CHAPTER 4

An HTML and CSS Primer

Developers come to the world of web app development via many paths and are not always grounded in the basic technologies that web apps rely on. In this chapter, I provide a brief primer for HTML and introduce the Bootstrap CSS library, which I use to style the examples in this book. In Chapters 5 and 6, I introduce the basics of JavaScript and TypeScript and give you the information you need to understand the examples in the rest of the book. If you are an experienced developer, you can skip these primer chapters and jump right to Chapter 7, where I use Angular to create a more complex and realistic application.

Preparing the Example Project

For this chapter, I need only a simple example project. I started by creating a folder called `HtmlCssPrimer`, created a file called `package.json` within it, and added the content shown in Listing 4-1.

Tip You can download the example project for this chapter—and for all the other chapters in this book—from https://github.com/Apress/pro-angular-6.

Listing 4-1. The Contents of the package.json File in the HtmlCssPrimer Folder

```
{
  "dependencies": {
    "bootstrap": "4.1.1"
  },

  "devDependencies": {
    "lite-server": "2.3.0"
  },

  "scripts": {
    "start": "npm run lite",
    "lite": "lite-server"
  }
}
```


A. Freeman, *Pro Angular 6*, https://doi.org/10.1007/978-1-4842-3649-9_4

Run the following command within the HtmlCssPrimer folder to download and install the NPM packages specified in the package.json file:

```
npm install
```

Next, I created a file called index.html in the HtmlCssPrimer folder and added the content shown in Listing 4-2.

Listing 4-2. The Contents of the index.html File in the HtmlCssPrimer Folder

```
<!DOCTYPE html>
<html>
<head>
    <title>ToDo</title>
    <meta charset="utf-8" />
    <link href="node_modules/bootstrap/dist/css/bootstrap.min.css"
          rel="stylesheet" />
</head>
<body class="m-1">

    <h3 class="bg-primary text-white p-3">Adam's To Do List</h3>

    <div class="my-1">
        <input class="form-control" />
        <button class="btn btn-primary mt-1">Add</button>
    </div>

    <table class="table table-striped table-bordered">
        <thead>
            <tr>
                <th>Description</th>
                <th>Done</th>
            </tr>
        </thead>
        <tbody>
            <tr><td>Buy Flowers</td><td>No</td></tr>
            <tr><td>Get Shoes</td><td>No</td></tr>
            <tr><td>Collect Tickets</td><td>Yes</td></tr>
            <tr><td>Call Joe</td><td>No</td></tr>
        </tbody>
    </table>
</body>
</html>
```

This is the HTML content I used in Chapter 2 to mock up the appearance of the example application. Run the following command in the HtmlCssPrimer folder to start the development HTTP server:

```
npm start
```

A new browser tab or window will open and show the content in Figure 4-1.

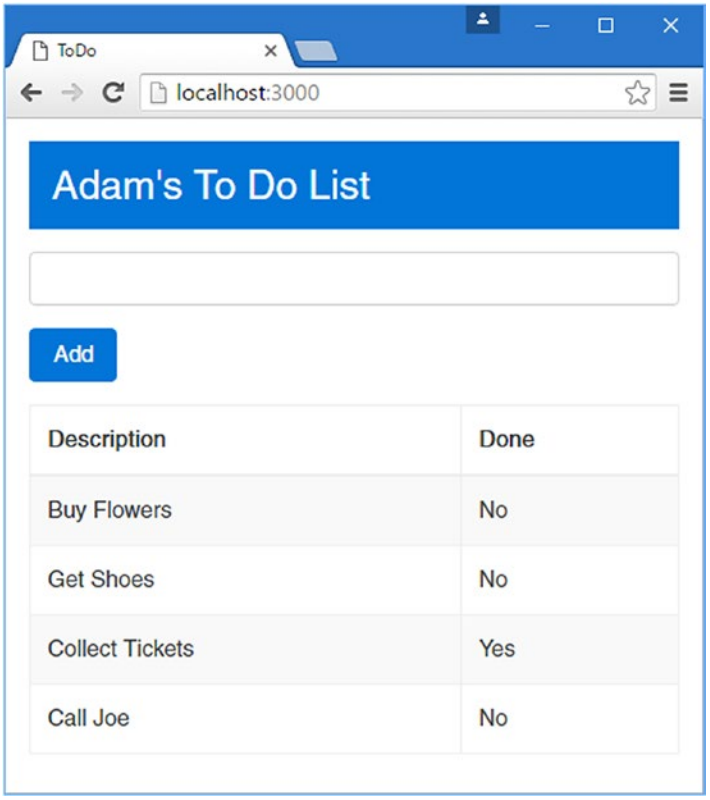

Figure 4-1. *Running the example application*

Understanding HTML

At the heart of HTML is the *element*, which tells the browser what kind of content each part of an HTML document represents. Here is an element from the example HTML document:

```
...
<td>Buy Flowers</td>
...
```

As illustrated in Figure 4-2, this element has three parts: the start tag, the end tag, and the content.

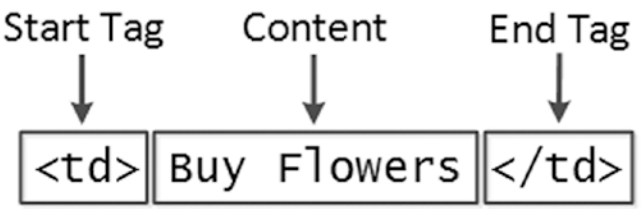

Figure 4-2. *The anatomy of a simple HTML element*

The *name* of this element (also referred to as the *tag name* or just the *tag*) is `td`, and it tells the browser that the content between the tags should be treated as a table cell. You start an element by placing the tag name in angle brackets (the < and > characters) and end an element by using the tag in a similar way, except that you also add a / character after the left-angle bracket (<). Whatever appears between the tags is the element's content, which can be text (such as `Buy Flowers` in this case) or other HTML elements.

Understanding Void Elements

The HTML specification includes elements that are not permitted to contain content. These are called *void* or *self-closing* elements, and they are written without a separate end tag, like this:

```
...
<input />
...
```

A void element is defined in a single tag, and you add a / character before the last angle bracket (the > character). The `input` element is the most commonly used void element, and its purpose is to allow the user to provide input, through a text field, radio button, or checkbox. You will see lots of examples of working with this element in later chapters.

Understanding Attributes

You can provide additional information to the browser by adding *attributes* to your elements. Here is an element with an attribute from the example document:

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

This is a `link` element, and it imports content into the document. There are two attributes, which I have emphasized so they are easier to see. Attributes are always defined as part of the start tag, and these attributes have a *name* and a *value*.

The names of the two attributes in this example are `href` and `rel`. For the `link` element, the `href` attribute specifies the content to import, and the `rel` attribute tells the browser what kind of content it is. The attributes on this `link` element tell the browser to import the `bootstrap.min.css` file and to treat it as a style sheet, which is a file that contains CSS styles.

Applying Attributes Without Values

Not all attributes are applied with a value; just adding them to an element tells the browser that you want a certain kind of behavior. Here is an example of an element with such an attribute (not from the example document; I just made up this example element):

```
...
<input class="form-control" required />
...
```

This element has two attributes. The first is `class`, which is assigned a value just like the previous example. The other attribute is just the word `required`. This is an example of an attribute that doesn't need a value.

Quoting Literal Values in Attributes

Angular relies on HTML element attributes to apply a lot of its functionality. Most of the time, the values of attributes are evaluated as JavaScript expressions, such as with this element, taken from Chapter 2:

```
...
<td [ngSwitch]="item.done">
...
```

The attribute applied to the `td` element tells Angular to read the value of a property called `done` on an object that has been assigned to a variable called `item`. There will be occasions when you need to provide a specific value rather than have Angular read a value from the data model, and this requires additional quoting to tell Angular that it is dealing with a literal value, like this:

```
...
<td [ngSwitch]="'Apples'">
...
```

The attribute value contains the string `Apples`, which is quoted in both single and double quotes. When Angular evaluates the attribute value, it will see the single quotes and process the value as a literal string.

Understanding Element Content

Elements can contain text, but they can also contain other elements, like this:

```
...
<thead>
    <tr>
        <th>Description</th>
        <th>Done</th>
    </tr>
</thead>
...
```

The elements in an HTML document form a hierarchy. The `html` element contains the `body` element, which contains content elements, each of which can contain other elements, and so on. In the listing, the `thead` element contains `tr` elements that, in turn, contain `th` elements. Arranging elements is a key concept in HTML because it imparts the significance of the outer element to those contained within.

Understanding the Document Structure

There are some key elements that define the basic structure of an HTML document: the `DOCTYPE`, `html`, `head`, and `body` elements. Here is the relationship between these elements with the rest of the content removed:

```
<!DOCTYPE html>
<html>
<head>
    ...head content...
</head>
<body>
    ...body content...
</body>
</html>
```

Each of these elements has a specific role to play in an HTML document. The DOCTYPE element tells the browser that this is an HTML document and, more specifically, that this is an *HTML5* document. Earlier versions of HTML required additional information. For example, here is the DOCTYPE element for an HTML4 document:

```
...
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
...
```

The `html` element denotes the region of the document that contains the HTML content. This element always contains the other two key structural elements: `head` and `body`. As I explained at the start of the chapter, I am not going to cover the individual HTML elements. There are too many of them, and describing HTML5 completely took me more than 1,000 pages in my HTML book. That said, Table 4-1 provides brief descriptions of the elements I used in the `index.html` file in Listing 4-2 to help you understand how elements tell the browser what kind of content they represent.

UNDERSTANDING THE DOCUMENT OBJECT MODEL

When the browser loads and processes an HTML document, it creates the *Document Object Model* (DOM). The DOM is a model in which JavaScript objects are used to represent each element in the document, and the DOM is the mechanism by which you can programmatically engage with the content of an HTML document.

You rarely work directly with the DOM in Angular, but it is important to understand that the browser maintains a live model of the HTML document represented by JavaScript objects. When Angular modifies these objects, the browser updates the content it displays to reflect the modifications. This is one of the key foundations of web applications. If we were not able to modify the DOM, we would not be able to create client-side web apps.

Table 4-1. *HTML Elements Used in the Example Document*

Element	Description
`DOCTYPE`	Indicates the type of content in the document
`body`	Denotes the region of the document that contains content elements
`button`	Denotes a button; often used to submit a form to the server
`div`	A generic element; often used to add structure to a document for presentation purposes
`h3`	Denotes a header
`head`	Denotes the region of the document that contains metadata
`html`	Denotes the region of the document that contains HTML (which is usually the entire document)
`input`	Denotes a field used to gather a single data item from the user
`link`	Imports content into the HTML document
`meta`	Provides descriptive data about the document, such as the character encoding
`table`	Denotes a table, used to organize content into rows and columns
`tbody`	Denotes the body of the table (as opposed to the header or footer)
`td`	Denotes a content cell in a table row
`th`	Denotes a header cell in a table row
`thead`	Denotes the header of a table
`title`	Denotes the title of the document; used by the browser to set the title of the window or tab
`tr`	Denotes a row in a table

Understanding Bootstrap

HTML elements tell the browser what kind of content they represent, but they don't provide any information about how that content should be displayed. The information about how to display elements is provided using *Cascading Style Sheets* (CSS). CSS consists of a comprehensive set of *properties* that can be used to configure every aspect of an element's appearance and a set of *selectors* that allow those properties to be applied.

One of the main problems with CSS is that some browsers interpret properties slightly differently, which can lead to variations in the way that HTML content is displayed on different devices. It can be difficult to track down and correct these problems, and CSS frameworks have emerged to help web app developers style their HTML content in a simple and consistent way.

The most widely used framework is Bootstrap, which consists of a set of CSS classes that can be applied to elements to style them consistently and JavaScript code that performs additional enhancement. I use the Bootstrap CSS styles in this book because they let me style my examples without having to define custom styles in each chapter. I don't use the Bootstrap JavaScript features at all in this book.

I don't want to get into too much detail about Bootstrap because it isn't the topic of this book, but I do want to give you enough information so you can tell which parts of an example are Angular features and which are Bootstrap styling. See `http://getbootstrap.com` for full details of the features that Bootstrap provides.

Applying Basic Bootstrap Classes

Bootstrap styles are applied via the `class` attribute, which is used to group together related elements. The `class` attribute isn't just used to apply CSS styles, but it is the most common use, and it underpins the way that Bootstrap and similar frameworks operate. Here is an HTML element with a `class` attribute, taken from the `index.html` file:

```
...
<button class="btn btn-primary mt-1">Add</button>
...
```

The `class` attribute assigns the `button` element to three classes, whose names are separated by spaces: `btn`, `btn-primary`, and `mt-1`. These classes correspond to styles defined by Bootstrap, as described in Table 4-2.

Table 4-2. *The Three Button Element Classes*

Name	Description
`btn`	This class applies the basic styling for a button. It can be applied to `button` or `a` elements to provide a consistent appearance.
`btn-primary`	This class applies a style context to provide a visual cue about the purpose of the button. See the "Using Contextual Classes" section.
`mt-1`	This class adds a gap between the top of the element and the content that surrounds it. See the "Using Margin and Padding" section.

Using Contextual Classes

One of the main advantages of using a CSS framework like Bootstrap is to simplify the process of creating a consistent theme throughout an application. Bootstrap defines a set of *style contexts* that are used to style related elements consistently. These contexts, which are described in Table 4-3, are used in the names of the classes that apply Bootstrap styles to elements.

Table 4-3. *The Bootstrap Style Contexts*

Name	Description
`primary`	This context is used to indicate the main action or area of content.
`secondary`	This context is used to indicate the supporting areas of content.
`success`	This context is used to indicate a successful outcome.
`info`	This context is used to present additional information.
`warning`	This context is used to present warnings.
`danger`	This context is used to present serious warnings.
`muted`	This context is used to de-emphasize content.
`dark`	This context is used to increase contrast by using a dark color.
`white`	This context is used to increase contrast by using white.

Bootstrap provides classes that allow the style contexts to be applied to different types of elements. Here is the `primary` context applied to the `h3` element, taken from the `index.html` file created at the start of the chapter:

```
...
<h3 class="bg-primary text-white p-3">Adam's To Do List</h3>
...
```

One of the classes that the element has been assigned to is `bg-primary`, which styles the background color of an element using the style context's color. Here is the same style context applied to a `button` element:

```
...
<button class="btn btn-primary mt-1">Add</button>
...
```

The `btn-primary` class styles a button or anchor element using the style context's colors. Using the same context to style different elements will ensure their appearance is consistent and complementary, as shown in Figure 4-3, which highlights the elements to which the style context has been applied.

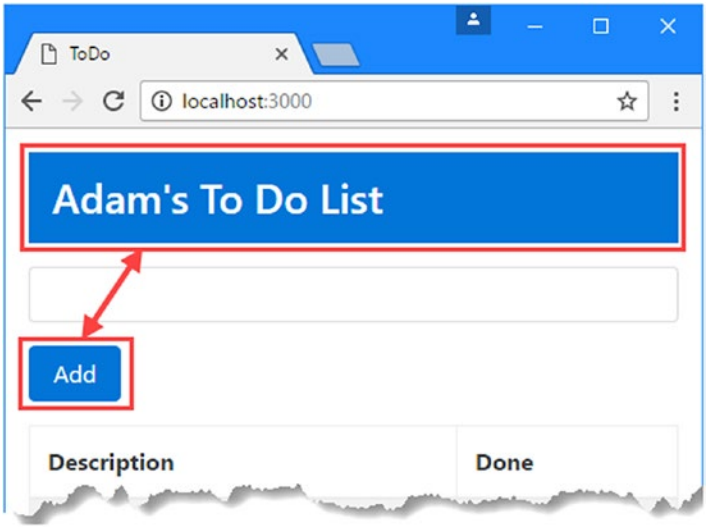

Figure 4-3. *Using style contexts for consistency*

Using Margin and Padding

Bootstrap includes a set of utility classes that are used to add padding (space between an element's inner edge and its content) and margin (space between an element's edge and the surrounding elements). The benefit of using these classes is that they apply a consistent amount of spacing throughout the application.

The names of these classes follow a well-defined pattern. Here is the `body` element from the `index.html` file created at the start of the chapter, to which margin has been applied:

```
...
<body class="m-1">
...
```

The classes that apply margin and padding to elements follow a well-defined naming schema: first, the letter `m` (for margin) or `p` (for padding), then a hyphen, and then a number indicating how much space should be applied (`0` for no spacing, or `1`, `2`, or `3` for increasing amounts). You can also add a letter to apply spacing only to specific sides, so `t` for top, `b` for bottom, `l` for left, `r` for right, `x` for left and right, and `y` for top and bottom).

To help put this scheme in context, Table 4-4 lists the combinations used in the `index.html` file.

Table 4-4. *Sample Bootstrap Margin and Padding Classes*

Name	Description
`p-1`	This class applies padding to all edges of an element.
`m-1`	This class applies margin to all edges of an element.
`mt-1`	This class applies margin to the top edge of an element.
`mb-1`	This class applies margin to the bottom edge of an element.

Changing Element Sizes

You can change the way that some elements are styled by using a size modification class. These are specified by combining a basic class name, a hyphen, and `lg` or `sm`. In Listing 4-3, I have added `button` elements to the `index.html` file, using the size modification classes that Bootstrap provides for buttons.

Listing 4-3. Using Button Size Modification Classes in the index.html File in the HtmlCssPrimer Folder

```
<!DOCTYPE html>
<html>
<head>
    <title>ToDo</title>
    <meta charset="utf-8" />
    <link href="node_modules/bootstrap/dist/css/bootstrap.min.css"
          rel="stylesheet" />
</head>
<body class="m-1">

    <h3 class="bg-primary text-white p-3">Adam's To Do List</h3>

    <div class="my-1">
        <input class="form-control" />
        <button class="btn btn-lg btn-primary mt-1">Add</button>
        <button class="btn btn-primary mt-1">Add</button>
        <button class="btn btn-sm btn-primary mt-1">Add</button>
    </div>

    <table class="table table-striped table-bordered">
        <thead>
            <tr>
                <th>Description</th>
                <th>Done</th>
            </tr>
        </thead>
```

```
        <tbody>
            <tr><td>Buy Flowers</td><td>No</td></tr>
            <tr><td>Get Shoes</td><td>No</td></tr>
            <tr><td>Collect Tickets</td><td>Yes</td></tr>
            <tr><td>Call Joe</td><td>No</td></tr>
        </tbody>
    </table>
</body>
</html>
```

The `btn-lg` class creates a large button, and the `btn-sm` class creates a small button. Omitting a size class uses the default size for the element. Notice that I am able to combine a context class and a size class. Bootstrap class modifications work together to give you complete control over how elements are styled, creating the effect shown in Figure 4-4.

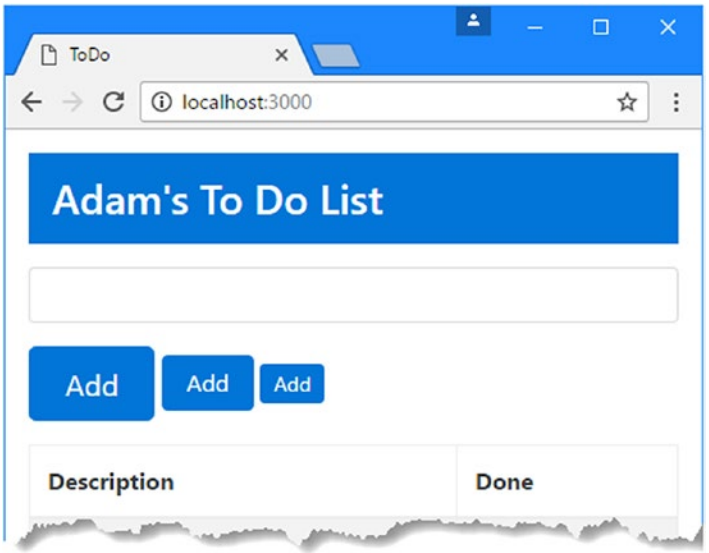

Figure 4-4. *Changing element size*

Using Bootstrap to Style Tables

Bootstrap includes support for styling `table` elements and their contents, which is a feature I use throughout this book. Table 4-5 lists the key Bootstrap classes for working with tables.

Table 4-5. *The Bootstrap CSS Classes for Tables*

Name	Description
`table`	Applies general styling to a `table` element and its rows
`table-striped`	Applies alternate-row striping to the rows in the `table` body
`table-bordered`	Applies borders to all rows and columns
`table-hover`	Displays a different style when the mouse hovers over a row in the table
`table-sm`	Reduces the spacing in the table to create a more compact layout

All these classes are applied directly to the `table` element, as shown in Listing 4-4, which highlights the Bootstrap classes applied to the table in the `index.html` file.

Listing 4-4. Using Bootstrap to Style Tables

```
...
<table class="table table-striped table-bordered">
    <thead>
        <tr>
            <th>Description</th>
            <th>Done</th>
        </tr>
    </thead>
    <tbody>
        <tr><td>Buy Flowers</td><td>No</td></tr>
        <tr><td>Get Shoes</td><td>No</td></tr>
        <tr><td>Collect Tickets</td><td>Yes</td></tr>
        <tr><td>Call Joe</td><td>No</td></tr>
    </tbody>
</table>
...
```

■ **Tip** Notice that I have used the `thead` element when defining the tables in Listing 4-4. Browsers will automatically add any `tr` elements that are direct descendants of `table` elements to a `tbody` element if one has not been used. You will get odd results if you rely on this behavior when working with Bootstrap because most of the CSS classes that are applied to the `table` element cause styles to be added to the descendants of the `tbody` element.

Using Bootstrap to Create Forms

Bootstrap includes styling for form elements, allowing them to be styled consistently with other elements in the application. In Listing 4-5, I have expanded the form elements in the `index.html` file and temporarily removed the table.

Listing 4-5. Defining Additional Form Elements in the index.html File in the HtmlCssPrimer Folder

```
<!DOCTYPE html>
<html>
<head>
    <title>ToDo</title>
    <meta charset="utf-8" />
    <link href="node_modules/bootstrap/dist/css/bootstrap.min.css"
          rel="stylesheet" />
</head>
<body class="m-2">
    <h3 class="bg-primary text-white p-3">Adam's To Do List</h3>
    <form>
        <div class="form-group">
```

```
            <label>Task</label>
            <input class="form-control" />
        </div>
        <div class="form-group">
            <label>Location</label>
            <input class="form-control" />
        </div>
        <div class="form-group">
            <input type="checkbox" />
            <label>Done</label>
        </div>
        <button class="btn btn-primary">Add</button>
    </form>
</body>
</html>
```

The basic styling for forms is achieved by applying the `form-group` class to a `div` element that contains a `label` and an `input` element, where the input element is assigned to the `form-control` class. Bootstrap styles the elements so that the `label` is shown above the `input` element and the `input` element occupies 100 percent of the available horizontal space, as shown in Figure 4-5.

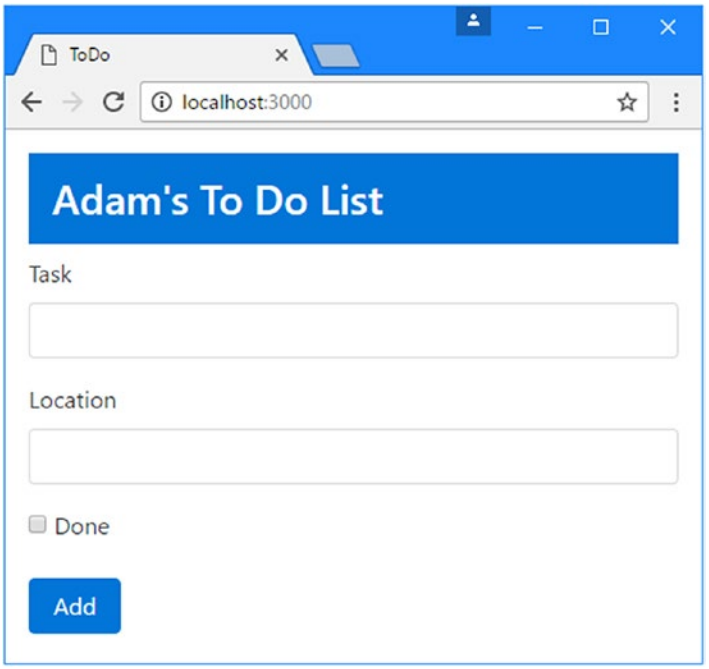

Figure 4-5. *Styling form elements*

Using Bootstrap to Create Grids

Bootstrap provides style classes that can be used to create different kinds of grid layout, ranging from one to twelve columns and with support for responsive layouts, where the layout of the grid changes based on the width of the screen. Listing 4-6 replaces the content of the example HTML file to demonstrate the grid feature.

Listing 4-6. Using a Bootstrap Grid in the index.html File in the HtmlCssPrimer Folder

```
<!DOCTYPE html>
<html>
<head>
    <title>ToDo</title>
    <meta charset="utf-8" />
    <link href="node_modules/bootstrap/dist/css/bootstrap.min.css"
          rel="stylesheet" />
    <style>
        .row > div {
                    border: 1px solid lightgrey; padding: 10px;
                    background-color: aliceblue; margin: 5px 0;
                    }
    </style>
</head>
<body class="m-2">
    <h3>Grid Layout</h3>
    <div class="container">
        <div class="row">
            <div class="col-1">1</div>
            <div class="col-1">1</div>
            <div class="col-2">2</div>
            <div class="col-2">2</div>
            <div class="col-6">6</div>
        </div>

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

        <div class="row">
            <div class="col-6">6</div>
            <div class="col-6">6</div>
        </div>

        <div class="row">
            <div class="col-11">11</div>
            <div class="col-1">1</div>
        </div>

        <div class="row">
            <div class="col-12">12</div>
        </div>
    </div>
</body>
</html>
```

The Bootstrap grid layout system is simple to use. A top-level `div` element is assigned to the `container` class (or the `container-fluid` class if you want it to span the available space). You specify a column by applying the `row` class to a `div` element, which has the effect of setting up the grid layout for the content that the `div` element contains.

Each row defines 12 columns, and you specify how many columns each child element will occupy by assigning a class whose name is `col-` followed by the number of columns. For example, the class `col-1` specifies that an element occupies one column, `col-2` specifies two columns, and so on, right through to `col-12`, which specifies that an element fills the entire row. In the listing, I have created a series of `div` elements with the `row` class, each of which contains further `div` elements to which I have applied `col-*` classes. You can see the effect in the browser in Figure 4-6.

■ **Tip** Bootstrap doesn't apply any styling to the elements within a row, which I why I have used a `style` element to create a custom CSS style that sets a background color, sets up some spacing between rows, and adds a border.

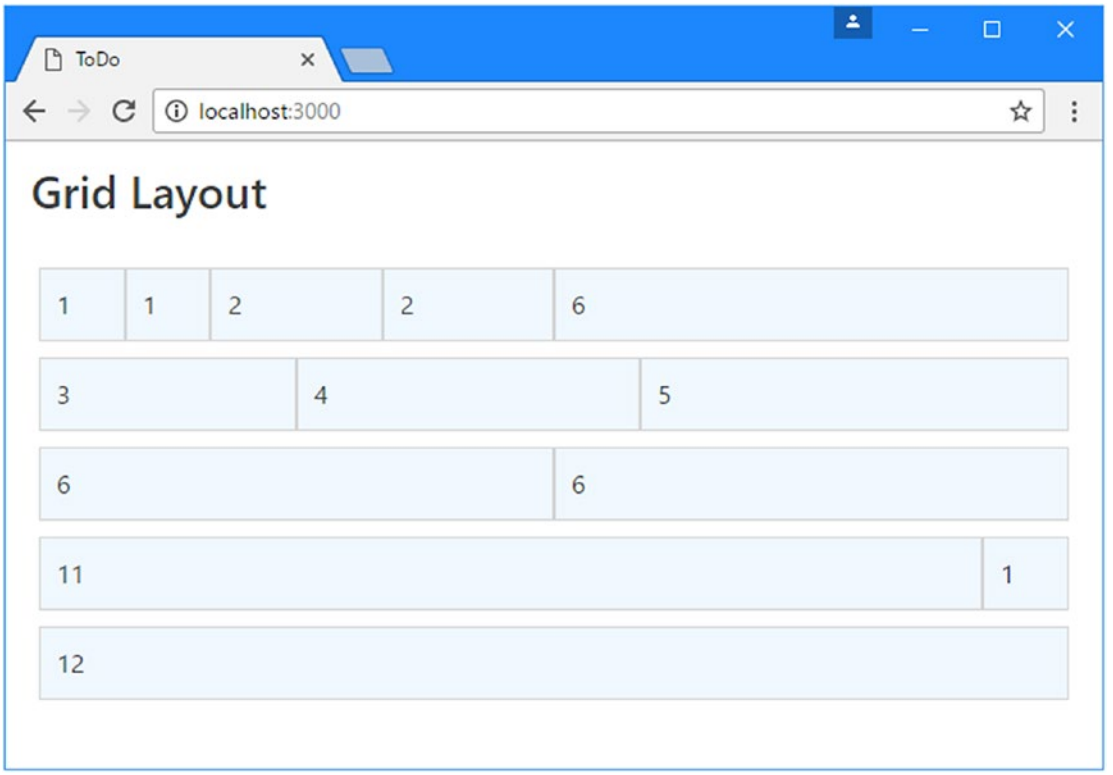

Figure 4-6. *Creating a Bootstrap grid layout*

Creating Responsive Grids

Responsive grids adapt their layout based on the size of the browser window. The main use for responsive grids is to allow mobile devices and desktops to display the same content, taking advantage of whatever screen space is available. To create a responsive grid, replace the `col-*` class on individual cells with one of the classes shown in Table 4-6.

Table 4-6. *The Bootstrap CSS Classes for Responsive Grids*

Bootstrap Class	Description
`col-sm-*`	Grid cells are displayed horizontally when the screen width is greater than 576 pixels.
`col-md-*`	Grid cells are displayed horizontally when the screen width is greater than 768 pixels.
`col-lg-*`	Grid cells are displayed horizontally when the screen width is greater than 992 pixels.
`col-xl-*`	Grid cells are displayed horizontally when the screen width is greater than 1200 pixels.

When the width of the screen is less than the class supports, the cells in the grid row are stacked vertically rather than horizontally. Listing 4-7 demonstrates a responsive grid in the `index.html` file.

Listing 4-7. Creating a Responsive Grid in the index.html File in the HtmlCssPrimer Folder

```
<!DOCTYPE html>
<html>
<head>
    <title>ToDo</title>
    <meta charset="utf-8" />
    <link href="node_modules/bootstrap/dist/css/bootstrap.min.css"
          rel="stylesheet" />
    <style>
        #gridContainer {padding: 20px;}
        .row > div {
                     border: 1px solid lightgrey; padding: 10px;
                     background-color: aliceblue; margin: 5px 0;
                    }
    </style>

</head>
<body class="m-1">

    <h3>Grid Layout</h3>
    <div class="container">
            <div class="row">
                <div class="col-sm-3">3</div>
                <div class="col-sm-4">4</div>
                <div class="col-sm-5">5</div>
            </div>

            <div class="row">
                <div class="col-sm-6">6</div>
                <div class="col-sm-6">6</div>
            </div>
```

```
            <div class="row">
                <div class="col-sm-11">11</div>
                <div class="col-sm-1">1</div>
            </div>
    </div>
</body>
</html>
```

I removed some grid rows from the previous example and replaced the `col-*` classes with `col-sm-*`. The effect is that the cells in the row will be stacked horizontally when the browser window is greater than 576 pixels wide and stacked horizontally when it is smaller, as shown in Figure 4-7.

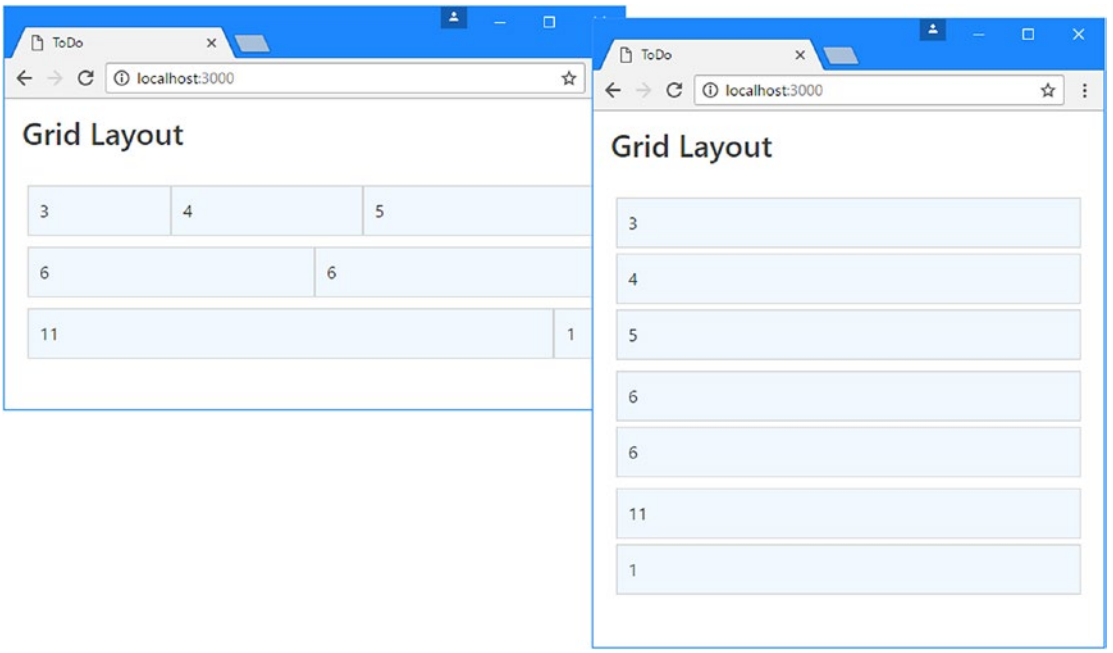

Figure 4-7. *Creating a responsive grid layout*

Creating a Simplified Grid Layout

For most of the examples in this book that rely on the Bootstrap grid, I use a simplified approach that displays content in a single row and requires only the number of columns to be specified, as shown in Listing 4-8.

Listing 4-8. Using a Simplified Grid Layout in the index.html File in the HtmlCssPrimer Folder

```
<!DOCTYPE html>
<html>
<head>
    <title>ToDo</title>
    <meta charset="utf-8" />
    <link href="node_modules/bootstrap/dist/css/bootstrap.min.css"
          rel="stylesheet" />
</head>
```

```
<body class="m-1">
    <h3 class="bg-primary text-white p-3">Adam's To Do List</h3>
    <div class="container-fluid">
        <div class="row">
            <div class="col-4">
                <form>
                    <div class="form-group">
                        <label>Task</label>
                        <input class="form-control" />
                    </div>
                    <div class="form-group">
                        <label>Location</label>
                        <input class="form-control" />
                    </div>
                    <div class="form-group">
                        <input type="checkbox" />
                        <label>Done</label>
                    </div>
                    <button class="btn btn-primary">Add</button>
                </form>
            </div>
            <div class="col-8">
                <table class="table table-striped table-bordered">
                    <thead>
                        <tr>
                            <th>Description</th>
                            <th>Done</th>
                        </tr>
                    </thead>
                    <tbody>
                        <tr><td>Buy Flowers</td><td>No</td></tr>
                        <tr><td>Get Shoes</td><td>No</td></tr>
                        <tr><td>Collect Tickets</td><td>Yes</td></tr>
                        <tr><td>Call Joe</td><td>No</td></tr>
                    </tbody>
                </table>
            </div>
        </div>
    </div>
</body>
</html>
```

This listing uses the `col-4` and `col-8` classes to display two `div` elements side by side, allowing the form and the table that displays the to-do items to be displayed horizontally, as illustrated in Figure 4-8.

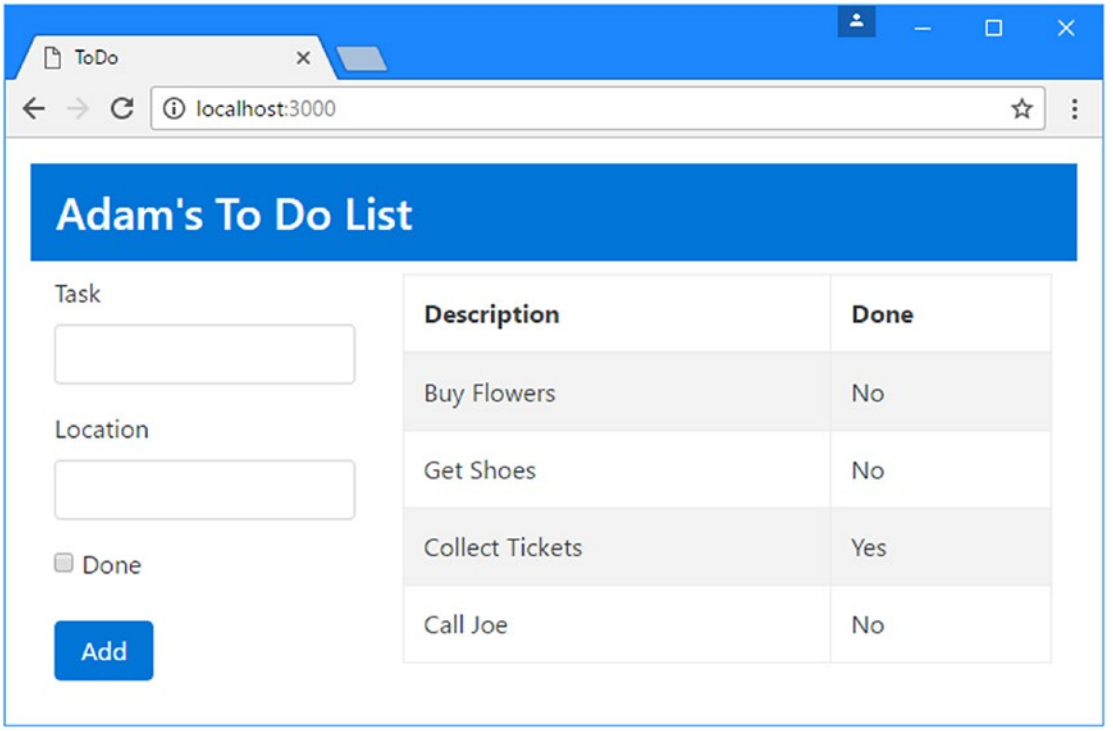

Figure 4-8. *Using a simplified grid layout*

Summary

In this chapter, I provided a brief overview of HTML and the Bootstrap CSS framework. You need to have a good grasp of HTML and CSS to be truly effective in web application development, but the best way to learn is by firsthand experience, and the descriptions and examples in this chapter will be enough to get you started and provide just enough background information for the examples ahead. In the next chapter, I continue the primer theme and introduce the basic features of JavaScript that I use in this book.