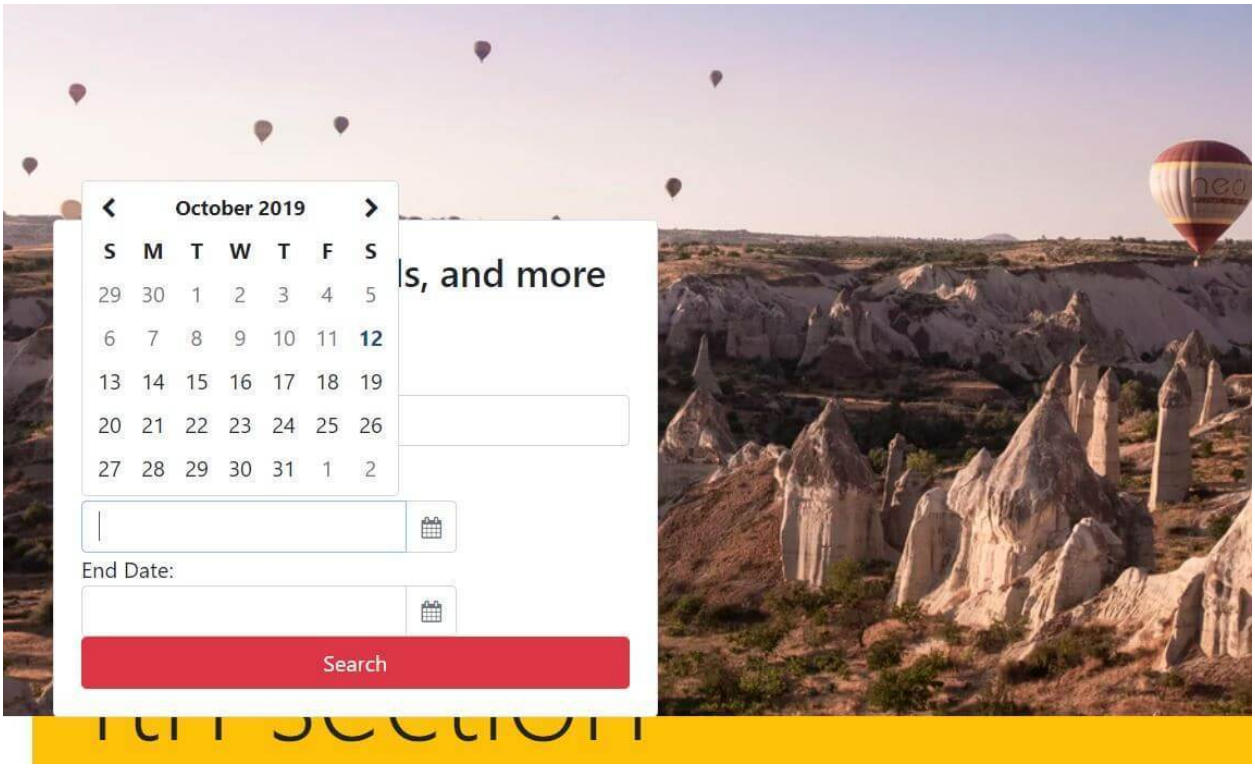
```
    href="https://unpkg.com/gijgo@1.9.13/css/gijgo.min.css"
    rel="stylesheet"
    type="text/css" />
</head>
<body>
    <div class="container">
        Start Date: <input id="startDate" width="276" />
        End Date: <input id="endDate" width="276" />
    </div>
    <script>
        let today = new Date(
            new Date().getFullYear(),
            new Date().getMonth(),
            new Date().getDate()
        );
        $('#startDate').datepicker({
            uiLibrary: 'bootstrap4',
            iconsLibrary: 'fontawesome',
            minDate: today,
            maxDate: function () {
                return $('#endDate').val();
            }
        });
        $('#endDate').datepicker({
            uiLibrary: 'bootstrap4',
            iconsLibrary: 'fontawesome',
            minDate: function () {
                return $('#startDate').val();
            }
        });
    </script>
</body>
```

And here's the link to commit titled Add rangepicker[133].

Note that in order for the datepicker to work, we had to move the jQuery `<script>` up into the `<head>` tag.

After these changes, our card now looks like this:

[133] https://github.com/ImsirovicAjdin/Airbnb-clone-layout/commit/5a507dbc156c7e969ca7e9f1195a7c0411182840



Layout after commit titled "Add rangepicker"

Even though the styling is not as good as it could be, the date picker is fully functional. We'll improve the styles once we've added the entire form structure.

Next, we'll add the fourth and fifth form items: the two dropdowns.

10.8 Adding the dropdowns

Inside the official documentation on Bootstrap, there's a component's toggle in the left sidebar. There you'll find the *Forms* link, and inside the forms link, there's the form controls section[134].

We'll need to copy only a small part of this form controls section, namely, the example `<select>` element, as follows:

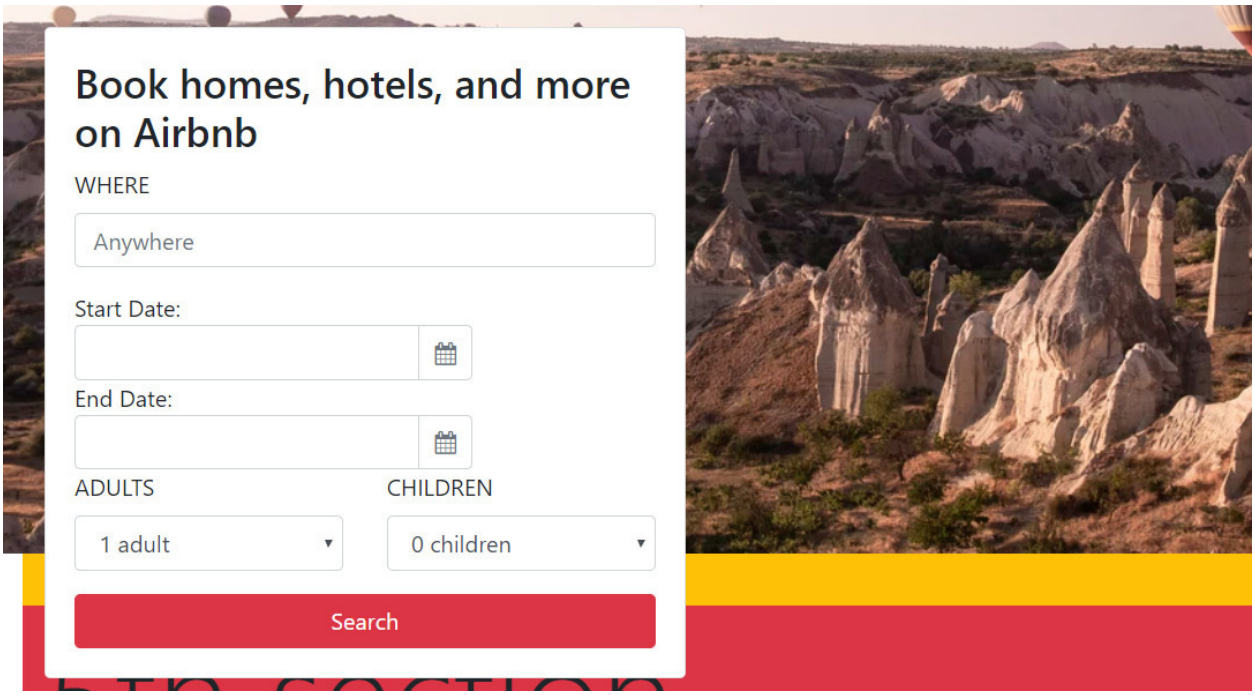
[134] https://getbootstrap.com/docs/4.3/components/forms/#form-controls

```
<div class="form-group">
  <label for="exampleFormControlSelect1">
      Example select
  </label>
  <select class="form-control" id="exampleFormControlSelect1">
    <option>1</option>
    <option>2</option>
    <option>3</option>
    <option>4</option>
    <option>5</option>
  </select>
</div>
```

Of course, we'll update this example select so that it includes a dropdown with a maximum of 16 adult guests and 5 children guests.

Rather than listing out the code right here, let's just have a look at the commit message titled Add the guests dropdowns[135].

After this update, our card in the first section now looks like this:



Layout after commit titled "Add the guests dropdowns"

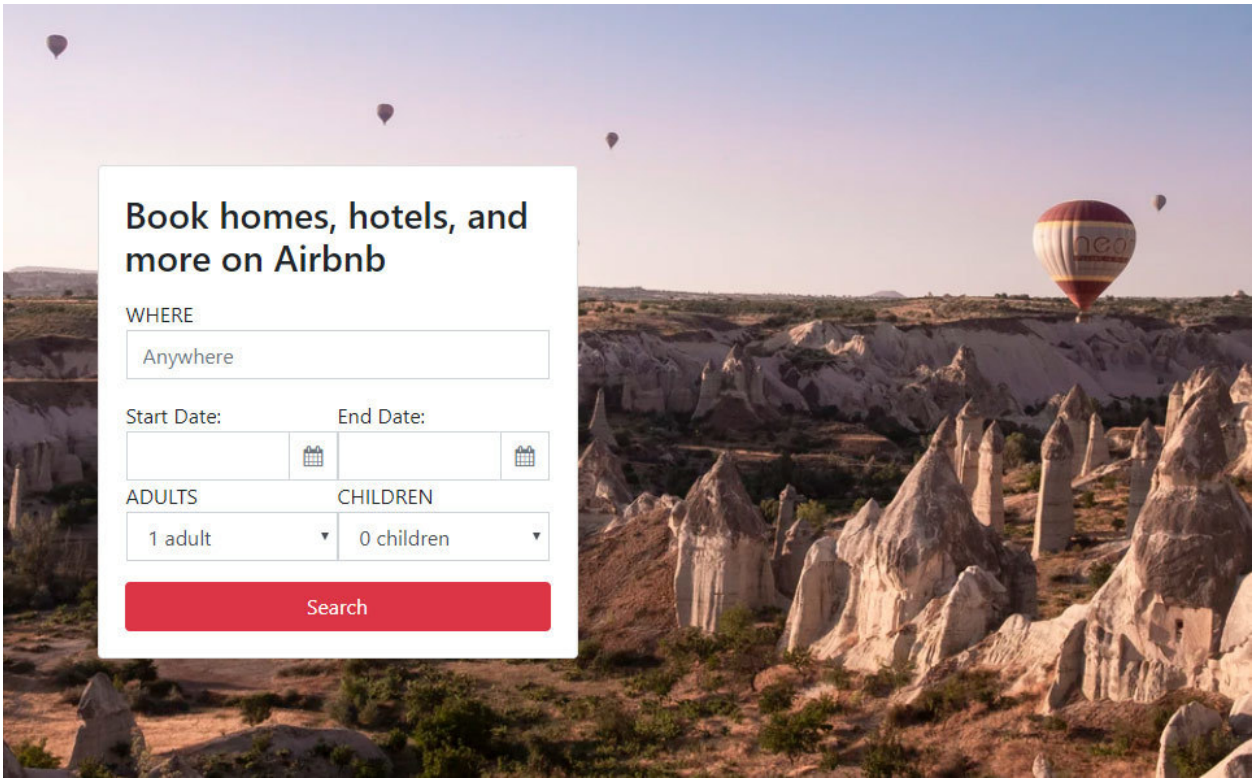
Next, let's fix the styling on the datepicker similar to how we did it with the dropdowns. We'll just add a wrapping `row` and two `col-sm-6` classes for each datepicker.

[135] https://github.com/ImsirovicAjdin/Airbnb-clone-layout/commit/95e68482fc5aec4e977fa71f6386178d6d932846

We'll also add some additional spacing utility classes to make the labels group visually with their inputs.

Finally, we'll add a custom CSS class of `br0` - which stands for `border-radius: 0`.

With all these styling updates, we'll call our commit Complete the Book now card[136], and the screenshot of our completed "Book now" card looks like this:



Layout after the commit titled "Complete the Book now card"

Now we need to add the additional smaller card - the *Become a host* card - under the *Book now* card.

10.9 Add the Become a host card

Let's copy another card from getbootstrap.com[137].

The source card has an image to the left, but we've replaced it with text, and we've also flipped the colums.

In the source card, the columns are ordered so that the first one has the `col-md-4` class, and the second one has the `col-md-8` class, but in our code, we'll flip their order.

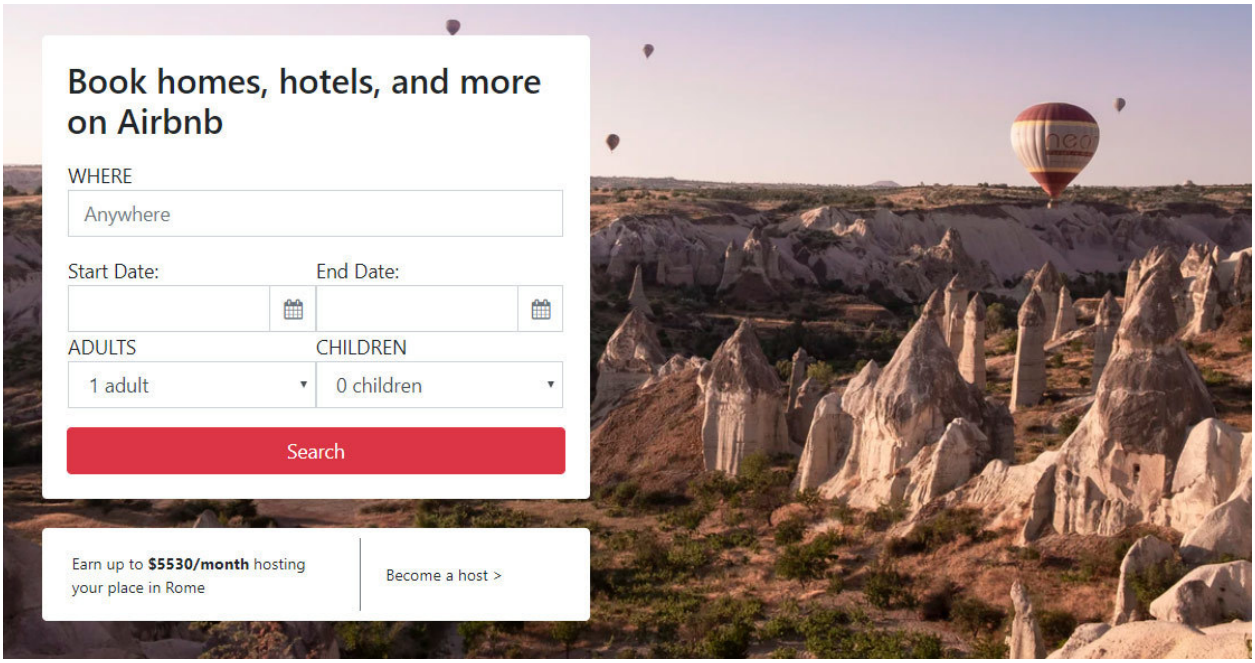
Here's our customized code:

[136] https://github.com/ImsirovicAjdin/Airbnb-clone-layout/commit/17d25a85ae340d9b5afd14633a3e2b999da89f2f
[137] https://getbootstrap.com/docs/4.3/components/card/#horizontal

```
<div class="card my-4 p-2 small border-0">
    <div class="row no-gutters">
        <div class="col-md-7 d-flex align-items-center">
            <p class="my-0 px-3">
                Earn up to
                <strong>$5530/month</strong>
                hosting your place in Rome
            </p>
        </div>
        <div class="col-md-5 border-left border-secondary">
            <div class="card-body">
                <p class="card-text">
                    Become a host <span>&gt;</span>
                </p>
            </div>
        </div>
    </div>
</div>
```

We'll give this commit the message of Complete the Become a host card[138].

Here's the screenshot of the result in browser:



Layout after the commit titled "Complete the Become a host card"

[138]https://github.com/ImsirovicAjdin/Airbnb-clone-layout/commit/add575b8594f31eacc3df4a225288886b2a7779f

Notice that I've forgotten to give some spacing between the datepicker and the `<select>` elements in our *Book now* card. This is a quick fix, including the addition of two `mt-3` classes at the right place.

This update can be seen in the commit message Fix spacing in Book now card[139].

Now that we've completed the first section, let's begin working on the second one: Testimonials.

10.10 Working on the testimonials section

We'll begin work on the testimonials section with a brand new commit: Format the code[140].

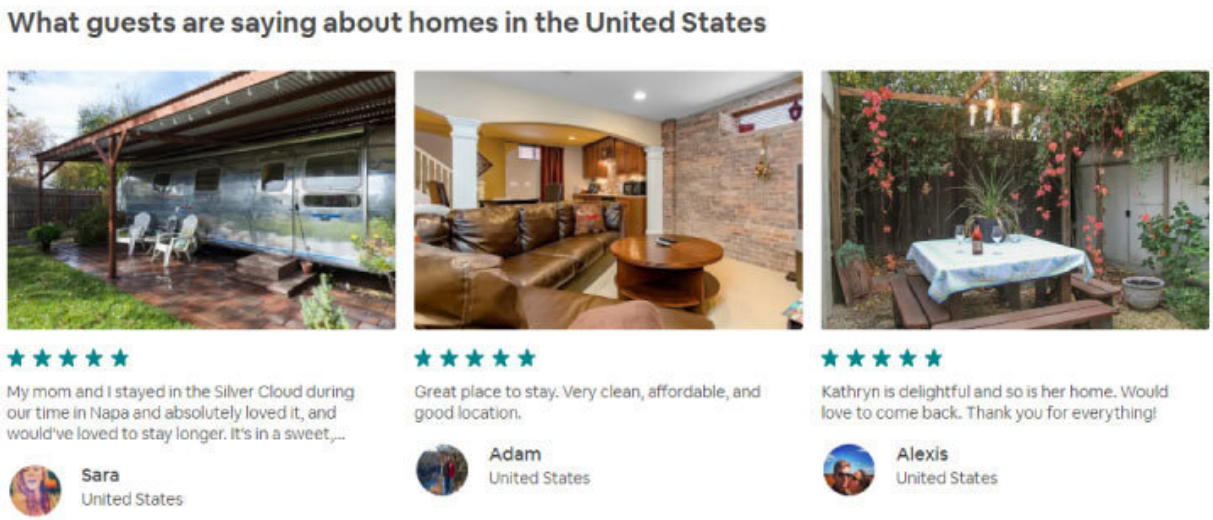
This commit is just some formatting applied to our layout.

Remember: It's recommended to add code formatting in a separate commit.

The reason for this is that formatting updates the entire code of the formatted file, and adding another change together with formatting would make that change very hard to spot when you inspect a specific commit.

Now we can continue with updating the testimonials section.

This is the source AirBnb testimonials section:



The source AirBnB testimonials section

Obviously, we'll use an `<h2>` element with the class of `h3` on it, and we'll follow it up with three cards with images on top[141]. Each of the three cards will have a little media object[142] at the bottom. Also, we need to decide how we'll add the stars: should they change on hover? Or maybe we should

[139] https://github.com/ImsirovicAjdin/Airbnb-clone-layout/commit/a4c0c9c5825be046994070e8ae80be913d52896f
[140] https://github.com/ImsirovicAjdin/Airbnb-clone-layout/commit/2fdac262bf104f09d9acbb40610021f8ded36884
[141] https://getbootstrap.com/docs/4.3/components/card/#images-1
[142] https://getbootstrap.com/docs/4.3/components/media-object/

take the easy route and just display static star icons. Thus, if there is a four star rating, we'll just add four star icons, one after another.

Now, with this tiny bit of planning, we know what it will take to build this.

Here are the copied snippets from the Bootstrap docs.

The cards:

```
<div class="card mb-3">
  <img src="..." class="card-img-top" alt="...">
  <div class="card-body">
    <h5 class="card-title">Card title</h5>
    <p class="card-text">This is a wider card with supporting text below as a natura\
l lead-in to additional content. This content is a little bit longer.</p>
    <p class="card-text"><small class="text-muted">Last updated 3 mins ago</small></\
p>
  </div>
</div>
```

The media objects:

```
<div class="media">
  <div class="media-body">
    <h5 class="mt-0 mb-1">Media object</h5>
    Cras sit amet nibh libero, in gravida nulla. Nulla vel metus scelerisque ante so\
llicitudin. Cras purus odio, vestibulum in vulputate at, tempus viverra turpis. Fusc\
e condimentum nunc ac nisi vulputate fringilla. Donec lacinia congue felis in faucib\
us.
  </div>
  <img src="..." class="ml-3" alt="...">
</div>
```

As for the stars, we'll just use the HTML entity for the star character, which is `&#9733`:

```
<span>&#9733;</span>
```

Now that we've got all the ingredients in place, let's combine them as follows:

```
<div class="container-fluid container py-3 mt-4">
    <h2 class="h3">
        What guests are saying about homes in the United States
    </h2>
    <div class="row mt-4">
        <div class="col-md-3">
            <div class="card mb-3">
                <img
                    src="https://images.unsplash.com/
                    photo-1533654793924-4fc4949fb71e
                    ?ixlib=rb-1.2.1&
                    ixid=eyJhcHBfaWQiOjEyMDd9&
                    auto=format&
                    fit=crop&
                    w=1351&
                    q=80"
                    class="card-img-top"
                    alt="..."
                >
                <div class="card-body">
                    <h5 class="card-title">
                        Card title
                    </h5>
                    <p class="card-text">
                        This is a wider card with
                        supporting text below as a
                        natural lead-in to additional
                        content. This content is a
                        little bit longer.
                    </p>
                    <p class="card-text">
                        <small class="text-muted">
                            Last updated 3 mins ago
                        </small>
                    </p>
                </div>
            </div>
        </div>
        <div class="col-md-3">
            <div class="card mb-3">
                <img
                    src="https://images.unsplash.com/
                    photo-1520908695049-da13395b27a6?
```

```
                    ixlib=rb-1.2.1&
                    ixid=eyJhcHBfaWQiOjEyMDd9&
                    auto=format&
                    fit=crop&
                    w=1225&q=80"
                    class="card-img-top" alt="..."
                >
                <div class="card-body">
                    <h5 class="card-title">
                        Card title
                    </h5>
                    <p class="card-text">
                        This is a wider card with
                        supporting text below as a
                        natural lead-in to additional
                        content. This content is a
                        little bit longer.
                    </p>
                    <p class="card-text">
                        <small class="text-muted">
                            Last updated 3 mins ago
                        </small>
                    </p>
                </div>
            </div>
        </div>
        <div class="col-md-3">
            <div class="card mb-3">
                <img
                    src="https://images.unsplash.com/
                    photo-1501785888041-af3ef285b470?
                    ixlib=rb-1.2.1&
                    ixid=eyJhcHBfaWQiOjEyMDd9&
                    auto=format&
                    fit=crop&
                    w=1350&
                    q=80"
                    class="card-img-top" alt="..."
                >
                <div class="card-body">
                    <h5 class="card-title">
                        Card title
                    </h5>
```

```
                    <p class="card-text">
                        This is a wider card with
                        supporting text below as a
                        natural lead-in to additional
                        content. This content is a
                        little bit longer.
                    </p>
                    <p class="card-text">
                        <small class="text-muted">
                            Last updated 3 mins ago
                        </small>
                    </p>
                </div>
            </div>
        </div>
    </div>
</div>
```

For the home images, I'm using these three photos from Unsplash:

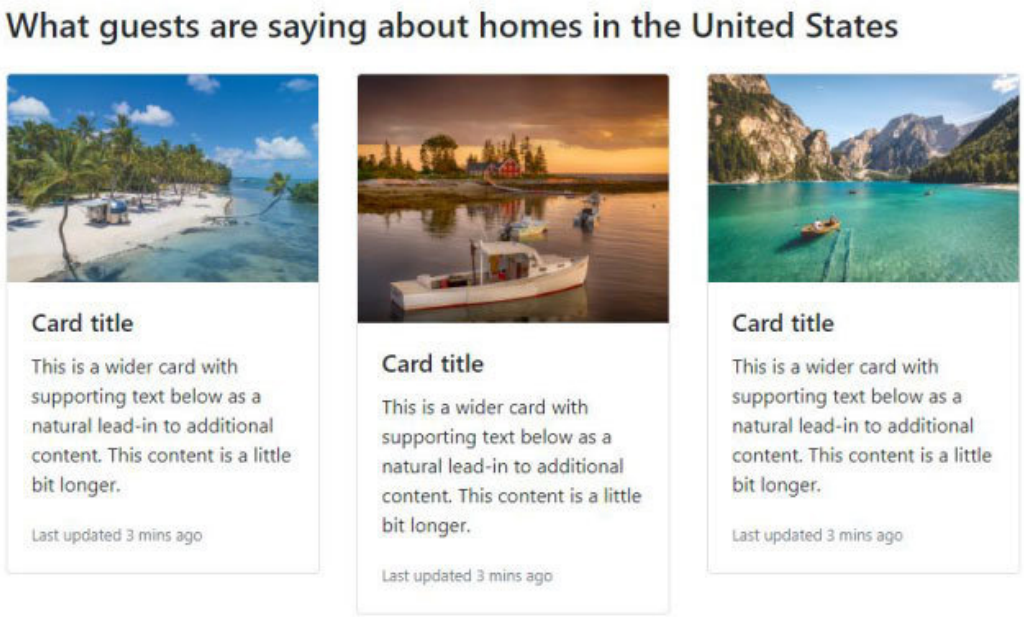
- beach by airstream[143]
- cape harbour by Keith Luke[144]
- wooden boat on blue lake[145]

Note: The slightly "weirdly" formatted code above is formatted as it is so as to cater for the limitations of the PDF format. This is especially true for the `img` elements' `src` attributes above, where I've split each of the query parameters onto the separate line. However, this is just a note to keep in mind the wrong formatting. If you copy-paste the code from the above example verbatim, it might not work in the browser. However, that's ok, because we've also committed those changes, and they look as shown in this update's commit.

This commit is titled Add testimonial cards[146].

Currently, it looks like this:

[143] https://unsplash.com/photos/e4nZLI9Wbpg
[144] https://unsplash.com/photos/Ad2BXIYc_gc
[145] https://unsplash.com/photos/T7K4aEPoGGk
[146] https://github.com/ImsirovicAjdin/Airbnb-clone-layout/commit/a62b1a3a81d22eb5f9a12f52e8e33c55eb84e640



Our layout after we've added the testimonial cards

These cards are using a lot of defaults and dummy text, since we've copied it from the official docs. We'll next update them so that they look more like the AirBnB homepage we're emulating.

For example, instead of a card title, we'll be adding three 5-star ratings, just like in the source layout.

10.11 Adding the 5-star ratings

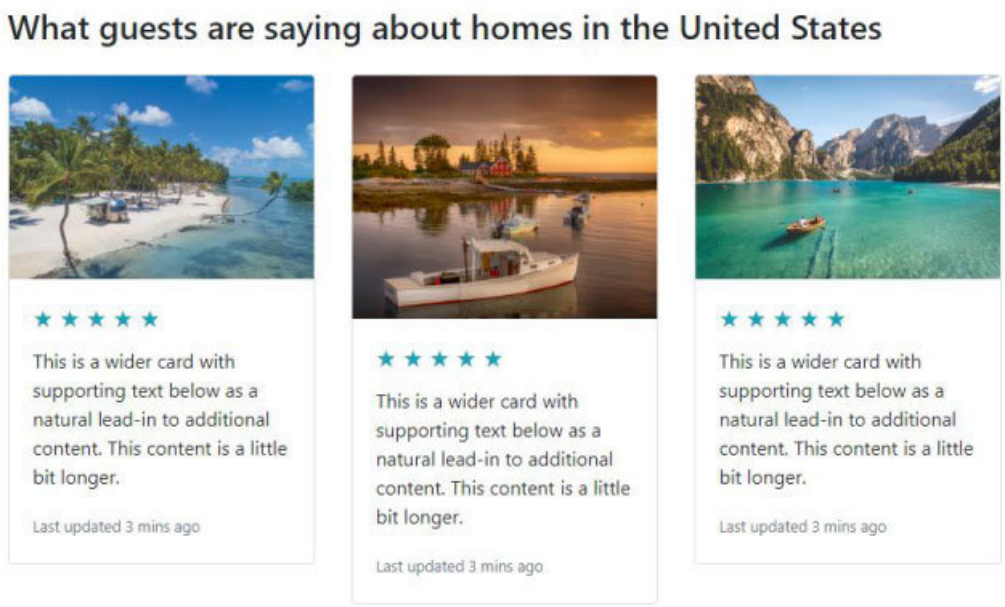
Let's update each `<h5>` element like this:

```
<h5 class="card-title text-info">
    &#9733; &#9733; &#9733; &#9733; &#9733;
</h5>
```

This commit is saved as Add a 5-star rating[147].

Here's the updated image:

[147] https://github.com/ImsirovicAjdin/Airbnb-clone-layout/commit/807f1cdd7a6a314a845e1eb1027a17cb44f7e206



Add a 5-star rating

Now we'll add the "media" cards.

Note that I'm putting the word media in quotes above because we're not really using Bootstrap's media object to build this part of the card.

Instead, we're just using some regular helper Bootstrap 4 classes, and we're combining them to get the desired effect that kind of looks like the media object in Bootstrap 4.

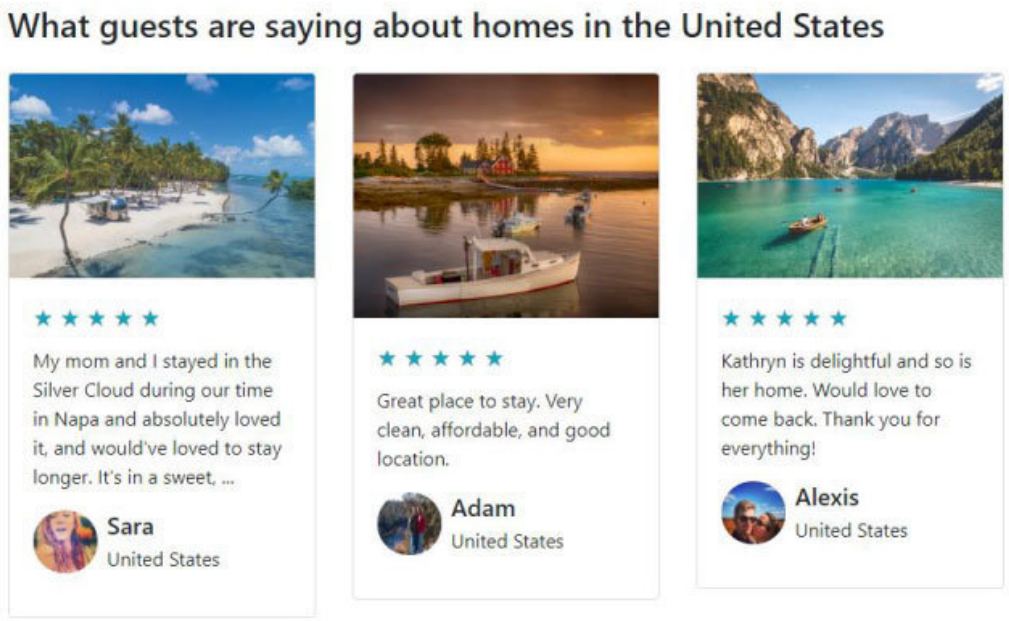
We'll replace this...

```
<p class="card-text">
    <small class="text-muted">
        Last updated 3 mins ago
    </small>
</p>
```

...with this:

```
<div class="">
    <div class="d-flex">
    <img
        src="https://a0.muscache.com/im/pictures/
        660e2273-b37b-4f50-917e-369e7bc3f665.jpg?
        aki_policy=large"
        class="ml-0 mr-2 rounded-circle w-25 mb-3"
        alt="..."
    >
    <div class="d-flex flex-column">
        <h5 class="mt-0 mb-1">Sara</h5>
        <p>United States</p>
    </div>
    </div>
</div>
```

The result will be as follows:



Update testimonial with people bios

Note that the images have been copied from the actual AirBnb homepage.

The commit message for this update is Add testimonial bios[148].

Next, we'll fix those image heights.

[148] https://github.com/ImsirovicAjdin/Airbnb-clone-layout/commit/d61f53d3ca9d7e420be76c40924d37373fd4969a

10.12 Add the same image height on all cards in Bootstrap 4

We need to extend each card's `card-img-top` class with the following CSS declarations:

```
.card-img-top {
    max-height: 160px;
    min-height: 160px;
    object-fit: cover;
}
```

We'll do that by appending the above CSS class to the existing custom CSS in the `<styles>` element, just above the closing `</head>` tag.

We've saved this update as the Add same height on card imgs[149] commit message.

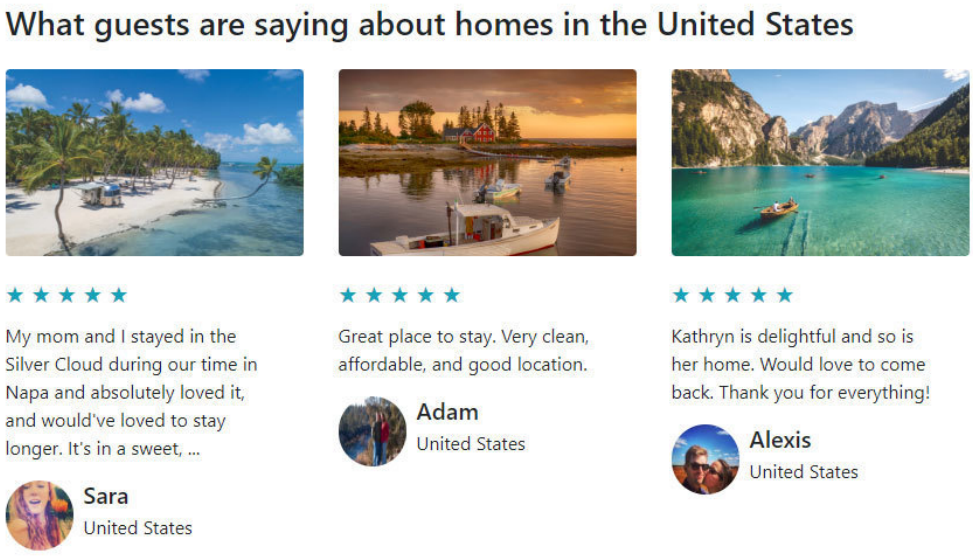
Finally, we have a few minor tweaks on the testimonials section:

- set the `border` property to `none` with this Bootstrap 4 CSS class: `border-0`
- remove the left-padding from `card-body` by adding the class of `pl-0` to it
- make images more like the AirBnB cards using the `rounded` class with `card-img-top`
- replace `mb-3` with `mb-2` classes on the testimonial bio images to make them closer to perfect circles

We'll save this update as Additional tweaks to testimonials[150] commit message.

Now the testimonals section is complete:

[149] https://github.com/ImsirovicAjdin/Airbnb-clone-layout/commit/d314612118c5fa2528ab89545509be2f8b5feaa8
[150] https://github.com/ImsirovicAjdin/Airbnb-clone-layout/commit/295bd0b0c287c9023d8e73c45c411dcf26838560



Completed testimonials section

10.13 Rebuilding the *Travelling with AirBnB* section

This section is similar to the previous one, minus the stars and the bio footers on each of the cards. Thus, we can begin by just copying already exisiting functionality from the previous section. Here's the first updated card, with the additional information removed:

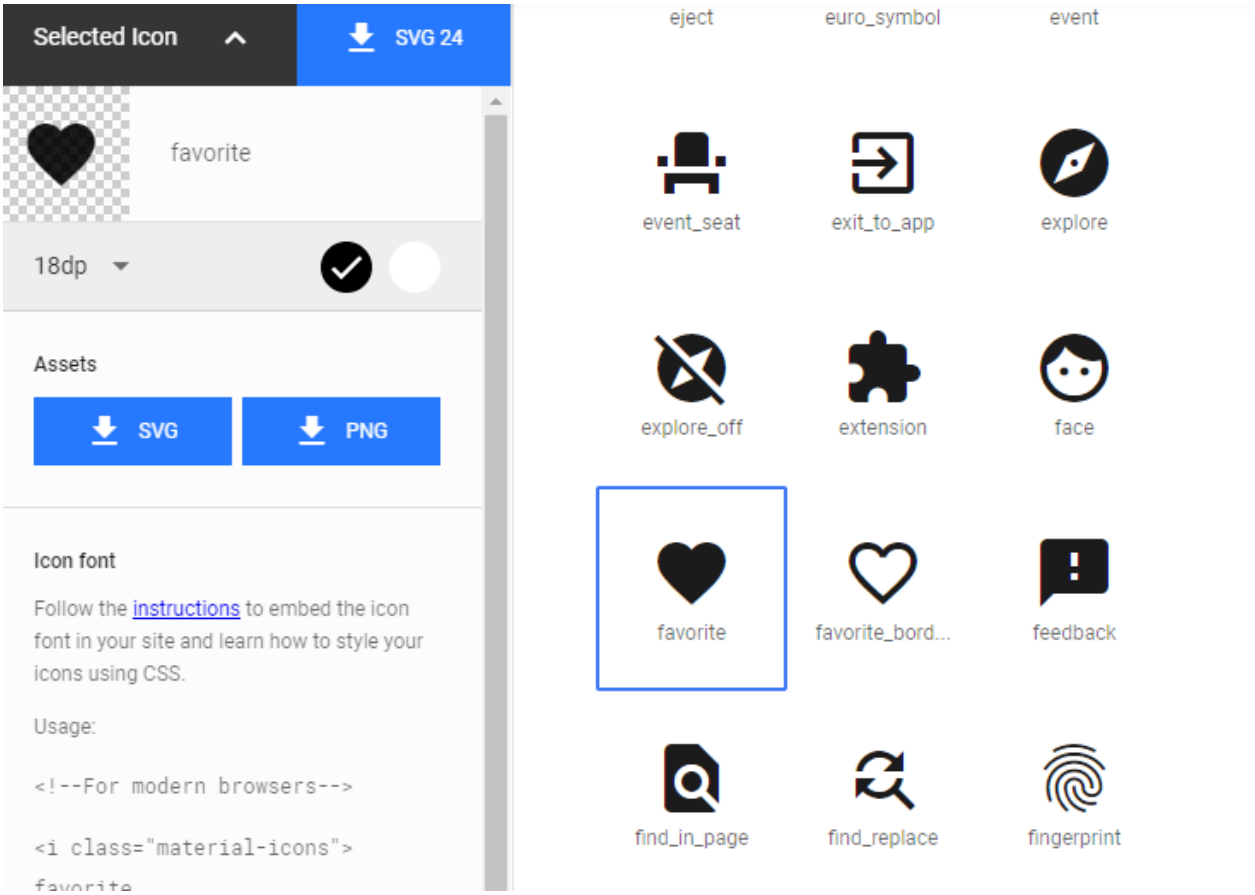
```
<div class="col-md-3">
    <div class="card mb-3 border-0">
        <svg></svg>
        <div class="card-body pl-0">
            <h5 class="card-title">
                24/7 customer support
            </h5>
            <p class="card-text">
                Day or night, we're here for you...
            </p>
        </div>
    </div>
</div>
```

In the place where there used to be an image, we're putting an `<svg>` element. SVG is the name of a format. It's not HTML, but it is quite similar to it. It stands for *Scalable Vector Graphics*, which makes it a perfect candidate for icons - and that's what we'll use.

Where do we get these icons though?

Let's navigate over to material design icons website[151] and get those icons that we need.

Once you find the heart-shaped icon, which is named "favorite", and you click on it to select it, the website will offer you two download options: SVG or PNG.



Heart-shaped icon download options

Let's click on SVG, and that will download the icon.

Once downloaded, let's open this icon in Brackets, then copy the entire contents of the file.

This is what we'll get:

[151] https://material.io/resources/icons/?style=baseline

```
<svg
viewBox="0 0 24 24"
xmlns="http://www.w3.org/2000/svg"
width="24"
height="24">
    <path d="M0 0h24v24H0z" fill="none" />
    <path d="M12 21.35l-1.45-1.32C5.4 15.36 2
             12.28 2 8.5 2 5.42 4.42 3 7.5 3c1.74 0
             3.41.81 4.5 2.09C13.09 3.81 14.76 3 16.5 3
             19.58 3 22 5.42 22 8.5c0 3.78-3.4
             6.86-8.55 11.54L12 21.35z" />
</svg>
```

Now that we've pasted in the first icon, let's find two additional icons.

Here's the `home` icon:

```
<svg
    xmlns="http://www.w3.org/2000/svg"
    width="24"
    height="24"
    viewBox="0 0 24 24">
    <path d="M10 20v-6h4v6h5v-8h3L12 3 2 12h3v8z" />
    <path d="M0 0h24v24H0z" fill="none" />
</svg>
```

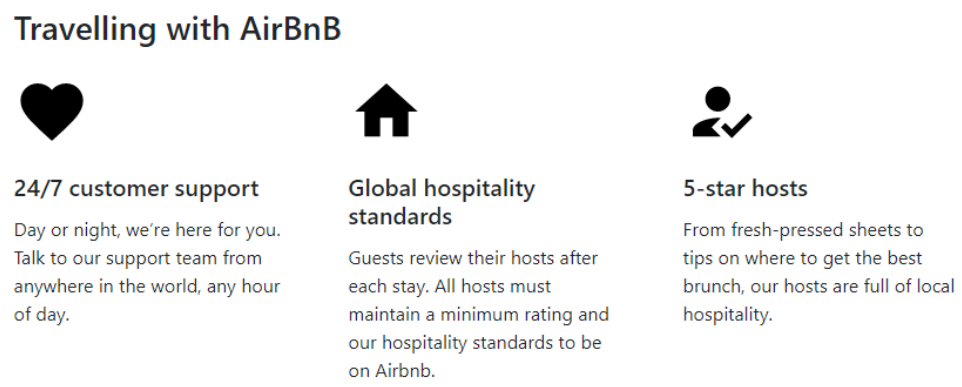
And here's the `how_to_reg` icon:

```
<svg
    xmlns="http://www.w3.org/2000/svg"
    width="24"
    height="24"
    viewBox="0 0 24 24"
>
    <path
        fill-rule="evenodd"
        clip-rule="evenodd"
        fill="none"
        d="M0 0h24v24H0z"
    />
    <g
        fill-rule="evenodd"
        clip-rule="evenodd"
```

```
16      >
17          <path
18              d="M9 17l3-2.94c-.39-.04-.68-.06-1-
19              .06-2.67 0-8 1.34-8 v2h9l-3-3zm2-5c2.21
20              0 4-1.79 4-4s-1.79-4-4-4-4 1.79-4 4
21              1.79 4 4 4"
22          />
23          <path
24              d="M15.47 20.5L12 17l1.4-1.41 2.07 2.08
25              5.13-5.17 1.4 1.41z"
26          />
27      </g>
28  </svg>
```

Note that this Material design icon font has its own icon names, and that they're sometimes unintuitive, so you might spend some time trying to locate a suitable icon.

With all these icons added, our third section now looks like this:



Travelling with AirBnB first change in this section

This is a good time for a commit, so let's add a new one with the message: Add SVG icons to third section[152].

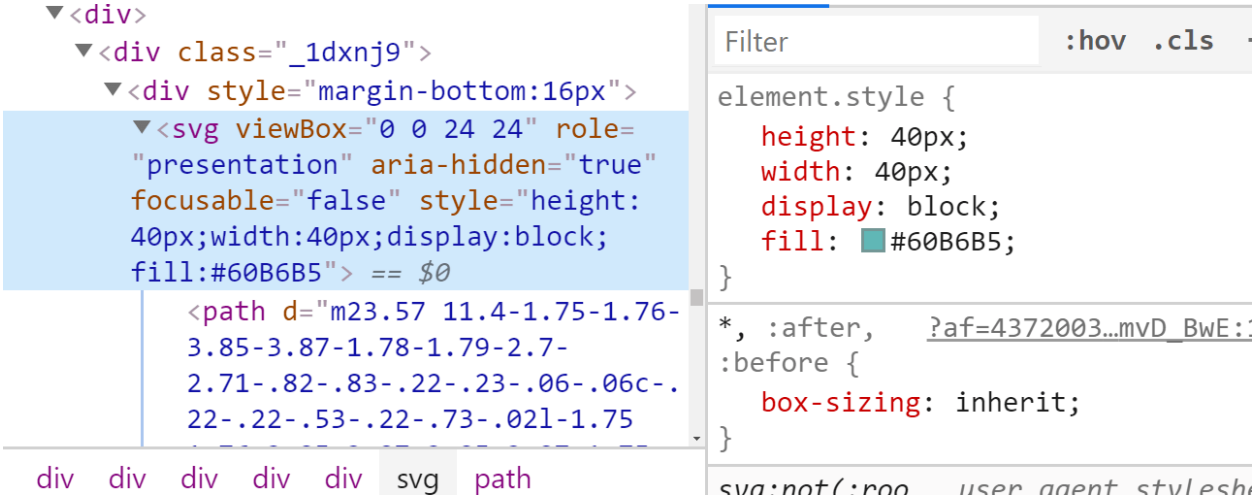
Let's next color the icons.

Since SVG is a different kind of format than HTML - altough very similar - we'll approach it differently.

We'll take the color from the original website by inspecting the original website's icons in developer tools in chrome. We've learned how to use the developer tools earlier in this book.

It shouldn't be hard to locate the `fill` color property inside any of the three original icons' `style` class. Note that these are also SVGs, as seen in the screenshot below:

[152]https://github.com/ImsirovicAjdin/Airbnb-clone-layout/commit/bbf095cacce254bcd243d067aa294a23af98a5f1



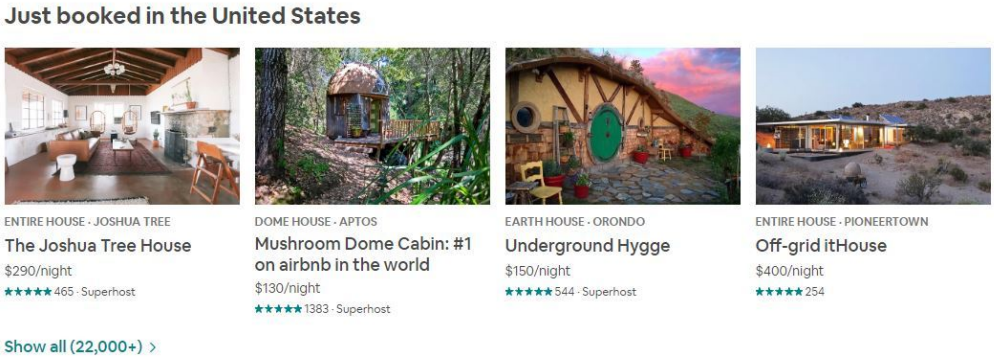
Screenshot of icon SVG structure on AirBnB website

The fill color for these icons, as we can see, is `#60B6B5`, and so let's add another commit, and add it this message: Add teal color to icons[153].

Next, we'll add the fourth section: *Just booked in the United States.*

10.14 Adding the fourth section to our AirBnB clone homepage

Looking at the fourth section, we can see a similar structure, only this time there are four cards on the same row:



Screenshot of the Just booked section

Looking at the above cards, we can see some similarity with the cards in section two of this rebuilt layout. Thus, let's begin by simply copying a card from the second section, and wrapping each card in a `col-md-3` class.

[153] https://github.com/ImsirovicAjdin/Airbnb-clone-layout/commit/11c67b4723d8a9598758f7db66c02cd0bda9cf0c

Next, we'll update the card's HTML code and look at the changes in Brackets editor's Live Preview, until we get the desired result:

```
<div class="col-md-3">
    <div class="card mb-3 border-0">
        <img
            src="https://images.unsplash.com/
            photo-1556912172-45b7abe8b7e1?
            ixlib=rb-1.2.1&
            ixid=eyJhcHBfaWQiOjEyMDd9&
            auto=format&
            fit=crop&
            w=1350&q=80"
            class="card-img-top br0" alt="..."
        >
        <p class="small text-uppercase pb-0">
            Entire house, Joshua Tree
        </p>
        <div class="card-body p-0">
            <h5 class="card-title">
                The Joshua Tree House
            </h5>
            <p class="card-text m-0">$290/night</p>
            <p class="small m-0 text-info">
                &#9733;&#9733;&#9733;&#9733;&#9733;
                    <span class="text-secondary">
                     465, Superhost
                    </span>
            </p>
        </div>
    </div>
</div>
```

We're also making sure to wrap all the cards inside a `div` with the class of `row`, and we're adding another `row` as a separate wrapper for the button that reads `Show all(22,000+)`. To make a button lose all its styling in bootstrap, we add it the following values in the `class` attribute:

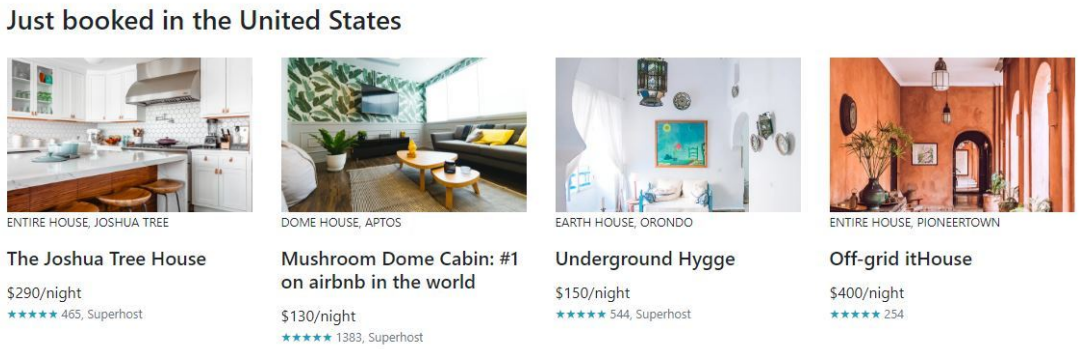
```
btn btn-link
```

To color the text in the button with any contextual class, we just use regular text coloring classes, so we get this HTML:

```
<div class="row mt-2 mb-5">
    <button class="btn text-info btn-link">
        <span class="h6">Show all(22,000+) &gt;</span>
    </button>
</div>
```

This update is saved as Complete the Just booked section[154] commit message on Github.

The updated section now looks like this:

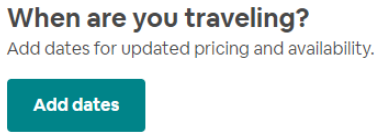


Screenshot of the just booked section in our own layout

Next, we'll add the fifth section to our AirBnB clone.

10.15 Add the *When are you travelling* section

On the source AirBnB homepage, the fifth section looks like this:



Screenshot of the When are you travelling section

This is a pretty easy thing to add.

We'll have a wrapping container, inside of which there will be a row. The row iself will wrap a `col-12` div, inside of which we'll have:

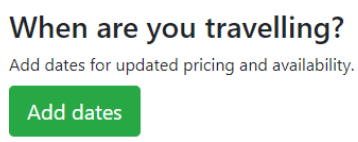
- an `h2` heading,
- a `p` tag, and
- a button

Here's the code for the complete section 5:

[154] https://github.com/ImsirovicAjdin/Airbnb-clone-layout/commit/e94642e2ae9f40060eb41aea47899c708afb7286

```
<div class="container-fluid container">
    <div class="container pl-0">
        <div class="row">
            <div class="col-12 pl-0 mb-5">
                <h2 class="h3 mt-0 mb-2">
                    When are you travelling?
                </h2>
                <p class="mt-0 ml-0 mb-2">
                    Add dates for updated pricing and availability.
                </p>
                <button class="btn btn-lg btn-success">
                    Add dates
                </button>
            </div>
        </div>
    </div>
</div>
```

With this code added to our own layout's fifth section, the update looks like this:



Screenshot of our own When are you travelling section

The commit message of this update is: Add the When are you travelling section[155].

All we've got left to do is add the footer now.

10.16 Adding the footer section

To quickly add the footer, let's find a suitable footer example on official Bootstrap 4 docs.

The footer on the pricing layout[156] is a nice starting point, so let's begin by copy-pasting that code into our AirBnB clone:

[155] https://github.com/ImsirovicAjdin/Airbnb-clone-layout/commit/5d40476bae5db1cb3732121cf809ae4fd3faf782
[156] https://getbootstrap.com/docs/4.3/examples/pricing/

```
<footer class="pt-4 my-md-5 pt-md-5 border-top">
    <div class="row">
        <div class="col-12 col-md">
            <img
                class="mb-2"
                src="/docs/4.3/assets/brand/bootstrap-solid.svg"
                alt=""
                width="24"
                height="24"
            >
            <small
                class="d-block mb-3 text-muted"
            >
                © 2017-2021
            </small>
        </div>
        <div class="col-6 col-md">
            <h5>Features</h5>
            <ul class="list-unstyled text-small">
                <li>
                    <a class="text-muted" href="#">
                        Cool stuff
                    </a>
                </li>
                <li>
                    <a class="text-muted" href="#">
                        Random feature
                    </a>
                </li>
                <li>
                    <a class="text-muted" href="#">
                        Team feature
                    </a>
                </li>
                <li>
                    <a class="text-muted" href="#">
                        Stuff for developers
                    </a>
                </li>
                <li>
                    <a class="text-muted" href="#">
                        Another one
                    </a>
```

```
                </li>
                <li>
                    <a class="text-muted" href="#">
                        Last time
                    </a>
                </li>
            </ul>
        </div>
        <div class="col-6 col-md">
            <h5>Resources</h5>
            <ul class="list-unstyled text-small">
                <li>
                    <a class="text-muted" href="#">
                        Resource
                    </a>
                </li>
                <li>
                    <a class="text-muted" href="#">
                        Resource name
                    </a>
                </li>
                <li>
                    <a class="text-muted" href="#">
                        Another resource
                    </a>
                </li>
                <li>
                    <a class="text-muted" href="#">
                        Final resource
                    </a>
                </li>
            </ul>
        </div>
        <div class="col-6 col-md">
            <h5>About</h5>
            <ul class="list-unstyled text-small">
                <li>
                    <a class="text-muted" href="#">
                        Team
                    </a>
                </li>
                <li>
                    <a class="text-muted" href="#">
```

```
                        Locations
                    </a>
                </li>
                <li>
                    <a class="text-muted" href="#">
                        Privacy
                    </a>
                </li>
                <li>
                    <a class="text-muted" href="#">
                        Terms
                    </a>
                </li>
            </ul>
        </div>
    </div>
</footer>
```

We're going to take social icon SVGs from simpleicons.org[157].

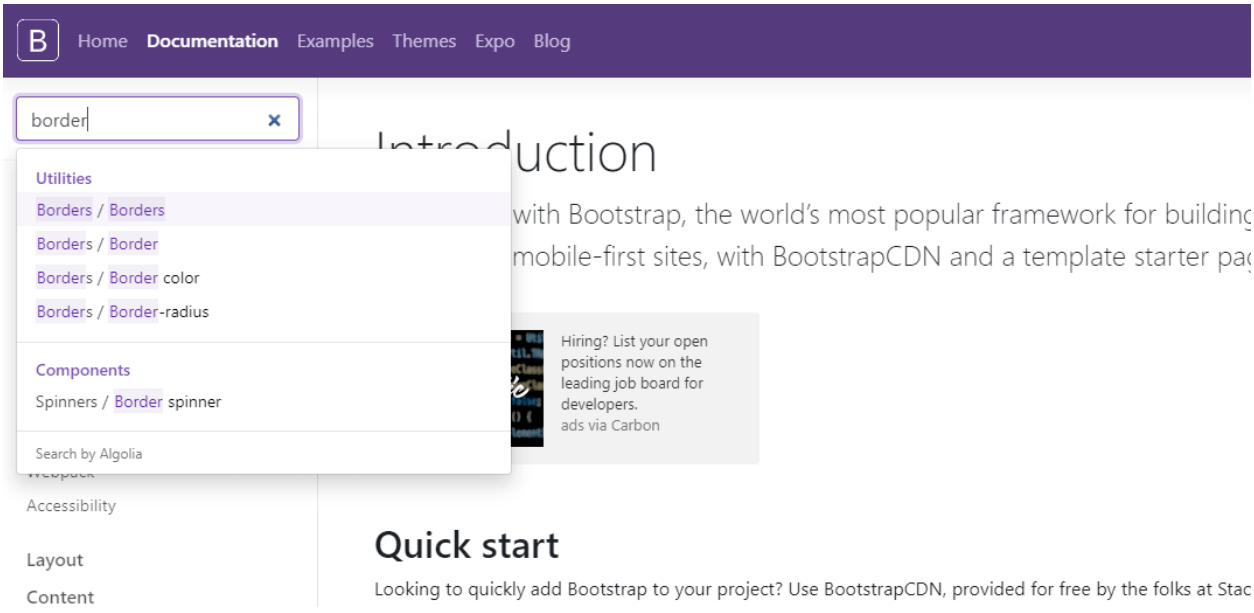
Other than that, you should mostly already understand what's going on it the footer code.

If by any chance you are still getting confused even at this point of our journey through building Bootstrap 4 layouts, you have at least two options:

1. Search keywords related to a CSS class name inside Bootstrap documentation
2. Inspect the element and its CSS using developer tools

Here's a screenshot of checking out CSS class `border-top`, which is a default Bootstrap 4 CSS class.

[157] https://simpleicons.org/



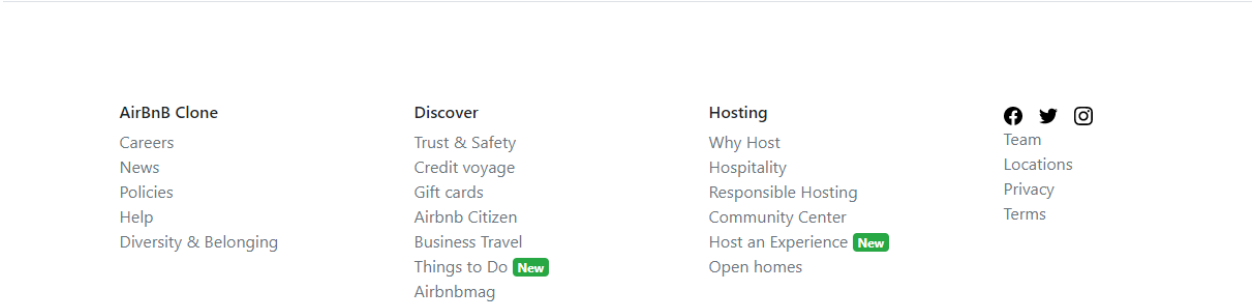
Screenshot of looking up the word border in Bootstrap documentation

We've already discussed how to inspect styles using devtools, so we won't cover it here.

This update is saved with the following commit message: Add the footer section[158].

Note that we've used a new component in the footer: the badge component[159].

The footer we've built looks like this:



Screenshot of the footer we've built

Let's now wrap up our repository with another commit: Format the layout code[160]. Like we already said earlier in the article, it is best to keep formatting changes to your code in separate commits.

[158] https://github.com/ImsirovicAjdin/Airbnb-clone-layout/commit/7b9f012e9fc6f563ce252b980a4bced26388bfc5
[159] https://getbootstrap.com/docs/4.3/components/badge/
[160] https://github.com/ImsirovicAjdin/Airbnb-clone-layout/commit/bfd03da4683334e5ed371fb137c5ea313d9793dd

10.17 Live preview of the AirBnB Bootstrap 4 clone

After all these updates completed, let's have a look at the completed AirBnB Bootstrap 4 clone[161].

That's it for this article. Let's revisit what we've learned.

Conclusion

In this chapter, we've improved our layout-building skills by building an AirBnB clone. The approach used was to get the maximum effect while using minimal custom CSS. We were relying mostly on the built-in Bootstrap 4 classes.

With this update, our AirBnB clone is complete and we can move onto building another layout.

In the next chapter, we'll build a Shopify clone.

[161] https://www.codingexercises.com/codelabs/2019-10-09-airbnb-bootstrap-4-clone

Chapter 11: Build a Shopify clone layout in Bootstrap 4

In this chapter we'll clone the Shopify website's homepage using Bootstrap 4.

Let's have a look at the screenshot of the actual site.

Since this layout is quite tall, we'll look at screenshots of each section separately - as we build it.

Altogether, there are 7 sections on this homepage. Let's inspect the actual HTML structure using devtools to "extract" the names of sections:

1. The navbar
2. Hero section (we usually call it jumbotron in Bootstrap)
3. Showcase section (inluding the heading and the start, sell, market, and manage sub-sections)
4. Support section (a simple container with 3 columns)
5. Merchants section (i.e "a success story" section)
6. Signup section (the "Start free trial" call to action)
7. Footer area (a bunch of links here!)

Let's start building our Shopify clone!

11.0 Setting up the project

To begin, just like we did in the previous chapter, let's setup a starter `index.html` file in our code editor, using Github Desktop to add Git tracking right from the start.https://www.codingexercises.com/codelabs/2019-10-11-shopify-bootstrap-4-clone-pt-3

As before, as soon as we add a local folder to a new repository with Github Desktop, a commit with the message of Initial commit[162] will be added.

Once we've added our `index.html` starter template with all the necessary Bootstrap imports and some starter text, we'll make another commit.

The commit message is Add starter index.html file[163].

Here's the live preview of our Shopify homepage clone[164] right now.

There's not much there to see on our preview right now, but if you've followed the series up to this point, you'll probably recognize the `display-1` Bootstrap 4 CSS class that styles this text. Next, we'll add the navbar.

[162] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/855dd3cc1562c367eb5aa688d84f661700c9da2c
[163] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/22babe38127b939798b469ec1ff81c7309e39fa5
[164] https://www.codingexercises.com/codelabs/2019-10-11-shopify-bootstrap-4-clone-pt-1/

11.1 Building the navbar

Let's start with the navbar:

Screenshot of shopify navbar area

To begin, we'll just copy-paste a navbar from Bootstrap 4 docs[165].

We're copying the navbar which has an image for `navbar-brand` on the left:

```
<!-- Just an image -->
<nav class="navbar navbar-light bg-light">
  <a class="navbar-brand" href="#">
    <img
        src="/docs/4.3/assets/brand/bootstrap-solid.svg"
        width="30"
        height="30"
        alt=""
    >
  </a>
</nav>
```

Of course, we'll need to use the image to the Shopify logo:

```
<!-- Just an image -->
<nav class="navbar navbar-light bg-light">
  <a class="navbar-brand" href="#">
    <!--
        We'll copy the SVG Shopify logo right from
        the devtools on the official Shopify homepage
    -->
  </a>
</nav>
```

In the next commit, we'll save the update with the Shopify SVG.

To properly size that SVG, we simply gave it a max-height, as can be seen here:

```
<svg id="logos-shopify-black" style="max-height:37px">
```

[165] https://getbootstrap.com/docs/4.3/components/navbar/#brand

The commit message for this update is Add properly-sized Shopify SVG[166].

Next, we'll add the rest of the navbar, as two separate `ul` tags.

We'll begin by copying the code from the very first navbar example on the Bootstrap docs, right under the Supported content heading[167].

```
<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>

<div
    class="collapse navbar-collapse"
    id="navbarSupportedContent"
>
    <ul class="navbar-nav mr-auto">
      <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="#">
            Link
        </a>
      </li>
      <li class="nav-item dropdown">
        <a
            class="nav-link dropdown-toggle"
            href="#"
            id="navbarDropdown"
```

[166] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/908de5f17c7856b0e1a2832ddac3ac9ba234a969
[167] https://getbootstrap.com/docs/4.3/components/navbar/#supported-content

```
            role="button"
            data-toggle="dropdown"
            aria-haspopup="true"
            aria-expanded="false"
        >
          Dropdown
        </a>
        <div
            class="dropdown-menu"
            aria-labelledby="navbarDropdown"
        >
            <a class="dropdown-item" href="#">
                Action
            </a>
            <a class="dropdown-item" href="#">
                Another action
            </a>
            <div class="dropdown-divider"></div>
            <a class="dropdown-item" href="#">
                Something else here
            </a>
        </div>
      </li>
      <li class="nav-item">
        <a
            class="nav-link disabled"
            href="#"
            tabindex="-1"
            aria-disabled="true"
        >
            Disabled
        </a>
      </li>
    </ul>
```

Note that the above code is incomplete - the `collapse navbar-collapse div` tag is not closed right. I just wanted to show you that you can start by just copy-pasting *the exact same code* from the docs.

We'll slightly tweak the above code, by adding another `ul` tag before closing the collapse div.

This will be another commit, and this one is titled: Add two ul tags to navbar[168].

The pdated, live, cloned layout can be found here[169].

[168] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/617c13d18a7e1dec47b31aba2a2ab07b708cfcf5
[169] https://www.codingexercises.com/codelabs/2019-10-11-shopify-bootstrap-4-clone-pt-2

We still have a wrinkle to iron out: all our links are hiding under the toggle button!

11.1.1 Setting the breakpoint for navbar toggle button

Our navbar is starting to look like the source Shopify homepage. However, we have this toggle button showing - even on Desktop resolutions - and it hides the actual navbar links.

To fix this, we'll simply update the existing opening nav tag from this:

```
<nav class="navbar navbar-light bg-light">
```

...to this:

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
```

By adding the `navbar-expand-lg` class to the `nav` tag, we have the toggle button disappear on resolutions greater than 1200 pixels.

How this works is explained under the Containers heading on navbar docs[170].

This update is commited as Hide toggle btn on desktop resolutions[171].

Next, using the developer tools, we'll copy some styles from the source theme, and add them to our layout's `<style>` tag, located just above the closing `</head>` tag:

```
<style>
    .bg-custom {
        background: #f2e0cf;
    }
    .nav-link,
    .nav-link:hover,
    .navbar-light .navbar-nav .nav-link {
        color: #212b35;
        font-size: 17px;
        font-weight: 500;
        letter-spacing: 0.02em;
    }
    .maxw1600 {
        max-width: 1600px;
    }
</style>
```

[170] https://getbootstrap.com/docs/4.3/components/navbar/#containers
[171] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/426d7dd23a99a7943a12ce23a514eff0a9a33649

The commit is titled: Improve navbar styling[172].

Next, we'll further improve both the styles and the structure, so that it looks a lot closer to the actual Shopify homepage. This involves many updates to the HTML structure (due to the fact that most links on the navbar are grouped into separate dropdowns).

We'll also update some missing styles. For example, we'll override the `btn-success` class by adding an additional custom class of: `btn-success-custom`.

```
.btn-success-custom {
    background: #5c6ac4;
}
```

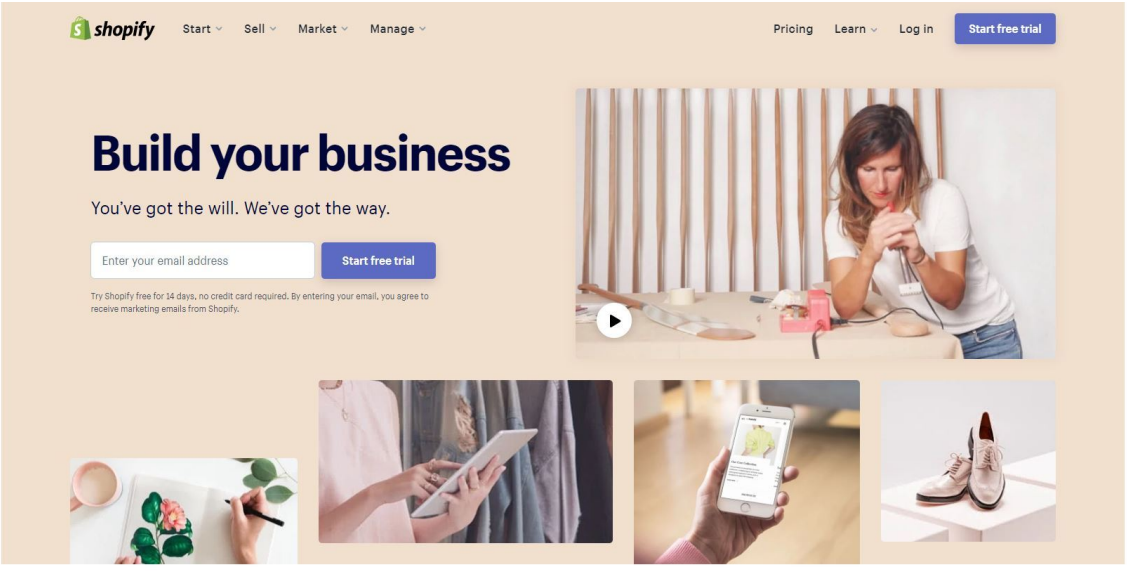
This improvement is saved in this commit message: Update navbar structure and styles[173].

Now our navbar is a lot closer to the source layout. This improvement can be seen in this codelab[174], as part 3 of our layout-building.

Next, we'll start working on the Hero section.

11.2 Hero section

Hero section is a trove of information, appearing above the fold:



Screenshot of Shopify homepage hero section

[172] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/0af08eed88c141558bacf7b6e7de355231497381
[173] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/7a2655471e7ddd115c2f355627a6aed846d4b202
[174] https://www.codingexercises.com/codelabs/2019-10-11-shopify-bootstrap-4-clone-pt-3

11.2.1 Above the fold

"Above the fold" is a term from the print media. People buy newspapers and often fold it in two so as to be able to carry them easier. The upper folded part is referred to as "above the fold". On the web, we sometime say that the topmost area of a website that fits on the monitor is the "above the fold" area.

Let's start building section two by adding containers, rows, and columns.

11.2.2 Adding in the containers, rows, and columns to section two

Let's begin by adding the two rows:

1. The first row will hold the *Start free trial* area and the video
2. The second row will hold the four images

Thus, we'll add this code to begin with:

```
<div class="container-fluid">
    <div class="bg-custom container maxw1600">
        <div class="row">
            <div class="col-md-6 p-5 border border-secondary">
                Build your business
            </div>
            <div class="col-md-6 p-5 border border-secondary">
                Shopify video from Youtube here
            </div>
        </div>
        <div class="row">
            <div class="col-md-3 border p-3 border-light">
                first image
            </div>
            <div class="col-md-3 border p-3 border-light">
                second image
            </div>
            <div class="col-md-3 border p-3 border-light">
                third image
            </div>
            <div class="col-md-3 border p-3 border-light">
                fourth image
            </div>
        </div>
    </div>
</div>
```

Adding the code above to our layout, will produce the following result:

Start adding the second section

What we did above was to add the `border border-light` or `border border-secondary` Bootstrap 4 CSS classes to spot the areeas we'll be working on easier.

To make it even easier to distinguish between them, we've given each column in the first row the padding of `p-5`, and each column in the second row the padding of `p-3`.

The commit for this update is titled: Start adding the second section[175].

Next, let's fill in these two rows with real content!

We'll begin with some low-hanging fruits and add a YouTube video to the second column in the first row. This is the *Embed* code copied from the video's share button on YouTube:

```
<iframe
    width="560"
    height="315"
    src="https://www.youtube.com/embed/JthaEfEsLYg"
    frameborder="0"
    allow="accelerometer; autoplay; encrypted-media; gyroscope; picture-in-picture"
    allowfullscreen>
</iframe>
```

Let's add it in inside the first row:

[175] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/263ba6a767feea4474457e3c2d70c51d269c90b4

```
<div class="row">
    <div class="col-md-6 p-5 border border-secondary">
        Build your business
    </div>
    <div class="col-md-6 p-5 border border-secondary">
        <!-- Shopify video from Youtube here -->
        <iframe
            width="560"
            height="315"
            src="https://www.youtube.com/embed/JthaEfEsLYg"
            frameborder="0"
            allow="accelerometer; autoplay; encrypted-media; gyroscope; picture-in-p\
icture"
            allowfullscreen>
        </iframe>
    </div>
</div>
```

Note that the above code has been formatted in a way that will hopefully make it easier to read.

This update is saved in commit titled Add the video[176].

The updated layout can be seen live here[177].

Here is the result of this latest update:

A screenshot of the layout after adding the video])(11-04-after-adding-the-video.png)

The video preview is showing, and that's a good thing. However, we still need to make the video responsive. Currently it's width is hard-coded to 560 pixels, and it's height is hard-coded to 315 pixels.

11.2.3 Making an embedded video responsive with Bootstrap 4

To make the embedded videos responsive, we'll use the Bootstrap 4 responsive embeds CSS classes[178].

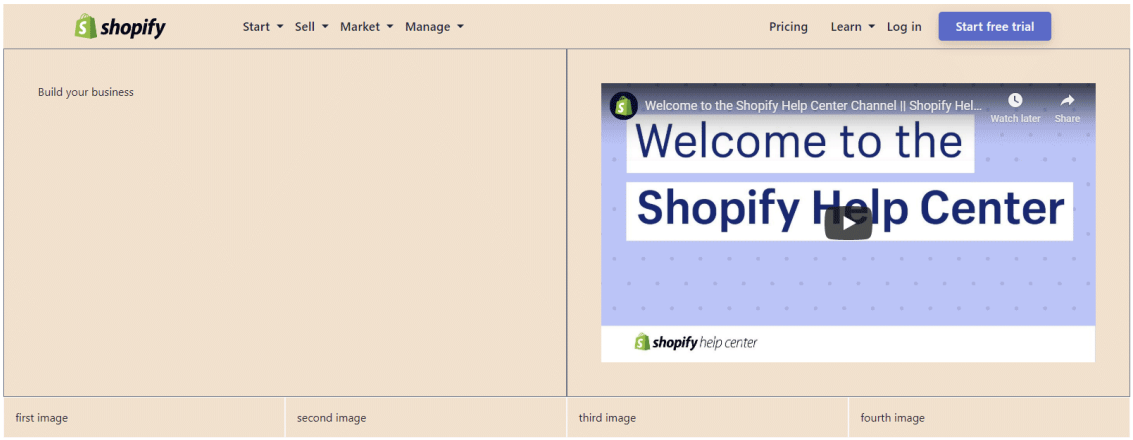
After inspecting the docs, we can see that a proper update to our embedded YouTube video would be this:

[176] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/c65993fdee44adae88e8768bbcc87d9e1ef451dc
[177] https://www.codingexercises.com/codelabs/2019-10-11-shopify-bootstrap-4-clone-pt-4
[178] https://getbootstrap.com/docs/4.3/utilities/embed/#aspect-ratios

```
<div class="col-md-6 p-5 border border-secondary">
    <!-- Shopify video from Youtube here -->
    <div class="embed-responsive embed-responsive-16by9">
        <iframe
            class="embed-responsive-item"
            src="https://www.youtube.com/embed/JthaEfEsLYg"
            frameborder="0"
            allow="accelerometer; autoplay; encrypted-media; gyroscope; picture-in-p\
icture"
            allowfullscreen>
        </iframe>
    </div>
</div>
```

This update is commited under this message: Add responsive embed classes to the video[179].

Now our layout looks like this:



A screenshot of the layout after adding the embed responsive video CSS

As we can see above, the embedded video now takes up the full width of its wrapping div - minus the `p-5` padding, of course.

Next, let's add the signup form in the first column of the first row.

11.2.4 Adding the signup form

Forms are a big topic in Bootstrap 4[180]. There are many ways to work with forms in Bootstrap 4. It's the same in regular HTML, when no framework is used.

[179] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/cb17c0b1ea894a674fd5849140e4ddf36a500d05
[180] https://getbootstrap.com/docs/4.3/components/forms/

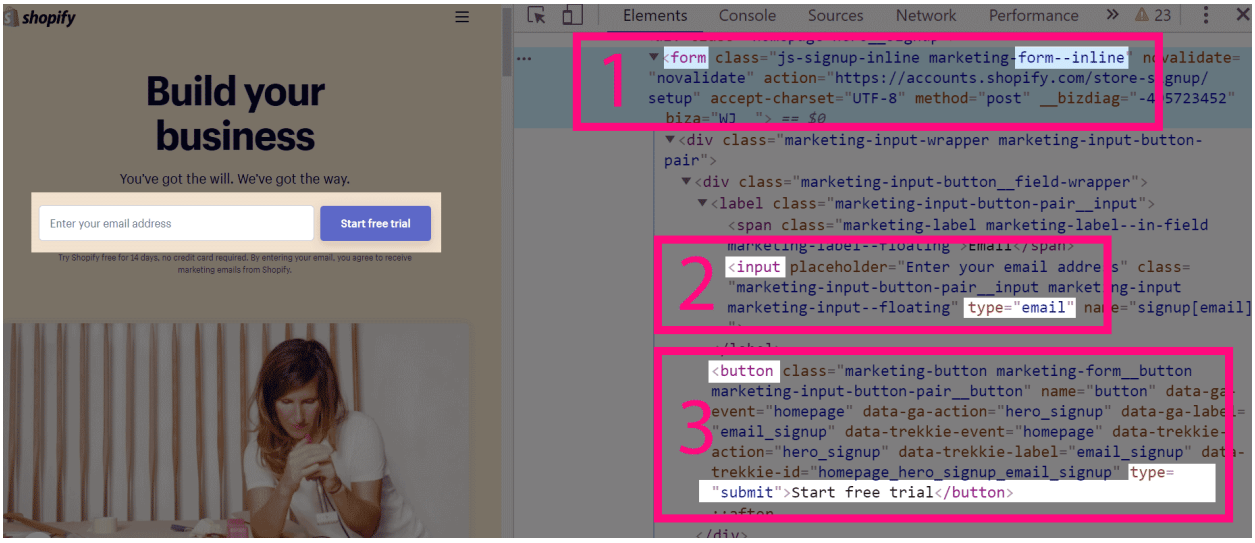
For example, there are 22 different values you can add to the `type` attribute on an `input` element in HTML:

1. email
2. password
3. checkbox
4. radio
5. file
6. text
7. submit
8. range
9. button
10. color
11. date
12. datetime-local
13. hidden
14. image
15. month
16. number
17. reset
18. search
19. tel
20. time
21. url
22. week

These 22 values are for the `<input>` HTML element alone. There are many other elements that we can use in HTML forms. Actually, HTML forms could have a series of articles on their own!

11.2.5 Inspecting the signup form on the Shopify homepage

This is what it looks like:



Analyzing the source form structure

Looking at the form structure above, we see some areas marked in numbers:

1. there's a wrapping `<form>` element with a custom `form--inline` CSS class
2. there's an `<input>` element, and its `type` attribute is set to the value of `email`
3. there's a `<button>` element, and its `type` is set to `submit`

Now we have at least a basic idea of what we need to do:

1. We need to have an *inline form* with two elements: an `<input>` element, and a `<button>`
2. The `type` attribute of the first element will be set to `email`

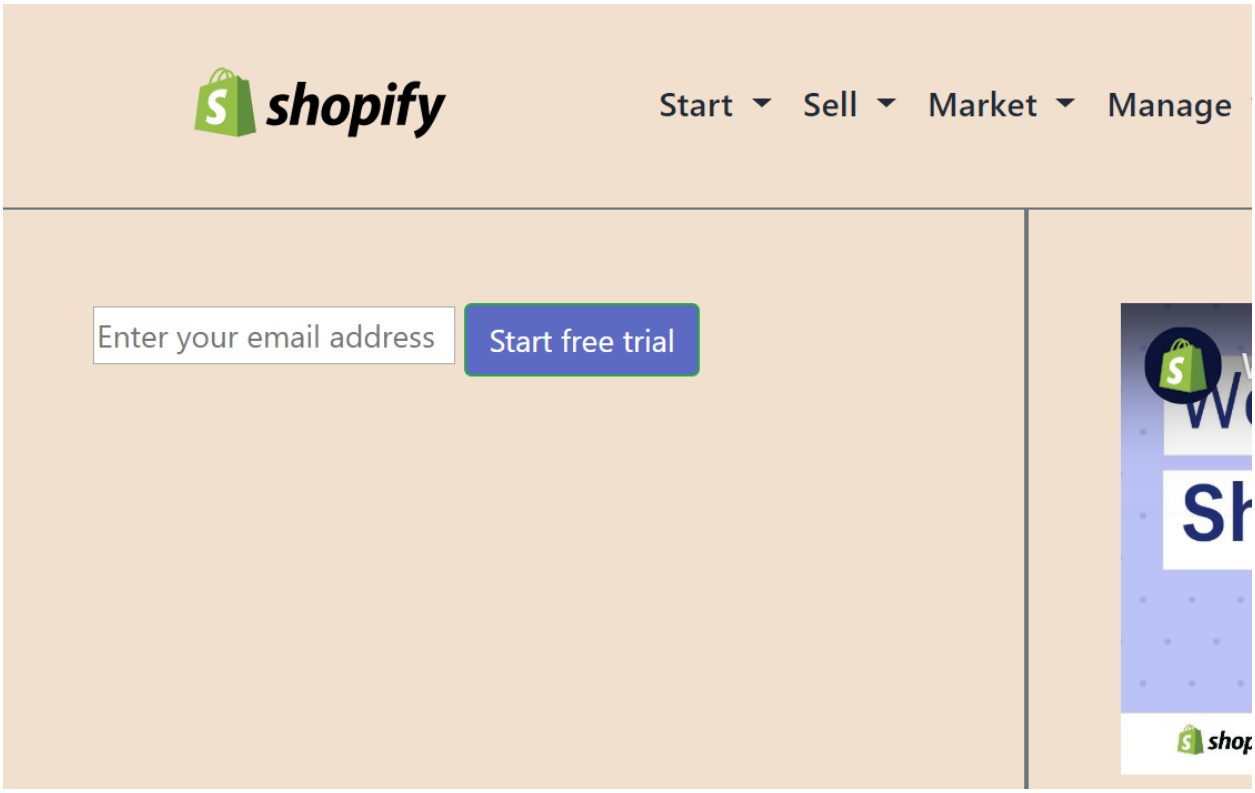
3, We will have a `<button>` element with the value of `submit`

Here are the above conclusions, translated into code:

```
<form>
    <input placeholder="Enter your email address" type="email">
    <button type="submit" class="btn btn-success btn-success-custom">
        Start free trial
    </button>
</form>
```

As we can see in the above code, there are only a few Bootstrap 4 CSS classes we're using on the `<button>` element.

After adding the update to our layout, this is what it looks like:



Adding the basic form structure

The commit message for this improvement is Add the basic form structure[181].

Now we can start adding the Bootstrap 4 form-specific CSS classes to make our form look better.

11.2.5 Making our forms look better with Bootstrap 4

On the official docs, there's an example of a horizontal form[182].

Here's one section of the example's code that we're interested in in particular:

```
<form>
  <div class="form-group row">
    <label for="inputEmail3" class="col-sm-2 col-form-label">Email</label>
    <div class="col-sm-10">
      <input type="email" class="form-control" id="inputEmail3" placeholder="Email">
    </div>
  </div>
  ...
</form>
```

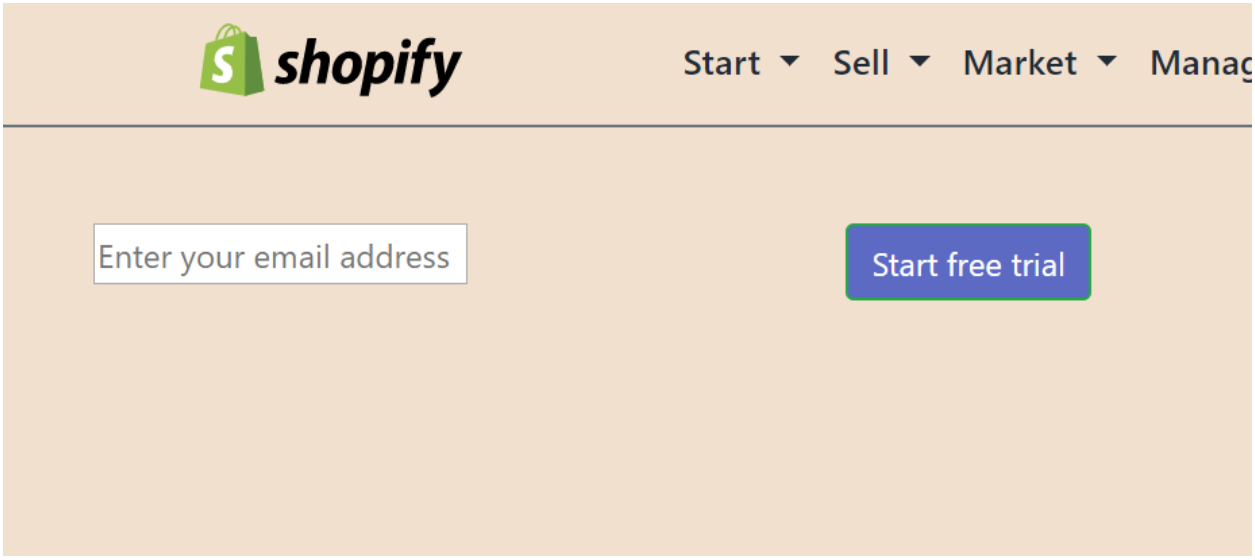
[181] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/380a99325bbb834a95738e1aedbf0335e73103ef
[182] https://getbootstrap.com/docs/4.3/components/forms/#horizontal-form

As the docs say, we add horizontal forms by grouping the labels, inputs, and other elements using the `row` class. Inside the `row`, each separate form member is assigned it's own `col-*-*` class.

Looking at the *Start free trial* form on the Shopify homepage, we can see that the input takes about 8 cols, while the button takes about 4. So let's try updating our code to this:

```
<form>
    <div class="form-group row">
        <label for="inputEmail" class="sr-only">Email</label>
        <div class="col-sm-8">
            <input placeholder="Enter your email address" type="email">
        </div>
        <div class="col-sm-4">
            <button type="submit" class="btn btn-success btn-success-custom">
                Start free trial
            </button>
        </div>
    </div>
</form>
```

Once we save this change in our code editor, our live preview will update the web page to this:

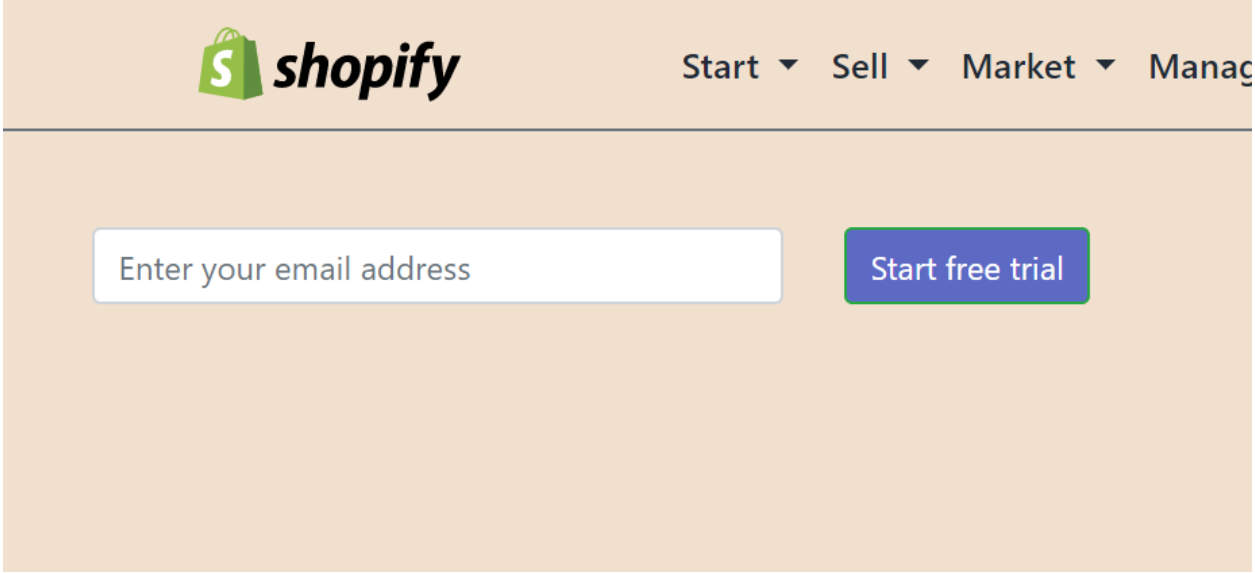


Updated the basic form structure

We did get the column grid working, but our input is still unstyled. We'll fix that easily, by adding the `form-control` class on the input, like this:

```
<input
    class="form-control"
    placeholder="Enter your email address"
    type="email"
>
```

With this little update, our form is starting to take shape:



Improve the look of inputs with the form-control CSS class

This is a nice time for a commit, so let's add it, with the message of Add Bootstrap 4 horizontal form[183].

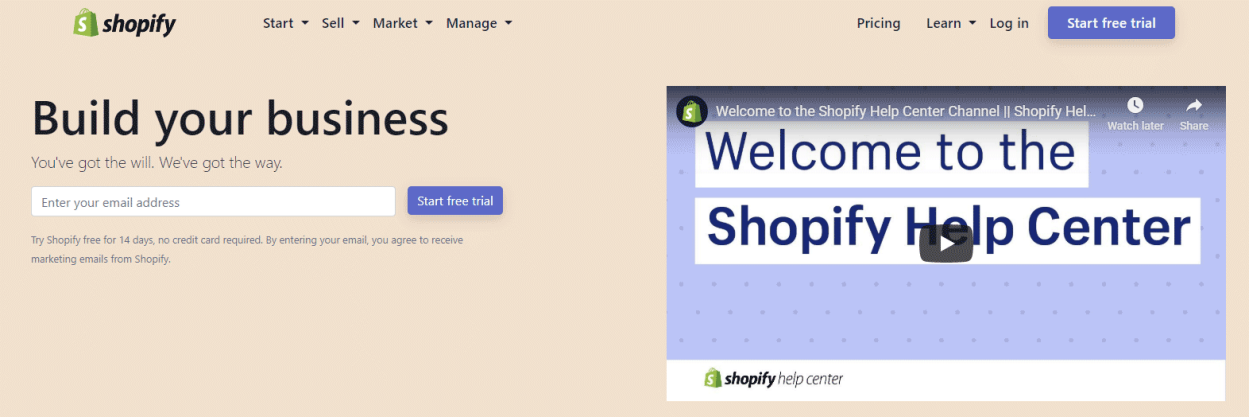
Next, we'll add the missing sections around our form: the *Build your business* title, and the motivational text under it: *You've got the will. We've got the way.*

We'll also add the small text under the input: *Try Shopify free for 14 days*, etc.

At this point of our article series, you should be able to add those changes yourself. That's why we won't discuss these improvements, but rather just add the code and commit the update as Complete the free trial form area[184].

This update makes our layout look like this:

[183] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/078566f81d61f591e82bc2faae5322d0eefe94ea
[184] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/078566f81d61f591e82bc2faae5322d0eefe94ea



Complete the free trial form area

Next, before we move on to section 3, we'll take a quick de-tour to discuss a very important topic: flexbox in Bootstrap 4.

11.2.6 Flexbox in Bootstrap 4

We've already covered flexbox in the HTML and CSS basics series[185].

Bootstrap 4 comes with many flexbox classes that closely mimic the actual CSS property-value pairs.

For example, the `align-items-center` CSS class in Bootstrap 4 has this CSS code:

```
.align-items-center {
    -ms-flex-align: center!important;
    align-items: center!important;
}
```

This is the code we'll apply on our `row` to make the content vertically centered:

```
<div class="row align-items-center">
```

Even though this is a very small change, we'll still add it as a new commit, titled Align items in center[186].

Here's the live preview of our site now[187].

Finally, before moving on to the next section, we'll just add the images in the second row, and commit the update as Add the images in the second row[188].

[185] https://www.codingexercises.com/guides/html-and-css-basics-part-23#flexbox-is-one-dimensional
[186] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/ddecde6ca665d17f7bb89e4ca4fe4081eaab5663
[187] https://www.codingexercises.com/codelabs/2019-10-11-shopify-bootstrap-4-clone-pt-5
[188] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/ddecde6ca665d17f7bb89e4ca4fe4081eaab5663

If you're wondering about the class names that have the capital "X" added on them, this is simply a quick and easy way to "turn off" a CSS class from our HTML, *without actually erasing it*. Thus, later, you can easily "turn it back on" if you need to: you just erase the capital "X".

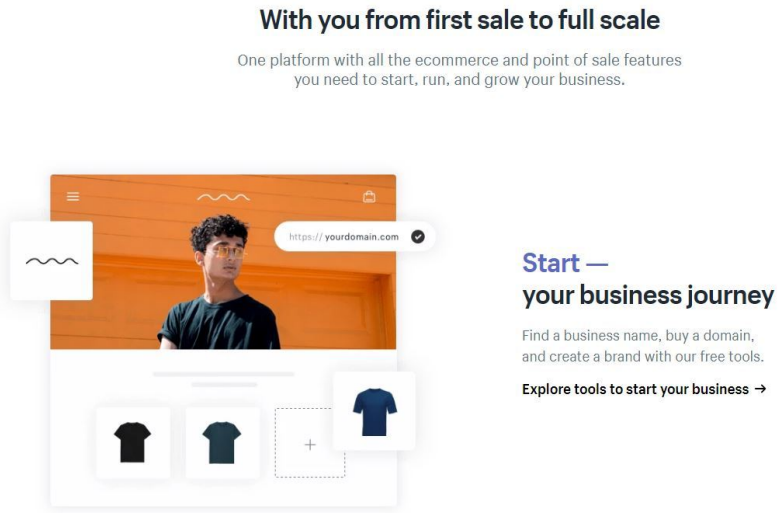
After we've added the images, here's the live website update[189].

Now we can move on to the showcase section.

11.3 Showcase section

The showcase area of Shopify's homepage is quite big, so we've split it in two parts.

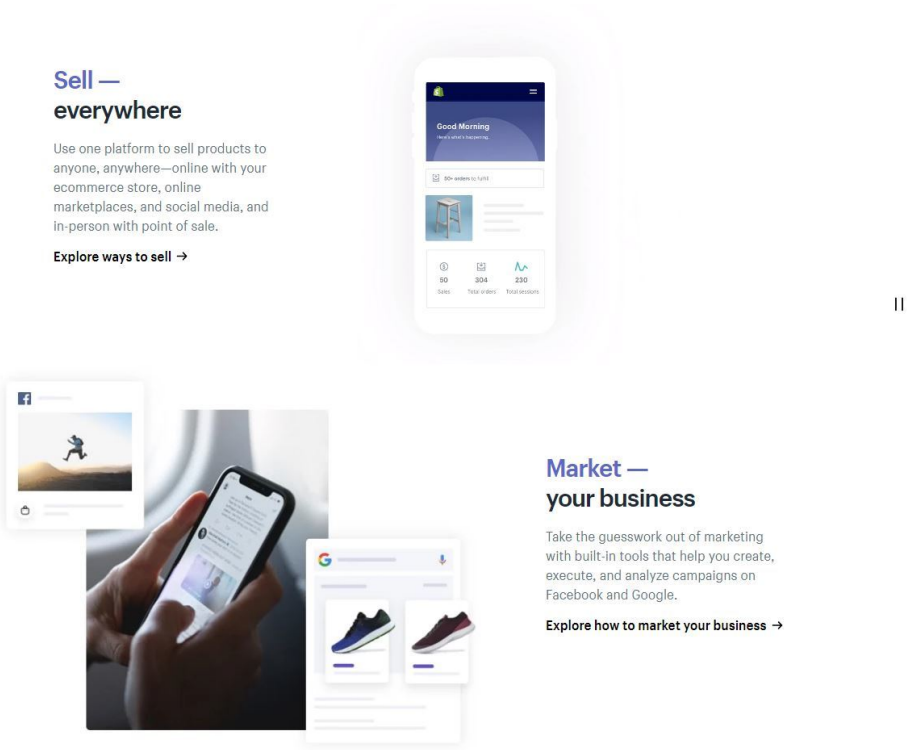
Here's part one...



Screenshot of Shopify homepage showcase section, part 1

and here's the part two:

[189] https://www.codingexercises.com/codelabs/2019-10-11-shopify-bootstrap-4-clone-pt-6



Screenshot of Shopify homepage showcase section, part 2

Looking at these two parts of section three, we can see that their structure is very simple. After the heading and a paragraph with the `lead` class, we have three rows, with two six-column divs in each row, and the content of the rows is alternating: image - text, text - image, image - text.

After observing these simple patterns, we can convert them into code:

```
<!-- section 3 -->
<div class="container-fluid ">
    <div class="container">

        <div class="text-center mt-5 py-5  d-flex flex-column align-items-center">
            <h2 class="h1 w-75 mb-4">
                With you from first sale to full scale
            </h2>
            <p class="lead w-50">
                One platform with all the ecommerce and
                point of sale features you need to
                start, run, and grow your business.
            </p>
        </div>

        <div class="row align-items-center">
```

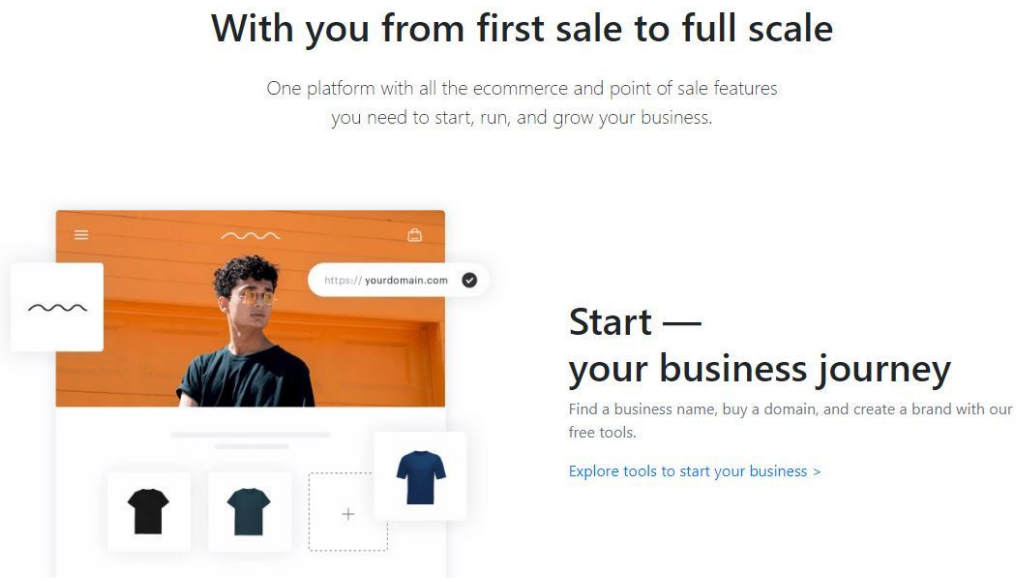
```
            <div class="col-md-6 px-5 order-2">
                <h2 class="h1">
                    Start – <br>
                    your business journey
                </h2>
                <p class="text-muted">
                    Find a business name, buy a domain,
                    and create a brand with our free
                    tools.
                </p>
                <p class="strong">
                    <a href="#" style="text-decoration: none">
                        Explore tools to start your
                        business &gt;
                    </a>
                </p>
            </div>
            <div class="col-md-6 bg-successX order-1">
                <img
                    src="https://cdn.shopify.com/
                    assets2/homepage/
                    start-en-small-99ac3533dfebcb2e
                    00592dce004d52f4d623ed0707
                    faf7422203ce4094f7c1bd.png"
                    class="img-fluid rounded"
                >
            </div>
        </div>
    </div>
</div>
```

Note that we didn't have to add any custom CSS here, except for the one on the anchor tag:

```
<a href="#" style="text-decoration: none">
```

Other than that, all our code is using regular Bootstrap 4 classes. It's amazing how useful the default Bootstrap 4 CSS classes can be - with just a little bit of tinkering, you can produce some pretty good results!

Our section 3 now looks like this:



Screenshot of the started Showcase section

The update is saved in a commit with the following message: Start the showcase section[190].

Since section 3 is pretty easy to complete, let's quickly add the rest of the code.

There are a couple of things worth mentioning in this update. First of all, the showcase section uses both images and video, which is a pretty neat design choice. However, to make things easier, I've decided to copy suitable images from other sections of the site.

Unfortunately, these images came in the form of SVGs, so the third image in our own Showcase section is actually an SVG image, with ***lots*** of code needed to have it look like it's supposed to. I'm not sure why Shopify chose to add a complex SVG when they could have just used a png. Maybe they want to provide better experience to viewers with high pixel density devices?

Anyway, here's our commit message: Complete the showcase section[191].

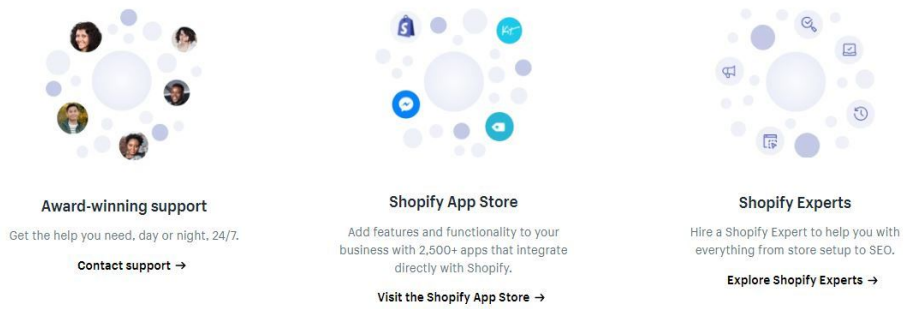
The updated layout can be seen live here[192].

Next, we'll update the support section.

11.4 Support section

The support section is a simple row with three cards in three columns:

[190] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/22483a5161f85c3419b1bc238d936146f020dcfb
[191] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/7a9d568dccac07035dc1a25bcab407accd4c4399
[192] https://www.codingexercises.com/codelabs/2019-10-11-shopify-bootstrap-4-clone-pt-7



Screenshot of Shopify homepage support section

Adding the support section should be very easy. You *should* try doing it on your own.

Here's what I came up with: In the first commit, I completed the Add the containers and columns[193] part - i.e, I've set up the basic structure:

```
<!-- section 4 -->
<div class="container-fluid bg-dark p-5">
    <div class="container bg-warning p-5">
        <div class="row">
            <div class="col-md-4 bg-danger p-5">
                1st image
            </div>
            <div class="col-md-4 bg-danger p-5">
                2nd image
            </div>
            <div class="col-md-4 bg-danger p-5">
                3rd image
            </div>
        </div>
    </div>
</div>
```

Note the temporary use of contextual background color classes and `p-5` utility spacing class above. This helps me see the immediate effect of adding this HTML structure.

Next, let's add in the images and the text[194] (using the images from the Shopify homepage):

[193] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/0a04ffe03f29299f07c2044d61d6c7399aea1b31
[194] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/1c9a00f9b7cbb4f074c89894eb84739e6e56a2a9

```
<!-- section 4 -->
<div class="container-fluid">
    <div class="container">
        <div class="row">
            <div class="col-md-4">
                <img
                src="https://cdn.shopify.com/assets2/homepage/contact-support-small-\
e82b47be9ed2932f6d88af22e3839941c9cf4a3fd2f2e4e3eef956a4c3d4695f.png"
                class="img-fluid rounded">
                <h3 class="h3">
                    Award-winning support
                </h3>
                <p>
                    Get the help you need, day or night, 24/7.
                </p>
                <p>
                    <a href="#" style="text-decoration: none">
                        Contact support
                    </a>
                </p>
            </div>
            <div class="col-md-4">
                <img
                src="https://cdn.shopify.com/assets2/homepage/app-store-small-c5a0c8\
7cea6be45e72976b87377f1ea91a48d98a03b4f8d82e37aeadce4ad54d.png"
                class="img-fluid rounded">
                <h3 class="h3">
                    Shopify App Store
                </h3>
                <p>
                    Add features and functionality to your business with 2,500+ apps\
 that integrate directly with Shopify.
                </p>
                <p>
                    <a href="#" style="text-decoration: none">
                        Visit the Shopify app store
                    </a>
                </p>
            </div>
            <div class="col-md-4">
                <img
                src="https://cdn.shopify.com/assets2/homepage/experts-small-d9665c16\
53a6f4b6ba7e509598160660129827b3cffcae9d259392e7f5b83994.png"
```

```
                class="img-fluid rounded">
                <h3 class="h3">
                    Shopify Experts
                </h3>
                <p>
                    Get the help you need, day or night, 24/7.
                    Hire a Shopify Expert to help you with everything from store set\
up to SEO.
                </p>
                <p>
                    <a href="#" style="text-decoration: none">
                        Explore Shopify Experts
                    </a>
                </p>
            </div>
        </div>
    </div>
</div>
```

Note that in this commit we also removed the temporary contextual background classes and spacing utility `p-5` CSS class.

Finally, we'll tweak the centering of the images and the text[195] so it looks more like the source layout.

Here's the live site[196] with the support section completed.

11.5 Merchants section

The merchants section shows off an actual success story:

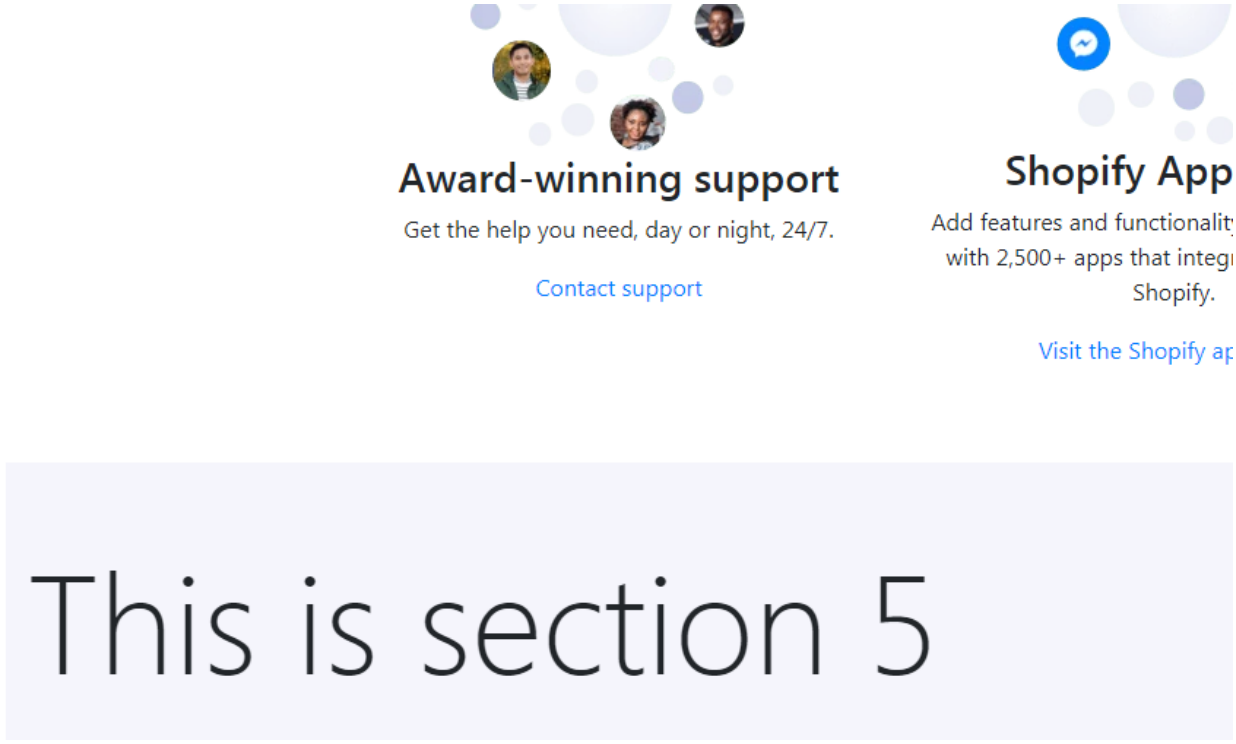
[195] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/b26813cc24899342ddd8a7383c98c529b3447c11
[196] https://www.codingexercises.com/codelabs/2019-10-11-shopify-bootstrap-4-clone-pt-8/

Screenshot of Shopify homepage merchants section

Let's begin by adding section 5 containers and custom background[197].

This is what our section 5 looks like with this update:

[197] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/ccbd536802139fad86c5aabf803b0ff6301f91bf



Screenshot of Shopify homepage support section

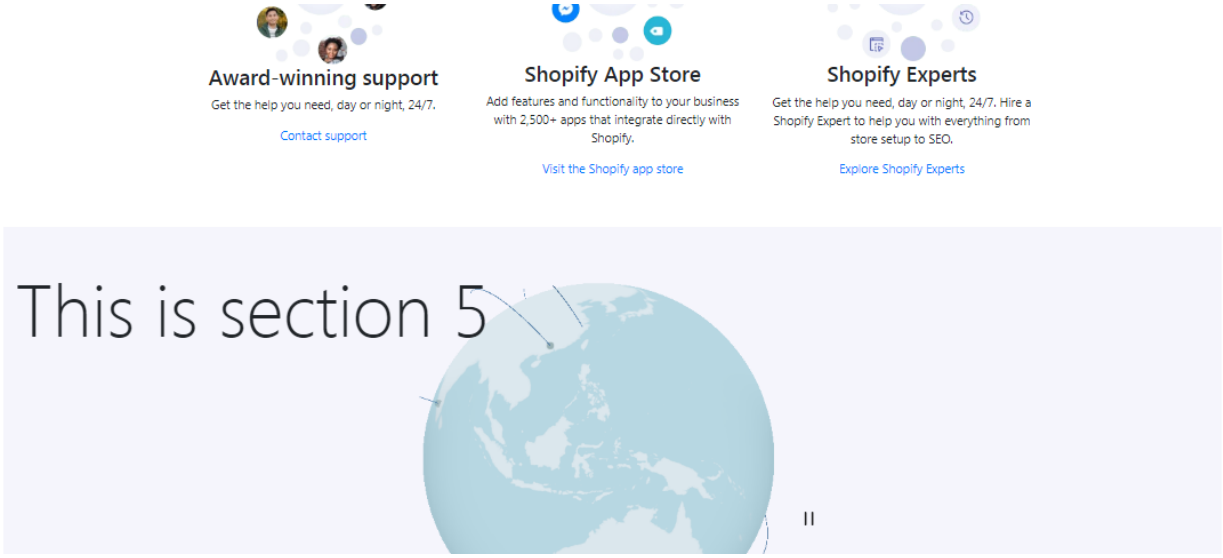
On the Shopify homepage, the merchant's section has a video background.

To keep things simple, we'll instead add a background image to this section's container.

We've already seen how easy it is to add a background image in previous articles in this article series, so let's just commit the update and give it the message of Add background image to merchants section[198].

Now the merchants section looks like this:

[198] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/dba74dc307140b6366cae1b06b7a907f5c7f974d



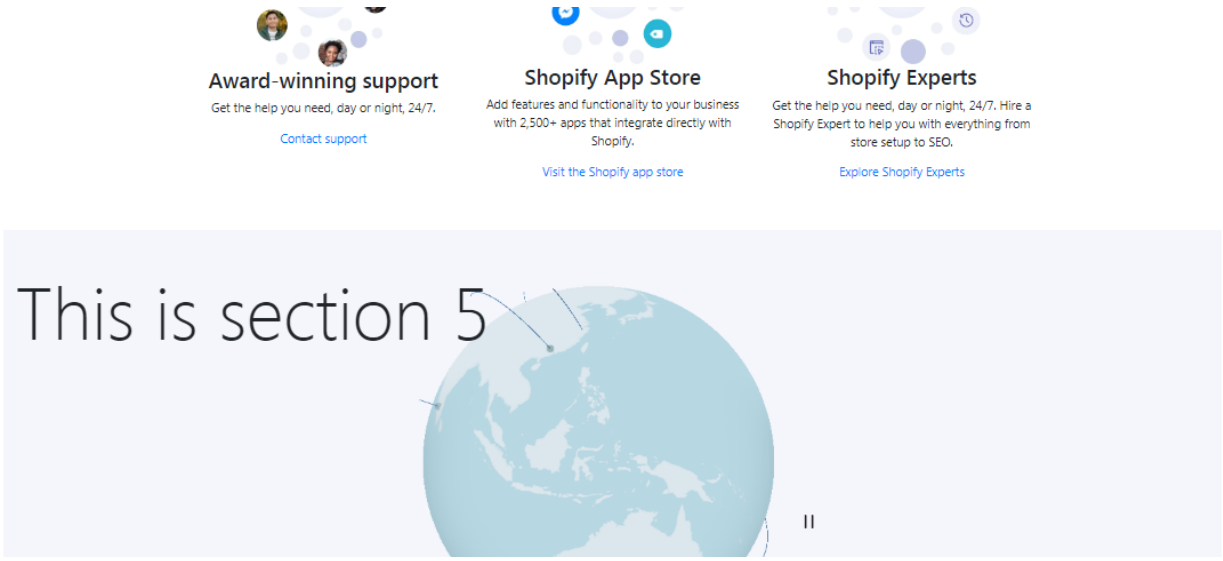
Screenshot of Shopify homepage support section

On the Shopify homepage, the merchant's section has a video background.

To keep things simple, we'll instead add a background image to this section's container.

We've already seen how easy it is to add a background image in previous articles in this article series, so let's just commit the update and give it the message of Add background image to merchants section[199].

Now the merchants section looks like this:



Screenshot of Shopify homepage support section

Next, we'll be adding the actual content to our merchants section, in the form of two divs. The left div will hold an image of a success story - a couple of entrepreneurs.

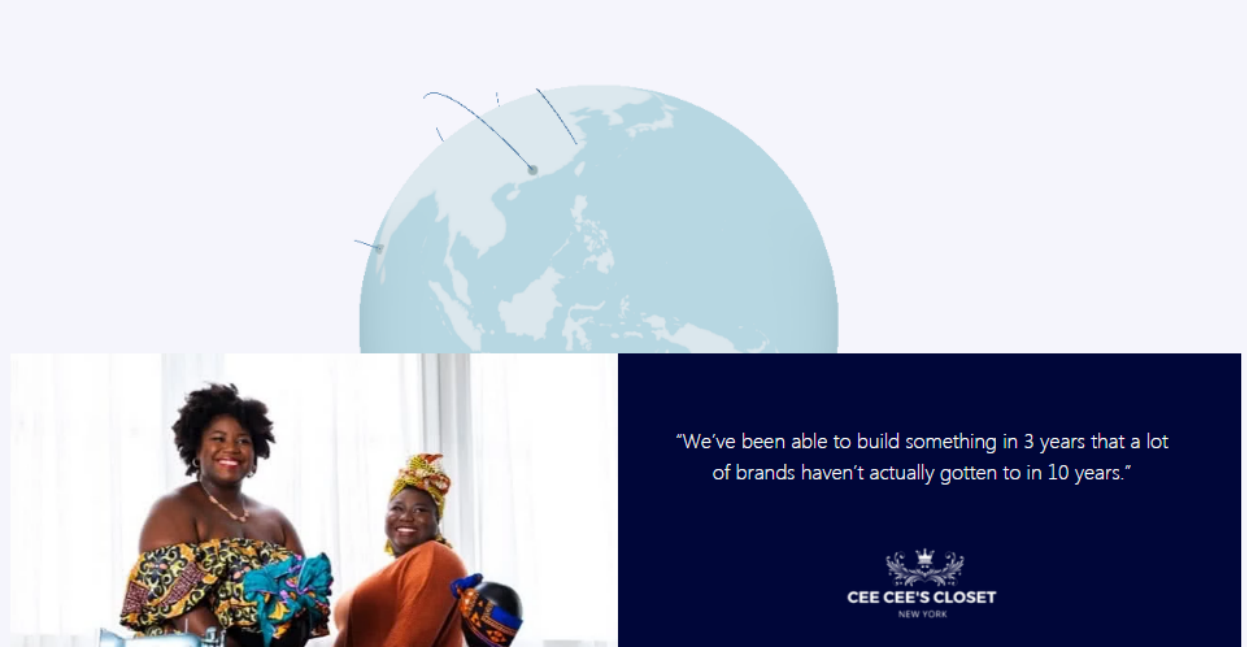
[199] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/dba74dc307140b6366cae1b06b7a907f5c7f974d

The right div will hold some text that gives additional information about the entrepreneurs and has a call to action button.

Let's try building it using mostly Bootstrap 4 default CSS classes.

The most recent update is commited with the message: Add two columns in merchants section[200].

This update looks as follows:



Screenshot of Shopify homepage support section

All that's left now for our merchant's section is to add it the heading, so let's commit this update as Add the heading and lead p for merchants[201].

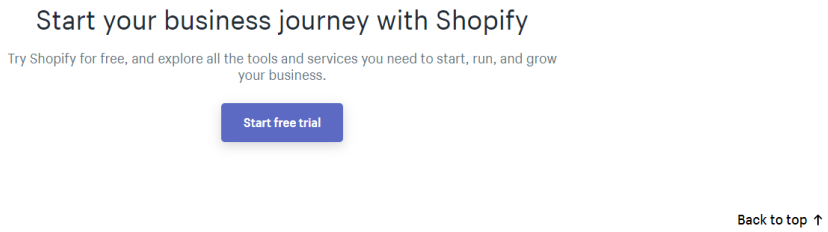
Our layout's merchant section now looks pretty close to the original Shopify homepage, as we can see in the updated live layout[202].

Next, we'll add the signup section.

11.6 Signup section

The signup section is pretty minimalistic:

[200] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/4c6d7b29530d545a5e544abb047a37f5efcaffae
[201] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/be6b3e0a7e719d079bd8363665d669b46b2e3c0b
[202] https://www.codingexercises.com/codelabs/2019-10-11-shopify-bootstrap-4-clone-pt-9



Screenshot of Shopify homepage signup section

This complete section is actually very simple:

```
<div class="container-fluid">
    <div class="container maxw1600 py-5">
        <div class="row py-5">
            <div class="col-12 text-center">
                <p class="h1">
                    Start your business journey with Shopify
                </p>
                <p class="lead">
                    Try Shopify for free, and explore all the tools and services you\
 need to start, run, and grow your business.
                </p>
                <button class="text-light br5 btn btn-success btn-success-custom btn\
-lg px-4 border-0" href="#">
                    Start free trial
                </button>
            </div>
        </div>
    </div>
</div>
```

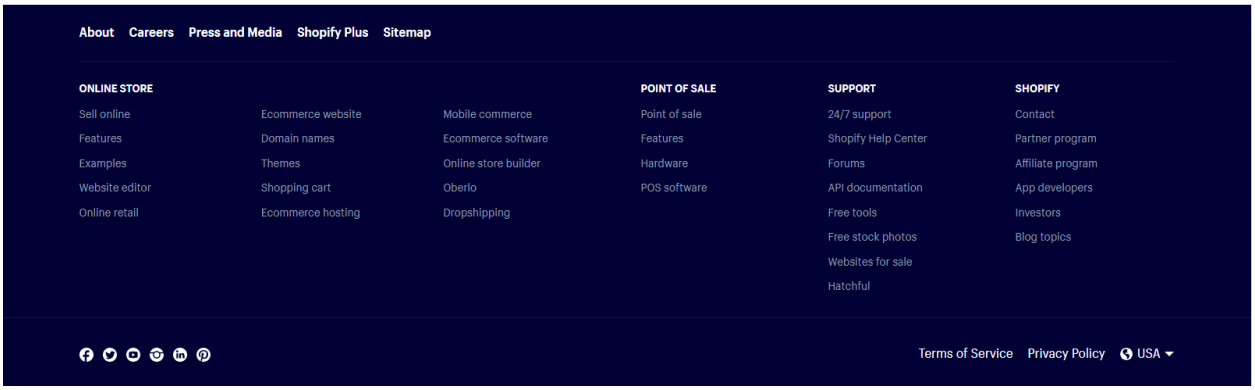
We'll commit this update with the message of Add the signup section[203].

Now we're ready to finish our Shopify clone; all we have to add is the footer.

11.7 Footer area

The footer area has a bunch of links, but this won't be difficult to build:

[203] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/12b92c4666d086eea94e76c0fb581c6617a2f759



Screenshot of Shopify homepage footer section

As usual, we'll start our section by setting up containers, rows, and columns:

```
<!-- section 7 -->
<div class="container-fluid bg-custom3 text-light py-5">
    <div class="container maxw1600">
        <div class="row">
            <div class="col-lg-2">
                one
            </div>
            <div class="col-lg-2">
                two
            </div>
            <div class="col-lg-2">
                three
            </div>
            <div class="col-lg-2">
                four
            </div>
            <div class="col-lg-2">
                five
            </div>
            <div class="col-lg-2">
                six
            </div>
        </div>
    </div>
</div>
```

Let's commit it as Add footer's containers, rows, and cols[204].

[204] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/cf22a63717410348defad4b6592621447e02a511

That's it for this article.

To quickly add our footer links, we'll use the list group component from Bootstrap 4[205].

We'll add this code inside each of the cols:

```
<h3 class="h5">Heading</h3>
<ul class="list-group list-group-flush">
    <li class="list-group-item py-1 pl-0 bg-custom3 mt-3">
        Cras justo odio
    </li>
    <li class="list-group-item py-1 pl-0 bg-custom3">
        Dapibus ac facilisis in
    </li>
    <li class="list-group-item py-1 pl-0 bg-custom3">
        Morbi leo risus
    </li>
    <li class="list-group-item py-1 pl-0 bg-custom3">
        Porta ac consectetur ac
    </li>
    <li class="list-group-item py-1 pl-0 bg-custom3">
        Vestibulum at eros
    </li>
</ul>
```

Note that the second and third column have a different margin-top class: instead of `mt-3`, we're using `mt-5`.

This update is saved in the commit titled Update section 7 with list-group-items[206].

The remaining updates in the footer section would just be reiterating what we've already built, so we're not going to do it.

Our layout is 95% complete, and if you feel like exercising your skills, feel free to update the footer and make the rest of the layout even closer to the source Shopify homepage.

In the meantime, you can checkout the completed live layout[207].

With this, we close this chapter.

[205] https://getbootstrap.com/docs/4.3/components/list-group/#flush
[206] https://github.com/ImsirovicAjdin/Shopify-homepage-clone/commit/c5c506fdef78c9976b9f6eebec90c80285226fa3
[207] https://www.codingexercises.com/codelabs/2019-10-11-shopify-bootstrap-4-clone-pt-10

Chapter 12: Conclusion

This is the end of the book. It's time to reflect on all the things we've learned.

12.1 Concepts covered

We've covered many concepts in this book.

We've practiced:

- using Git and GitHub Desktop,
- tracking our layout-building with it,
- copy-pasting code from other layouts on the web
- using developer tools in the browser
- thinking about responsive code and implementing it in our layouts
- using jQuery plugins
- the very basics of SCSS
- the very basics of Angular

However, we also haven't covered a number of important concepts.

12.2 Where to go from here?

When considering an answer to this question, it helps to know which topics were not covered, as this is a possible route that we can take in becoming better front-end developers.

Here's a list of conclusions and ideas about things that were not included (or skimmed over) in this book:

1. Bootstrap is just a framework
2. CSS has evolved significantly in the recent years (including the grid layout and the flexbox layout becoming mainstream)
3. Bootstrap comes with a bunch of components
4. How should we write CSS at scale (if we're not using Bootstrap)

Let's go through each of these.

12.2.1 Bootstrap is just a framework

It's obvious, but worth mentioning. Being "just a framework" means several things:

- There are other frameworks that we're not familiar with. Some examples are Bulma, Materialize, etc.
- There are many other underlying concepts that Bootstrap takes care of under the hood, such as accessibility out-of-the-box. This means that the best practices are "built in" and we don't even have to think about them when using Bootstrap

12.2.2 CSS has evolved significantly in the recent years

This is true for a number of concepts in CSS. And it is especially true for flexbox and grid.

Actually, these topics are so big that they deserve a book of their own.

12.2.3 Bootstrap comes with a bunch of components

Bootstrap comes with a bunch of components.

Here are the components from version 5, listed alphabetically:

1. Accordion
2. Alerts
3. Badge
4. Breadcrumb
5. Buttons
6. Button group
7. Card
8. Carousel
9. Close button
10. Collapse
11. Dropdowns
12. List groups
13. Modal
14. Navs & tabs
15. Navbar
16. Offcanvas
17. Pagination
18. Popovers
19. Progress
20. Scrollspy

21. Spinners
22. Toasts
23. Tooltips

We have covered some of these components in depth, while we barely touched on some others, or skipped them altogether.

The focus of this book was on *empowering* readers to become well-versed so that they can deal with components that we haven't covered in the book, on their own.

However, to truly get to the mastery level, we should be able, know, and understand, how to, for example, build a modal from scratch.

In the example of the modal, this raises the following questions:

1. What to do when the modal is taller then the viewport?
2. How to prevent the body under the modal from scrolling?
3. How to implement the *close-the-modal* feature by pressing the ESCAPE key on the keyboard?
4. How to implement the *close-the-modal* feature by clicking outside of the modal
5. How to properly structure the HTML of the modal?
6. How to deal with accessibility issues?

Building a modal with the above considerations is out of the scope of *this* book.

However, *rebuilding all the components from scratch*, without a framework, is a way forward in improving our skills. It's something we should do, and something that will be done, later on this book series.

Additionally, there are a few components that even the Bootstrap framework doesn't have. An example is some special versions of the carousel, such as the one that has multiple images with the right-most image not fully visible, inviting the user to swipe/scroll.

12.2.4 How should we write CSS at scale

Writing CSS at scale is a big topic in its own right.

There have been several ideas and approaches over the years, such as:
1. OOCSS
2. BEM
3. SMACSS
4. Atomic CSS
4. Our own implementation?

Rather than throwing in bits and pieces of writing CSS at scale, I've decided to write a book (albeit a short one) on the topic.

That's what the next book in this book series is about.