3
Enhancing the Portfolio Website with CSS3

In the previous chapter, we wrote the HTML5 code for our portfolio website. In this chapter, we will start working on CSS3. We will start with some simple visual effects that have just been introduced in CSS3, such as drop shadows, text shadows, and rounded corners.

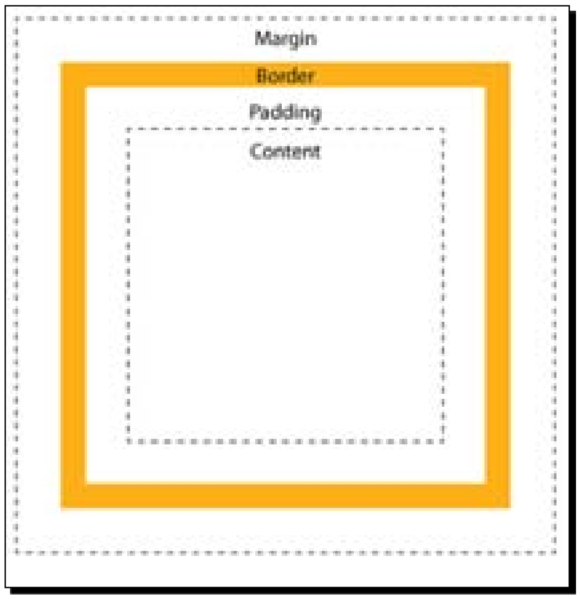
Then, we will also create a more advanced effect called the thumbnail hover effect, which was only achieved via JavaScript prior to the introduction of CSS3.

To sum up, here are the tasks we are going to perform in this chapter:

- Style the website, beginning from the header and navigation, then work through the content area, and finish with the footer with some new CSS3 properties
- Create a portfolio filter function with CSS3
- Create a thumbnail hover effect with CSS3 Transforms and Transitions
- Adjust website appearance for specific viewport sizes with CSS3 media queries

CSS box model

An HTML element that is categorized as a block-level element is essentially a box; it consists of the content, margin, padding, and borders that are specified through CSS, as illustrated in the following screenshot:



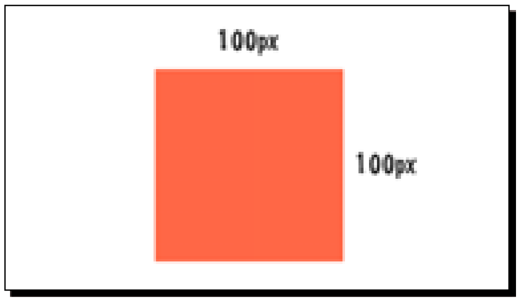
You can further see the difference between block and inline elements as well as the list of the elements from the following references:

- *Block-level elements* (`https://developer.mozilla.org/en-US/docs/HTML/Block-level_elements`)
- *Inline elements* (`https://developer.mozilla.org/en-US/docs/HTML/Inline_elements`)

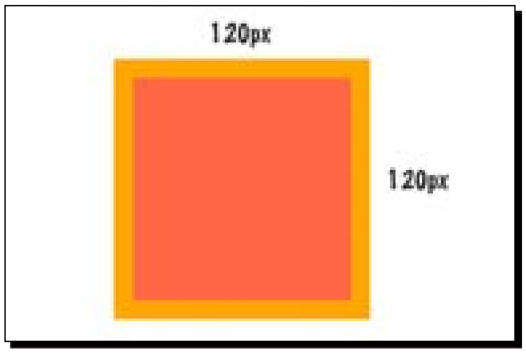
Prior to CSS3, we have been facing a constraint when specifying a box, for example, when we give an element a width and height of 100 pixels, as follows:

```
div {
  width: 100px;
  height: 100px;
}
```

The browser will simply translate it as a 100 pixel, square box.



However, this is only true if the padding, margin, or border has not been added. Since the box has four sides, a padding of 10 pixels (`padding: 10px;`) will actually add 20 pixels to the width and height—10 pixels for each side.



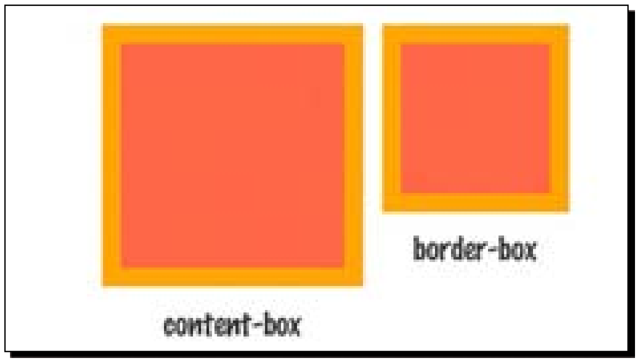
While it takes up space on the page, the element's margin is space reserved outside the element rather than part of the element itself; thus, if we give an element a background color, the margin area will not take on that color.

An introduction to the CSS3 box-sizing property

CSS3 offers additional options for controlling this box model with its `box-sizing` property.

Value	Description
`content-box`	This is the default value of the box model. This value specifies the padding and the border box's thickness outside the specified width and height of the content, as we have demonstrated in the preceding section.
`border-box`	This value does the opposite; it specifies the padding and the border box inside the specified width and height of the content.

Now let's get back to our example. This time we will set the `box-sizing` model to `border-box`. So the width and the height will remain 100 px, regardless of the padding and border length. The following illustration shows a comparison between the different outputs of the two values:



Time for action – specifying box-sizing

So, let's open up `style.css` and add the following rule at the beginning of the code:

```
* {
  -webkit-box-sizing: border-box;
  -moz-box-sizing: border-box;
  box-sizing: border-box;
}
```

What just happened?

We've used the CSS universal selector, the asterisk (`*`), to apply `border-box` sizing to all block-level elements so that we can easily set their final width and height.

> This simple tip was first promoted by Paul Irish, a lead developer of Modernizr and HTML5 Boilerplate. You can read his post (`http://paulirish.com/2012/box-sizing-border-box-ftw/`) for more details on this method.

CSS units of measurement

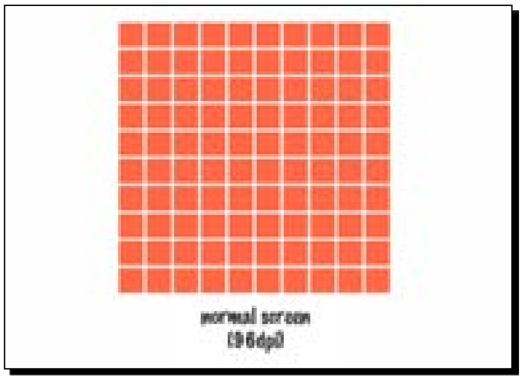
There are a number of units of measurement in CSS specification. In our website, we will mostly use the px, em, and percent units.

The pixel unit

px is an absolute length unit and probably the most popular unit used in web documents. px gives control of the exact length of an element. With reference to the documentation available at `http://www.w3.org/TR/css3-values/#reference-pixel`, a pixel in CSS refers to:

The visual angle of one pixel on a device with a pixel density of 96 DPI.

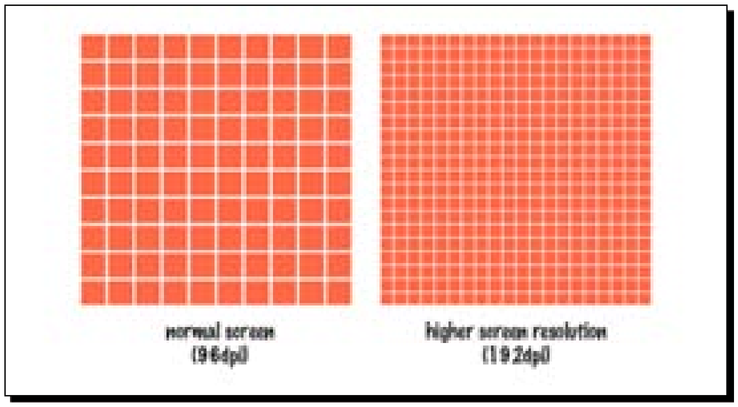
According to this explanation, 1 CSS pixel in a 96 DPI screen is equal to 1 device pixel. Thus, 10 px in CSS is simply equal to 10 px on the screen.



So for our project, we will use the px unit to measure box sizing.

The pixel unit in higher DPI screens

Today, with the increasing popularity of higher screen resolution, the preceding example is no longer relevant. The following screenshot is an example of a high-definition screen with a resolution of 192 DPI. An element that has a width and height of 10 px will actually take 20 device pixels.



The element will still have the same physical size on the screen,only now there will be more device pixels embedded in the 1 CSS pixel.

> There are a lot of discussions regarding a pixel and its relation to screen resolution.
>
> - Reda Lemeden, on his post, has covered the challenges and constraints on designing multiple device densities (`http://coding.smashingmagazine.com/2012/08/20/towards-retina-web/`)
> - Scott Kellum, at A List Apart, has covered the pixel unit and its relevance towards multiple devices with different screen sizes and resolutions (`http://www.alistapart.com/articles/a-pixel-identity-crisis/`)

The em unit

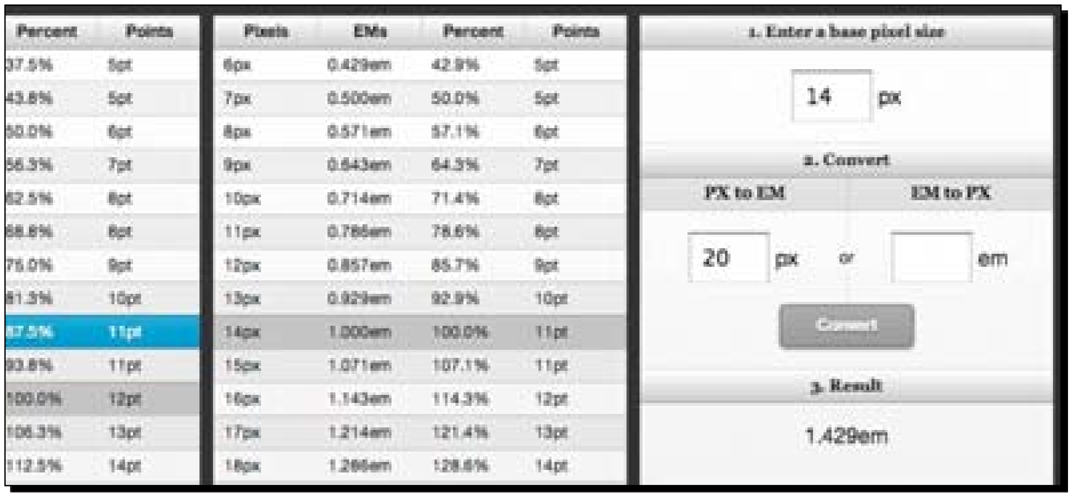
`em` is a relative unit. It actually refers to the size of the capital alphabet "M" of the specified font. In CSS, `1em` technically refers to the device- or document-based font size. If there is a parent element with a specified font size in the `em` unit, the child elements nested within it will take the parent element's font size as the reference instead.

In our project, we will use `em` to specify the font size recommended by W3C (`http://www.w3.org/Style/Examples/007/units.en.html`).

Converting px to em

The default `body` font size in Skeleton is specified in `base.css` along with the default font family for `14px`, so this would be the base font size the `em` unit would refer to.

So, let's say we need to find the `em` number of `20px` with `14px` as the base font size. There is a tool to convert `px` to `em` (or vice versa) easily, called PXtoEM.com (`http://pxtoem.com/`). The following screenshot shows how we do the calculation with this tool:



Percent	Points
37.5%	5pt
43.8%	5pt
50.0%	6pt
56.3%	7pt
62.5%	8pt
68.8%	8pt
75.0%	9pt
81.3%	10pt
87.5%	11pt
93.8%	11pt
100.0%	12pt
106.3%	13pt
112.5%	14pt

Pixels	EMs	Percent	Points
6px	0.429em	42.9%	5pt
7px	0.500em	50.0%	5pt
8px	0.571em	57.1%	6pt
9px	0.643em	64.3%	7pt
10px	0.714em	71.4%	8pt
11px	0.786em	78.6%	8pt
12px	0.857em	85.7%	9pt
13px	0.929em	92.9%	10pt
14px	1.000em	100.0%	11pt
15px	1.071em	107.1%	11pt
16px	1.143em	114.3%	12pt
17px	1.214em	121.4%	13pt
18px	1.286em	128.6%	14pt

1. Enter a base pixel size

14 px

2. Convert

PX to EM: 20 px or EM to PX: em

Convert

3. Result

1.429em

As you can see from the preceding screenshot, with `14px` as the base size, `20px` is equal to `1.429em`.

Calculating the em unit manually

Alternatively, we can convert `px` to `em` (and vice versa) using the formulas listed in the following table:

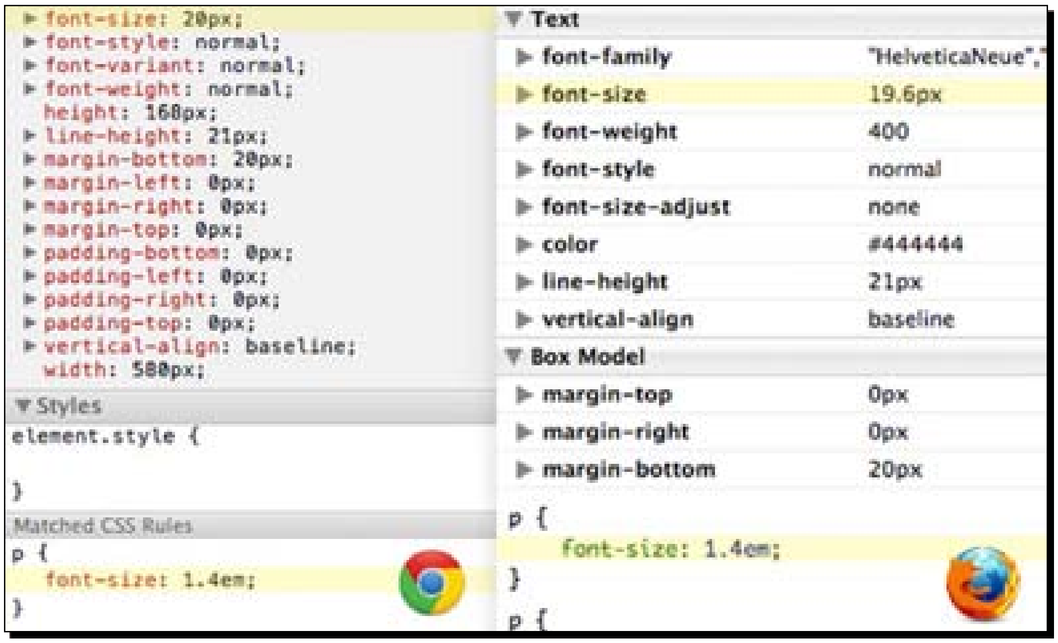
Unit conversion	Formula	Example
`px` to `em`	Size (px) / font size base (px)	20(px) / 14(px) = 1.429em (rounded)
`em` to `px`	Size (em) * font size base (px)	1.429(em) * 14(px) = 20px

Browser quirk for the em unit

Browsers translate the `em` unit a bit differently in some cases. In the example in the preceding section, `1.429em` with `14px` as the base size will turn into exactly `20px` across all browsers (Google Chrome, Opera, Safari, and Firefox).

However, as we round up this number to be `1.4em` by removing the last two numbers, the result will be slightly different. In Firefox and Opera, the resulting number will be `19.6px`, while in Webkit browsers (Google Chrome and Safari), this number is rounded up to `20px`.

You can inspect how the browsers translate `em` to px through **Developer Tool** under the the **Computed** panel.



```
font-size: 20px;
font-style: normal;
font-variant: normal;
font-weight: normal;
height: 168px;
line-height: 21px;
margin-bottom: 20px;
margin-left: 0px;
margin-right: 0px;
margin-top: 0px;
padding-bottom: 0px;
padding-left: 0px;
padding-right: 0px;
padding-top: 0px;
vertical-align: baseline;
width: 580px;
Styles
element.style {

}
Matched CSS Rules
p {
  font-size: 1.4em;
}
```

```
Text
  font-family        "HelveticaNeue","
  font-size          19.6px
  font-weight        400
  font-style         normal
  font-size-adjust   none
  color              #444444
  line-height        21px
  vertical-align     baseline
Box Model
  margin-top         0px
  margin-right       0px
  margin-bottom      20px
p {
  font-size: 1.4em;
}
p {
```

The percent unit

Percent is a relative unit and works similarly to `em`; while `em` refers to font size, percent refers to the parent length regardless of the unit being used. For example, if the parent element has a height of `100px`, `100%` of its child element will be equal to `100px`, `50%` will be equal to `50px`, and so on.

In our project, we will use the percent unit to measure box size, particularly when it is displayed in a smaller viewport size.

Setting font families

Skeleton sets Helvetica Neue and Helvetica as the default font in the body document; if these fonts are not available, it will apply Arial or the default sans-serif fonts to the user's machine.

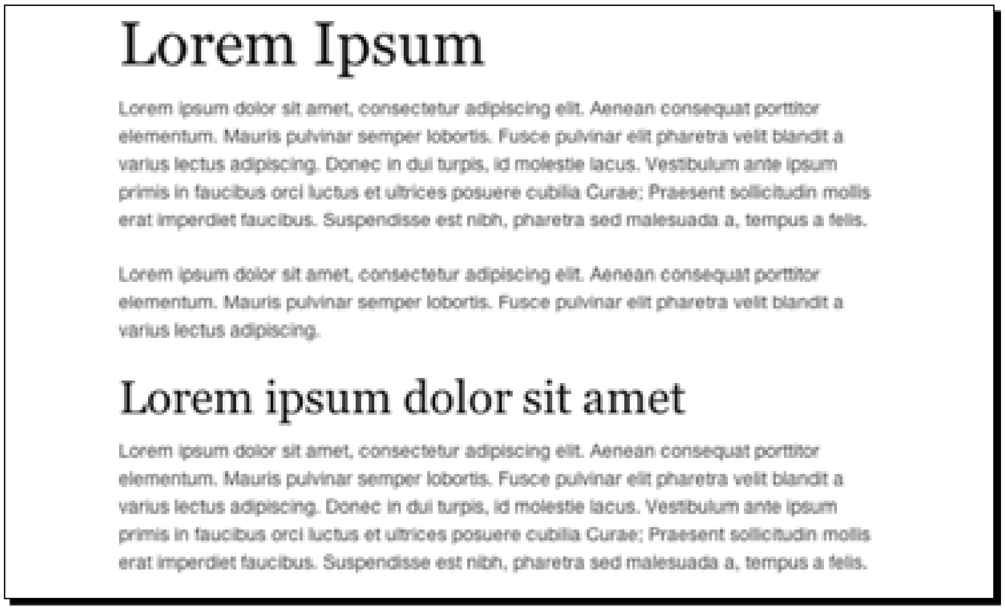
We can find these fonts defined in the `base.css` stylesheet, as follows:

```
body {
  background: #fff; font: 14px/21px "HelveticaNeue",
  "Helvetica Neue", Helvetica, Arial, sans-serif;
  color: #444;
  -webkit-font-smoothing: antialiased;
  -webkit-text-size-adjust: 100%;
}
```

The sans-serif fonts Georgia and Times New Roman are set for the Headings (`h1`, `h2`, `h3`, and so on).

```
h1, h2, h3, h4, h5, h6 {
  color: #181818; font-family: "Georgia", "Times New Roman", serif;
  font-weight: normal;
}
```

These fonts are well fitted to display paragraphs.



Lorem Ipsum

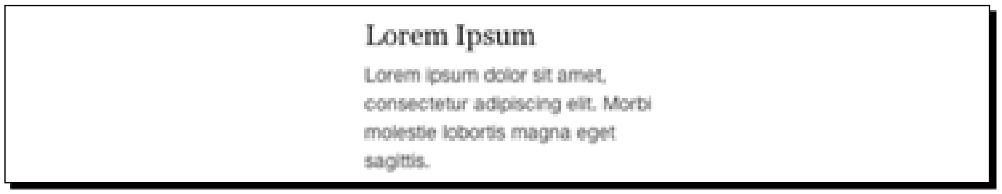
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean consequat porttitor elementum. Mauris pulvinar semper lobortis. Fusce pulvinar elit pharetra velit blandit a varius lectus adipiscing. Donec in dui turpis, id molestie lacus. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Praesent sollicitudin mollis erat imperdiet faucibus. Suspendisse est nibh, pharetra sed malesuada a, tempus a felis.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean consequat porttitor elementum. Mauris pulvinar semper lobortis. Fusce pulvinar elit pharetra velit blandit a varius lectus adipiscing.

Lorem ipsum dolor sit amet

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean consequat porttitor elementum. Mauris pulvinar semper lobortis. Fusce pulvinar elit pharetra velit blandit a varius lectus adipiscing. Donec in dui turpis, id molestie lacus. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Praesent sollicitudin mollis erat imperdiet faucibus. Suspendisse est nibh, pharetra sed malesuada a, tempus a felis.

However, they don't quite work well in our example, as we will only have very less text on our website.


Lorem Ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi molestie lobortis magna eget sagittis.

So, we will set the Headings fonts in the same way we set the body fonts to maintain uniformity in the website.

Time for action – setting the Headings font family

Let's open `style.css` and place the following rule in the `box-sizing` declaration that we added in the *Time for action – specifying box-sizing* section:

```
h1, h2, h3, h4, h5, h6 {
  font-family: "HelveticaNeue", "Helvetica Neue", Helvetica, Arial,
  sans-serif;
  font-weight: bold;
}
```

What just happened?

We have just set the Headings font to be the same as the body fonts and set `font-weight` to `bold`.

> There is no exact definitive formula for pairing fonts; it is an art. But there are some general tips to follow to make it work. Ian Yates has shared a few tips on this subject over at Webdesigntuts+ (`http://webdesign.tutsplus.com/articles/typography-articles/a-beginners-guide-to-pairing-fonts/`).

Header styles

Now, it is time to add styles to the web sections. The header of our website is defined with the HTML5 `<header>` element and assigned with the `header` class. We also have a `<div>` element with the `logo` class that contains the website logo.

Time for action – adding the header styles

To add the header styles, perform the following steps:

1. In our `style.css` file, add the following CSS rule. This CSS rule will set the header's background color, padding, border, box shadow (we add it with the CSS3 `box-shadow` property), and margin bottom to set the distance between the header and the lower section.

```
.header {
  padding: 22px 0;
  background-color: #3a3f43;
  margin-bottom: 14px;
  box-shadow: 0 1px 3px 0 rgba(0,0,0,0.3);
  border-bottom: 1px solid #181f25;
}
```

2. Then, add styles to the website logo's container. In the following CSS rule, add a CSS3 property `border-radius` to make the box's corners rounded:

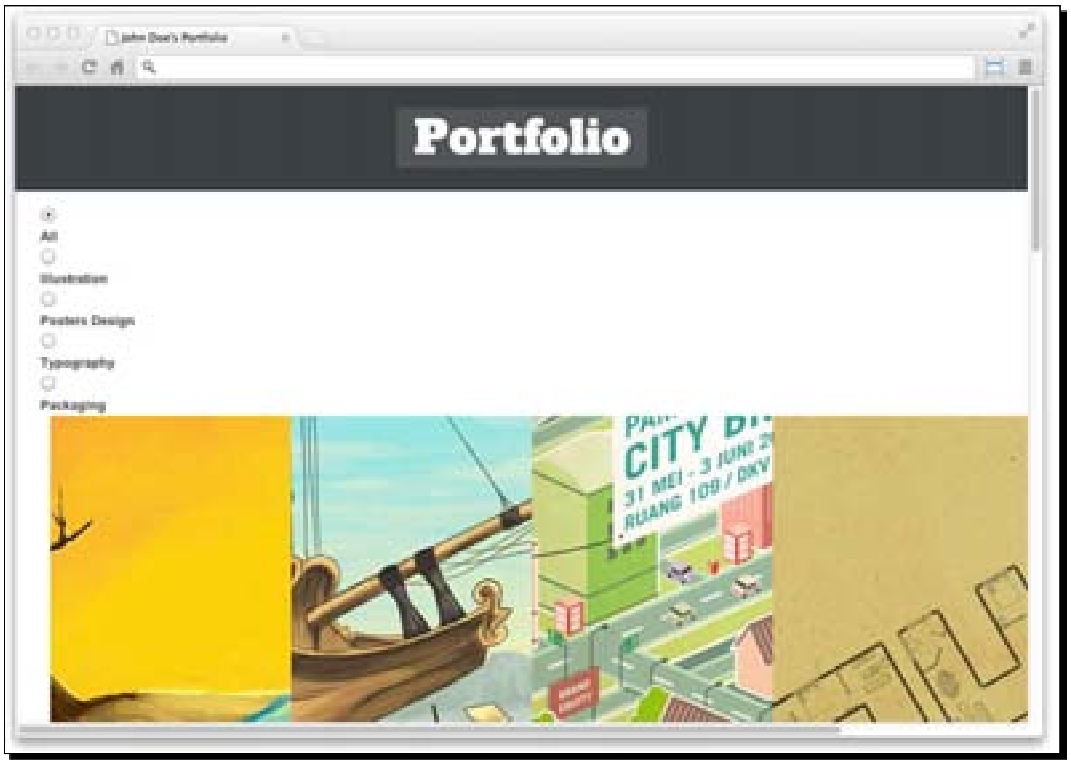
```
.logo {
  text-align: center;
  border-radius: 3px;
  background-color: #515558;
  width: 250px;
  padding: 5px 0;
  margin: 0 auto;
}
```

3. Now, add styles to the logo. Our website logo is simply text. We will assign a new font family "Alfa Slab One" to it, which we added with Google Web Font in *Chapter 2*, *Constructing a Responsive Portfolio Page with Skeleton*.

```
.logo h1 {
  color: #fff;
  font-weight: normal;
  font-family: "Alfa Slab One", Arial, sans-serif;
  margin-bottom: 0;
}
```

What just happened?

We have just added styles to the header, including the background color, box shadow, and box styles (padding, margin, and border). We also assigned a new font family from Google Web Font, `"Alfa Slab One"`, to the website's logo. The following screenshot shows how our website will look at this point:



Using CSS selectors

In our project, we will use several CSS selectors to select an element within a particular structure. These selectors include the direct child selector, adjacent sibling selector, and general sibling selector. Let's have a look at these selectors one by one.

Direct child selector

CSS allows us to select the child elements nested inside a specific element (the parent element). You're probably familiar with how we select a child element through CSS; we firstly select the parent element (with its class, ID, or element type) followed by the child element we intend to select.

```
.parent p {
  background-color: tomato;
}
```

The preceding code snippet selects every <p> element that is nested within an element with the class `parent`, without an exception.

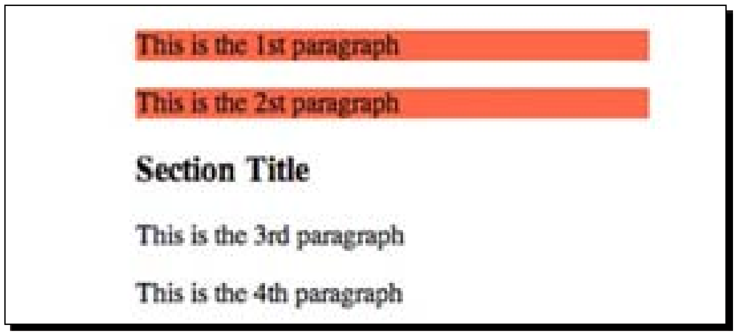
But, there may be times when we only want to select the direct child of the parent. In other words, the grandchild elements of the parent shouldn't be affected. If that is the case, we can add a > notation in between to limit the selection to only the direct child of the parent, as follows:

```
.parent > p {
  background-color: tomato;
}
```

Given the following HTML structure, the preceding CSS rule will only select the first and second paragraph.

```
<div class="parent">
  <p> This is the 1st paragraph </p>
  <p> This is the 2st paragraph </p>
  <section>
    <h3>Section Title</h3>
    <p> This is the 3rd paragraph </p>
    <p> This is the 4th paragraph </p>
  </section>
</div>
```

This gives us the following result:


This is the 1st paragraph

This is the 2st paragraph

Section Title

This is the 3rd paragraph

This is the 4th paragraph

Adjacent sibling selector

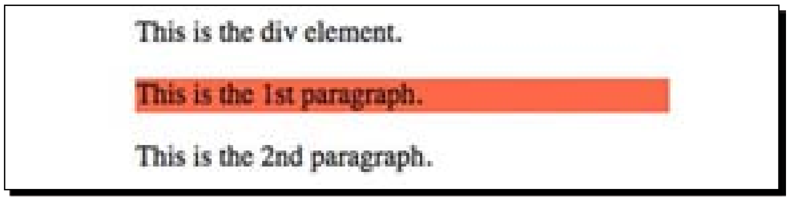
The adjacent sibling selector is defined with a plus (+) notation. It selects the element that directly follows the previous element that was specified, for example, if we have a `<div>` element that is followed by a `<p>` element, as follows:

```
<div>This is the div element.</div>
<p>This is the 1st paragraph.</p>
<p>This is the 2nd paragraph.</p>
```

We target a `<p>` element that is directly after the `<div>` element and give it a background color of tomato, as follows:

```
div + p {
  background-color: tomato;
}
```

The previous example gives us the following result:



This is the div element.

This is the 1st paragraph.

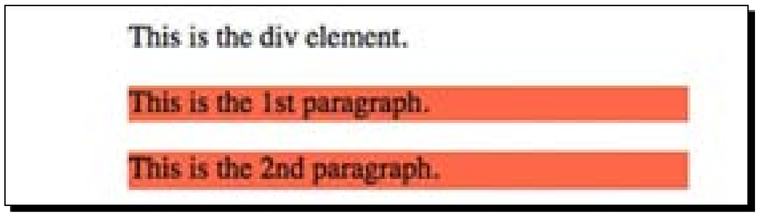
This is the 2nd paragraph.

General sibling selector

The CSS general sibling selector is a new type of selector that's just been added in CSS3. This type of selector is declared with a ~ notation, as follows:

```
div ~ p {
  background-color: tomato;
}
```

And the result is the same as in the adjacent sibling selector, but instead of targeting only the first child, the general sibling selector will target every selected element that follows the previous element. So if we have the same HTML structure as in the adjacent sibling selector, the background color will affect all paragraph elements, as shown in the following screenshot:



This is the div element.

This is the 1st paragraph.

This is the 2nd paragraph.

Using CSS3 pseudo classes

We will discuss a few CSS3 pseudo classes. A pseudo class is used to select an element within a particular expression or condition. For example, `:hover` is a pseudo class; it applies CSS rules when we point the element with a mouse cursor.

In this project, we are going to use the following pseudo classes:

- `:checked`
- `:nth-child`

Let's have a look.

The CSS3 checked pseudo class

CSS3 has introduced a new pseudo class called `:checked`. This pseudo class selects an HTML element, either the checkbox or the radio input type, that is being checked or selected. In following code snippet, we select the radio input type with an ID of the type posters when it is checked.

```
#posters:checked {
/* style rules */
}
```

This pseudo class, `:checked`, is useful for selecting the selected radio input that we are using as website navigation. Similar to a traditional menu navigation that is built with an <a> element, we use `:hover` when the mouse cursor is over the element.

The CSS3 nth-child pseudo class

CSS3 has also introduced a new pseudo class named `:nth-child`. This pseudo class allows us to select elements in their specified sequence. To select the elements, `:nth-child` needs an argument. The argument can take either numbers or keywords (odd and `even`).

For example, the following code will select the third `<li>` element and set the background color:

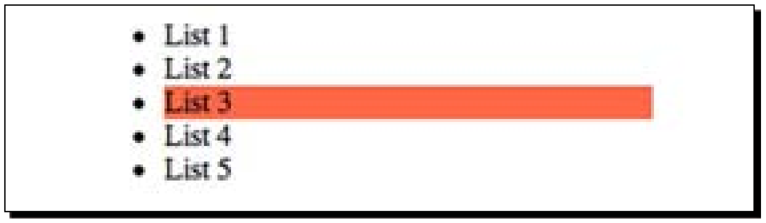
```
li:nth-child(3) {
  background-color: tomato;
}
```

Given the following HTML structure, the preceding CSS rule will add background color to the `<li>` element in the middle:

```
<ul>
  <li>List 1</li>
  <li>List 2</li>
```

```
    <li>List 3</li>
    <li>List 4</li>
    <li>List 5</li>
  </ul>
```

This is shown in the following screenshot:



Using a keyword, either odd or even, is also allowed. Intuitively, the following code will apply background color to every `<li>` element that is in an odd sequence (first, third, fifth, and so on):

```
  li:nth-child(odd) {
    background-color: tomato;
  }
```

The `:nth-child` pseudo class also accepts a formula to select elements in a more specific sequence.

```
  li:nth-child(2n+2) {
    background-color: tomato;
  }
```

n in the formula is a variable, which takes numbers starting from `0`, `1`, `2`, `3`, and so on. So, the formula 2n+2 from the preceding example will select the `<li>` element in the order second, fourth, eighth, tenth, and so on.

To know further about how `:nth-child` works, you can refer to a post by Chris Coyier at CSS Tricks (`http://css-tricks.com/how-nth-child-works/`). He has also created a handy tool to test the formula with `:nth-child` (`http://css-tricks.com/examples/nth-child-tester/`).

Portfolio thumbnail and caption styles

Once we are done with the website header, we will start adding styles and laying out the portfolio images. We have 12 image thumbnails displaying the portfolio. Each image is wrapped within an HTML5 `<figure>` element and has a caption containing the thumbnail description that is wrapped within an HTML `<figcaption>` element.

Time for action – adding thumbnail and caption styles

To add a thumbnail and caption styles, perform the following steps:

1. Open `style.css`. First of all, we will provide a little distance at the top of the portfolio container by adding a margin:

   ```
   .portfolio {
     margin-top: 20px;
   }
   ```

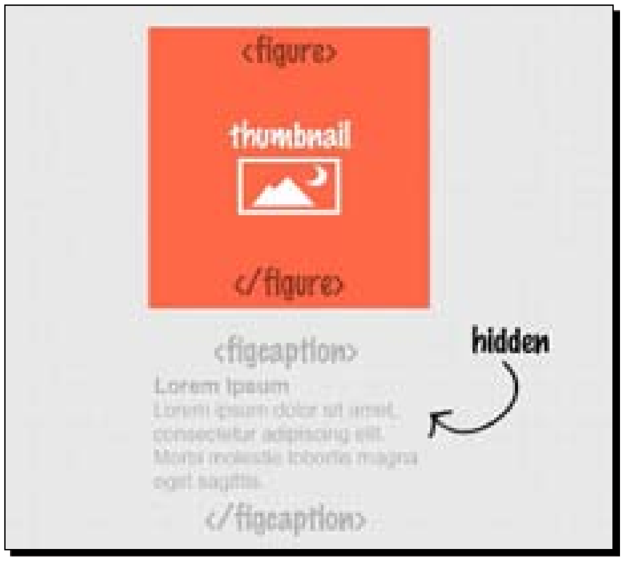
2. We'll divide the images into four columns; each column will have `width` set to `240px`, which we got from the division *960px / 4 = 240px*. In addition to this, to make this number fit into the container, we also need to remove `margin-left` and `margin-right` that have been acquired from the `.columns` class in Skeleton.

   ```
   .portfolio .four.columns {
     width: 240px;
     margin-right: 0;
     margin-left: 0;
   }
   ```

3. Then we'll set the `position` mode for the `<figure>` element to `relative`, so the child element positions, such as `<img>` and `<figcaption>`, are positioned relative to this `<figure>` element. We also set the `overflow` area of the `<figure>` element to `hidden`.

   ```
   .portfolio > figure {
     position: relative;
     overflow: hidden;
   }
   ```

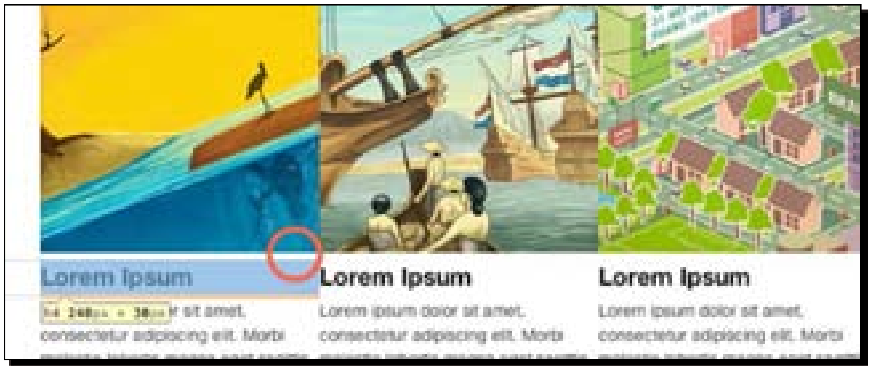
When setting `overflow` to `hidden` in the `<figure>` element, the element that flows over the `<figure>` element will be hidden. In our example, this area will be used to hide the `<figcaption>` element, as illustrated in the following screenshot:


<figure>
thumbnail
</figure>
<figcaption>
Lorem Ipsum
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi molestie lobortis magna eget sagittis.
</figcaption>
hidden

4. We will set the image's `max-width` to `100%` so the image fits inside its parent element (`figure`) regardless of how narrow it becomes.

```
.portoflio > figure img {
  max-width: 100%;
}
```

5. Furthermore, if we take a closer look at the image's thumbnail, we will find a little whitespace following the `<img>` element, which seems to be the nature of inline elements and, presumably, is also affected by its default vertical alignment (`http://www.impressivewebs.com/difference-block-inline-css/`).


Lorem Ipsum
h4 240px × 30px
Lorem Ipsum
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi
Lorem Ipsum
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi

One of the solutions to remove this whitespace is to set `display` to `block`. So let's add `display: block` to the `<img>` element, as follows:

```
.portfolio > figure img {
  max-width: 100%;
  display: block;
}
```

Alternatively, to remove the whitespace, we can also set the `vertical-align` property to `top`.

6. Next, we will add styles for the thumbnail caption. The caption is wrapped within the HTML5 `<figcaption>` element. First, we will set the caption's `position` attribute to `absolute`.

```
.portfolio figcaption {
  position: absolute;
}
```

This will affect the parent's height and it will follow the child elements that are not set for the absolute position. At this point, the caption is hidden due to the position that is now overflowing from the `<figure>` area.

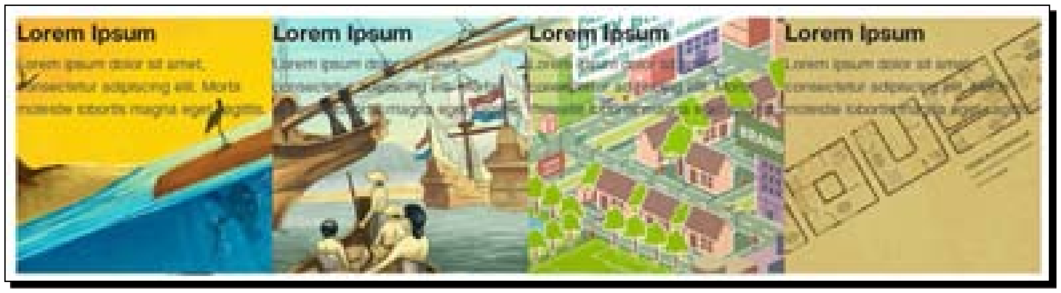
7. Then, we set the caption's height and width to `100%` so its dimensions (height and width) will always follow the parent, which in our case is the `<figure>` element.

```
.portfolio figcaption {
  position: absolute;
  width: 100%;
  height: 100%;
}
```

8. By setting the `<figcaption>` element's `position` attribute to `absolute`, we can freely reposition it to face any direction without affecting the surrounding elements. In this case, we set the caption's `left` and `top` position to `0`, as shown in the following code snippet:

```
.portfolio figcaption {
  position: absolute;
  width: 100%;
  height: 100%;

  left: 0;
  top: 0;
}
```

Since we have set the `<figure>` element's `position` attribute to `relative`, the caption's position is relative to the `<figure>` element (which is its parent), thereby resulting in the caption being on top of the image thumbnail, as shown in the following screenshot:


Lorem Ipsum
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi molestie lobortis magna eget sagittis.
Lorem Ipsum
Lorem Ipsum
Lorem Ipsum

9. We then set the caption's background color. We set the background with the RGBA (Red, Green, Blue, and Alpha) color mode. Each color channel—Red, Green, and Blue—is specified with a number ranging from `0` to `255`.

For example, setting `0` for each color channel (`rgba(0,0,0,1)`) will result in black color and is equal to `#000000` in the HEX color mode. Similarly, setting `255` for each color channel will result in white color; this is equal to `#ffffff` in the HEX color mode.

With RGBA, we can also adjust the color transparency through the Alpha channel. Values in the Alpha channel range from `0` to `1`, where `0` is equal to `0%` and `1` is equal to `100%`. So in other words, `0.5` would be equal to `50%`.

In the following rule, we set the background color to black and the transparency to `80%`:

```
.portfolio figcaption {
  position: absolute;

  width: 100%;
  height: 100%;

  left: 0;
  top: 0;

  background-color: rgba(58,63,67,.8);
}
```

10. We add padding to set the distance between the caption text and the container's edge. In the following rule, we set `padding` to `10%`:

```
.portfolio figcaption {
  position: absolute;

  width: 100%;
  height: 100%;

  left: 0;
  top: 0;

  background-color: rgba(58,63,67,.8);

  padding: 10%;
}
```

As we have set the box sizing to `border-box` earlier, the caption size (height and width) remains `100%` regardless of the padding addition. The caption size still follows the parent size, which in our case is 240 px x 240 px.

11. Since the background color is dark, we need to set a lighter color for the caption text. In this case, we change the caption text color to white (`#fff`).

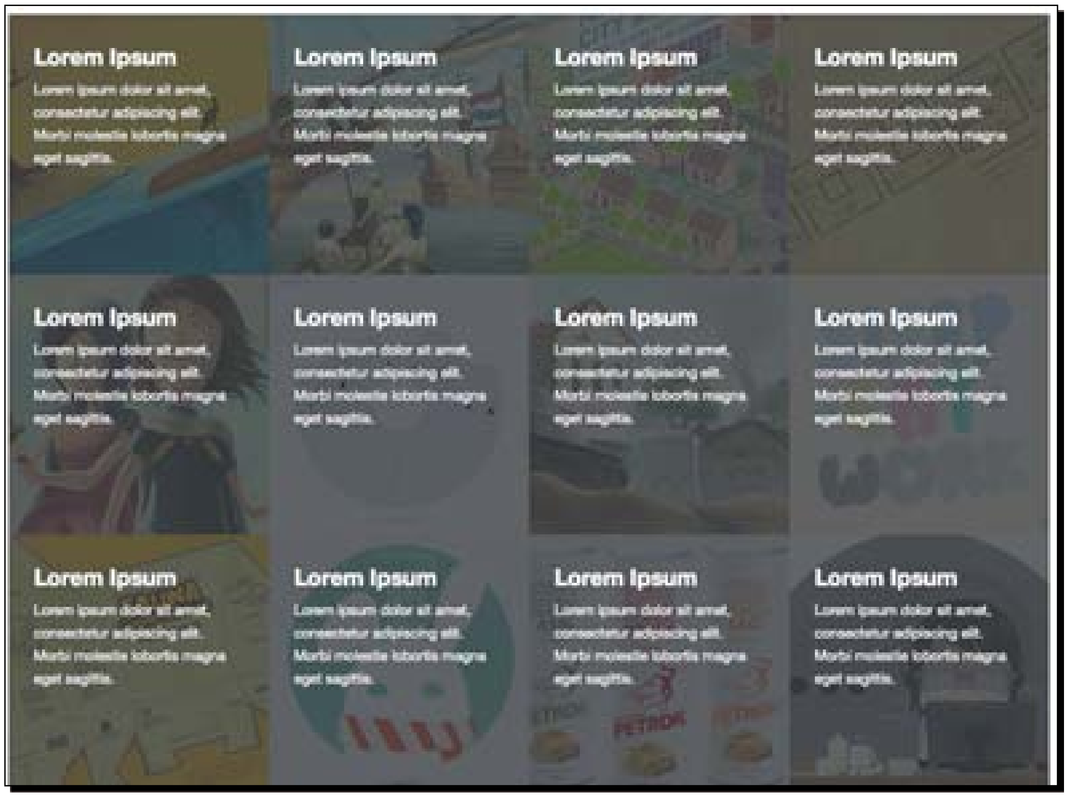
```
.portfolio figcaption h4 {
  color: #fff;
}
.portfolio figcaption p {
  color: #fff;
}
```

12. This is only a matter of preferences, but the text paragraph in the caption seems too big. So, we set the caption text paragraph to `1px` smaller than the base size. The base font size is `14px`, so `13px` is equal to `0.929em`.

```
.portfolio figcaption p {
  color: #fff;
  font-size: 0.87em;
}
```

What just happened?

We have just added style rules for the image thumbnail and the caption. At this stage, our portfolio section in the website appears as is shown in the following screenshot:



Lorem Ipsum
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi molestie lobortis magna eget sagittis.

Lorem Ipsum
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi molestie lobortis magna eget sagittis.

Lorem Ipsum
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi molestie lobortis magna eget sagittis.

Lorem Ipsum
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi molestie lobortis magna eget sagittis.

Lorem Ipsum
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi molestie lobortis magna eget sagittis.

Lorem Ipsum
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi molestie lobortis magna eget sagittis.

Lorem Ipsum
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi molestie lobortis magna eget sagittis.

Lorem Ipsum
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi molestie lobortis magna eget sagittis.

Lorem Ipsum
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi molestie lobortis magna eget sagittis.

Lorem Ipsum
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi molestie lobortis magna eget sagittis.

Lorem Ipsum
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi molestie lobortis magna eget sagittis.

Lorem Ipsum
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi molestie lobortis magna eget sagittis.

CSS3 2D Transformations

Over the last couple of years, several new CSS3 features have been released, including the CSS3 Transform (`http://www.w3.org/TR/css3-transforms/`). Using CSS3 Transform, we can translate, rotate, skew, and scale HTML elements.

The translate() function

The `translate()` function in CSS3 Transforms is used for moving elements relative to their original position. This function is declared with the following syntaxes:

- To move the element in the horizontal direction, we can write:
  ```
  transform: translateX(value);
  ```
- To move the element in the vertical direction, we can write:
  ```
  transform: translateY(value);
  ```
- Another way we can use the shorthand syntax is by combining the x and y values, as follows:
  ```
  transform: translate(x-value,y-value);
  ```

As you can see, the position of the element is specified with the x and y values, where x represents the horizontal coordinate and y represents the vertical coordinate. This principle relates to the Cartesian coordinate system (`http://en.wikipedia.org/wiki/Cartesian_coordinate_system`).

However, since the web page is read sequentially from top to bottom, the y coordinate is reversed; a negative y value specifies upward motion whereas a positive y value specifies downward motion.

Now let's say that we would like to move an element `100px` to the right. We can write this in the following way:

```
transform: translateX(100px)
```

Or we can specify it with the shorthand syntax, as follows:

```
transform: translate(100px,0)
```

Similarly, when we want to move it upward by `100px`, we can write this as follows:

```
transform: translateY(-100px)
```

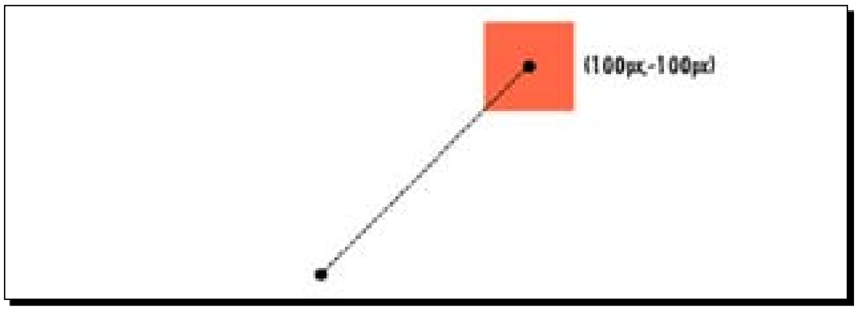
Or we can also write this in the following way:

```
transform: translate(0,-100px)
```

In addition to this, to move the element diagonally, we'll specify both the x and the y coordinates, as follows:

```
transform: translate(100px,-100px)
```

The preceding declaration will move the element upward and to the right, as illustrated in the following screenshot:



Vendor prefixes

The `translate()` function is supported in Google Chrome 4, Safari 3.1, Firefox 3.5, Internet Explorer 10, and Opera 10.5, though vendor prefixes are still required to make the transformations work on these browsers. So, the complete CSS rule to run a transformation is as follows:

```
-webkit-transform: translate(x,y); /* Webkit (eg. Chrome & Safari) */
-moz-transform: translate(x,y); /* Mozilla Firefox */
-ms-transform: translate(x,y); /* Internet Explorer */
-o-transform: translate(x,y); /* Opera */
transform: translate(x,y); /* Recommendation syntax from W3C */
```

The browsers will apply their specific prefix, for example, `-webkit-` will be implemented in WebKit-based browsers, such as Chrome and Safari, and ignore the other prefixes. Later, when the specification has been finalized and the browsers have fully applied it, the standard syntax from the W3C (`http://www.w3.org/`) is the one that will be applied.

So, it is a good idea to include all vendor prefixes for better browser compatibility.

In the above code snippet, we have to write five different lines that technically do the same thing to cover browser's capability. So if writing the vendor prefix manually seems to be a tedious task, there are several tools to help us deal with the vendor prefix.

Prefixr (`http://prefixr.com/index.php`) allows us to generate the vendor prefix quickly. If you are using Sublime Text 2, there is a package that allows you to run the prefix directly from the editor.

There is a JavaScript library called Prefix Free (`http://leaverou.github.com/prefixfree/`) that allows us to write the unprefixed syntax as it will append the needed vendor prefix on the fly.

CSS3 Transition

Another great addition in CSS3 is Transition. A CSS3 Transition allows us to change one CSS rule to another CSS rule gradually—rather than instantaneously—within a specific duration. A CSS3 Transition is defined with the following syntax (including the vendor prefix):

```
-webkit-transition: property duration timing-function delay; /* Webkit
(eg. Chrome & Safari) */
-moz-transition: property duration timing-function delay; /* Mozilla
Firefox */
-o-transition: property duration timing-function delay; /* Opera */
transition: property duration timing-function delay; /* Recommendation
syntax from W3C */
```

Currently, CSS3 Transition is supported in the Chrome 4, Firefox 4, Safari 3.1, Opera 10.5, and Internet Explorer 10.0 browsers (`http://caniuse.com/#feat=css-transitions`).

Internet Explorer 9 does not support CSS3 Transition; this is why you do not see the `-ms-` prefix in the preceding syntax. But, Internet Explorer 10 will support CSS3 Transition without the prefix (`http://msdn.microsoft.com/en-us/library/ie/hh673535(v=vs.85).aspx`).

CSS3 Transition values

As you can see, four values have been specified in the syntax, namely `property`, `transition-duration`, `timing-function`, and `delay`. Let's peel them up one by one:

Value	**Use**
`property`	This value targets the CSS property to which the transition effect should be applied. The property could be `width`, `height`, `color`, `background`, and so on. But when this value is not explicitly specified, it will take `all` as the default value, which will apply the transition to all properties.
`transition-duration`	This value specifies the length of transition effect; this value is specified in milliseconds (ms) and seconds (s). For instance, `200ms` and `0.2s`.
`timing-function`	This value specifies the transition acceleration. There are five predefined acceleration types that we can use; they are `ease`, `ease-in`, `ease-out`, `ease-in-out`, and `linear`. You can see how these timing functions play in a post, at `http://www.css3.info/preview/css3-transitions/`.
`delay`	This value sets the delay time before the transition effect starts.

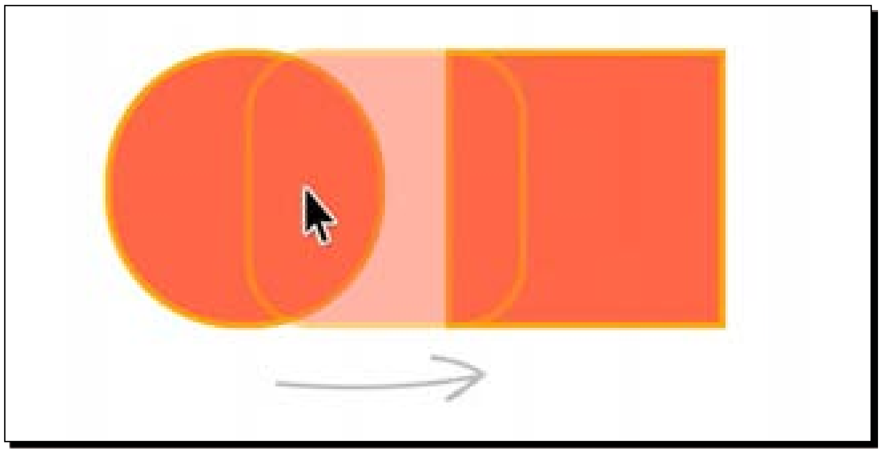
In the following code example, we have created a circle with a `div` element and we want to turn it into a rectangle when we hover over it.

```
div {
  width: 200px;
 height: 200px;
  border-radius: 100px;
 border: 5px solid orange;
  background-color: tomato;
}
div:hover {
  border-radius: 0;
}
```

As mentioned earlier in the chapter, since we did not add the transition, the change will be instantaneous. Now, let's add the transition effect to `border-radius`, which is set to `200ms`, as follows:

```
div {
/* the other rules, same as above*/

  -webkit-transition: border-radius 200ms;
  -moz-transition: border-radius 200ms;
  -o-transition: border-radius 200ms;
  transition: border-radius 200ms;
}
```

After that, the changes will take effect gradually and look more appealing, as illustrated in the following screenshot:



Time for action – creating a thumbnail hover effect

To create a thumbnail hover effect, perform the following steps:

1. Open `style.css`. Remember that we structure the caption with the HTML5 `<figcaption>` element. We have added several CSS rules in the previous steps, such as specifying the position, setting the width and height, setting the background color, changing the font color, and adding padding.

   This time, we will add CSS rules to create the hover effect. The idea here is that when we hover the thumbnail, the caption will gradually slide from a specific direction and cover the image.

   In the following CSS rule, we first move the caption to the right with CSS3 Transform:

   ```
   .portoflio figcaption {
     position: absolute;
     left: 0;
     top: 0;

   width: 100%;
     height: 100%;
     padding: 10%;

   background-color: rgba(58,63,67,.8);

     -webkit-transform: translateX(100%);
     -moz-transform: translateX(100%);
     -ms-transform: translateX(100%);
     -o-transform: translateX(100%);
     transform: translateX(100%);
   }
   ```

2. Then, we add the transition effect with CSS3 Transition. In the following rule, we set the transition for all the elements to `350ms`.

   ```
   .portoflio figcaption {
     position: absolute;
     left: 0;
     top: 0;

     width: 100%;
     height: 100%;
     padding: 10%;

     background-color: rgba(58,63,67,.8);
   ```

```
  -webkit-transform: translateX(100%);
  -moz-transform: translateX(100%);
  -ms-transform: translateX(100%);
  -o-transform: translateX(100%);
  transform: translateX(100%);

  -webkit-transition: all 350ms;
  -moz-transition: all 350ms;
  -o-transition: all 350ms;
  transition: all 350ms;
}
```

3. Lastly, we add the `hover` state. In the `hover` state, we set the caption to its original position by specifying `0` for `translateX`.

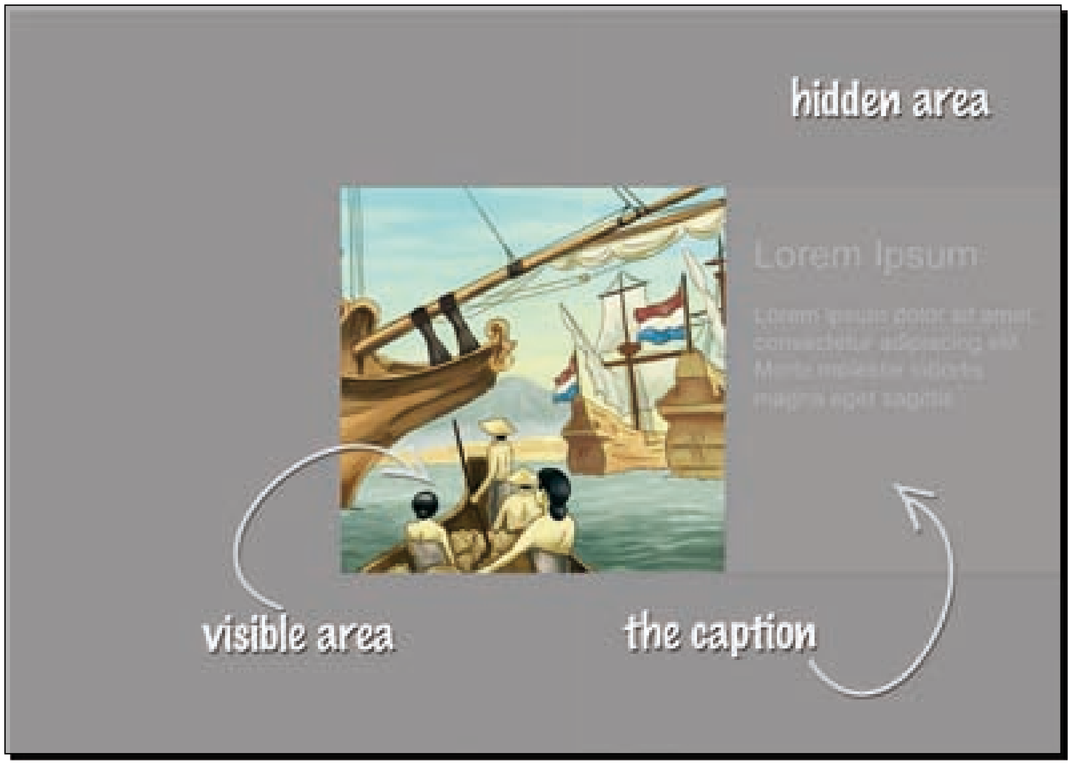
```
.container figure:hover figcaption {
  -webkit-transform: translateX(0);
  -moz-transform: translateX(0);
  -ms-transform: translateX(0);
  -o-transform: translateX(0);
  transform: translateX(0);
}
```

What just happened?

Well, there are several things that happen in this code. First, we reposition the caption with the `transform` property to `100%` of the parent's width to the right, as follows:

```
-webkit-transform: translateX(100%);
-moz-transform: translateX(100%);
-ms-transform: translateX(100%);
-o-transform: translateX(100%);
transform: translateX(100%);
```

At this point, the caption will not be visible because we have set the outer area of the `figure` element to be hidden. We have created a graphic to illustrate this.

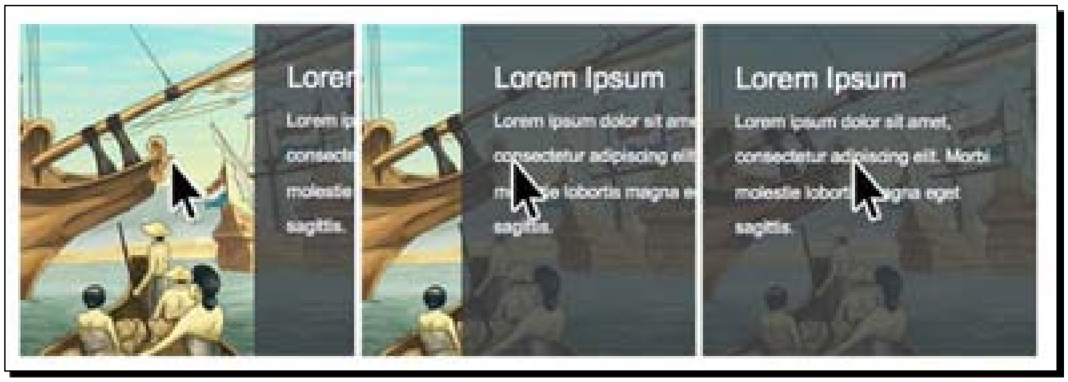


Then, we apply the transition effect to the caption and apply it to all the properties assigned with that caption:

```
-webkit-transition: all 350ms;
-moz-transition: all 350ms;
-o-transition: all 350ms;
transition: all 350ms;
```

Lastly, we add the styles to the `hover` state. The idea is that the caption will move from right to left and back to its original position. That is why we specified the translate coordinate as `0%` in the `hover` state.

After adding all the CSS rules mentioned in this section, you should be able to see the hover effect:


Lorem Ipsum
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi molestie lobortis magna eget sagittis.

Website navigation for filtering the portfolio

As we mentioned, website navigation is used to sort portfolios into their respective categories. Thus, our website navigation has been created with a radio input type followed by a label, as follows:

```
<input class="nav-menu" id="all" type="radio" name="filter"
checked="checked"/>
<label for="all">All</label>
<input class="nav-menu" id="illustrations" type="radio"
name="filter"/>
<label for="illustrations">Illustration</label>
<input class="nav-menu" id="posters" type="radio" name="filter"/>
<label for="posters">Posters Design</label>
<input class="nav-menu" id="typography" type="radio" name="filter"/>
<label for="typography">Typography</label>
<input class="nav-menu" id="packaging" type="radio" name="filter"/>
<label for="packaging">Packaging</label>
```

Time for action – creating a portfolio filter

To create a portfolio filter, perform the following steps:

1. Open `styles.css`. Our navigation is based on a radio input type that is assigned with the class `nav-menu`. First, we hide the radio button.

   ```
   .nav-menu {
     display: none;
   }
   ```

2. Then, we add styles to the input `<label>`. In the following rule, we set the label's `display` property to `inline-block` so the labels will be displayed beside each other.

   ```
   label {
     padding: 5px 10px;
     color: #3a3f43;
     cursor: pointer;
     display: inline-block;
   }
   ```

3. When the `<input>` radio button is selected, the `<label>` element's styles change. In this case, we set the background color to darker than it is to show that the menu is selected, so we also need to turn the text color to white.

   To select the `<label>` element next to the selected `<input>` radio, we use an adjacent selector.

   ```
   .nav-menu:checked + label {
     color: #fff;
     background-color: #3a3f43;
     border-radius: 3px;
   }
   ```

4. Next, we define the filter function. First, we hide the portfolio thumbnail. We define the following CSS rule with `.nav-menu`, as follows:

   ```
   .nav-menu,
   .portfolio > figure.columns {
     display: none;
   }
   ```

5. To achieve the filter functionality, we will combine several CSS selectors. First of all, we structure the website navigation with input radio type and each of them is assigned with a unique ID. These inputs have a label linked to them using the `for"[id]"` attribute. So when we click on the label, the input button is checked. Then, we can target these checked inputs using the `:checked` pseudo class from CSS3. In the following code, we first select the checked `<input>` radio that has an ID of `all`.

   ```
   #all:checked
   ```

   Coupled with the adjacent selector, we select the `portfolio` section:

   ```
   #all:checked ~ .portfolio
   ```

   After selecting the `portfolio` section, we can select the child elements inside it and apply the CSS rules. In this case, we will target elements that have the `all` class and set `display` to `block`.

   ```
   #all:checked ~ .portfolio .all {
     display: block;
   }
   ```

   In this way, all portfolio thumbnails are visible when the `<input>` radio with an ID of `all` is checked.

6. Now, let's add the same thing for specific categories such as `poster`, `illustration`, typography, and package, as follows:

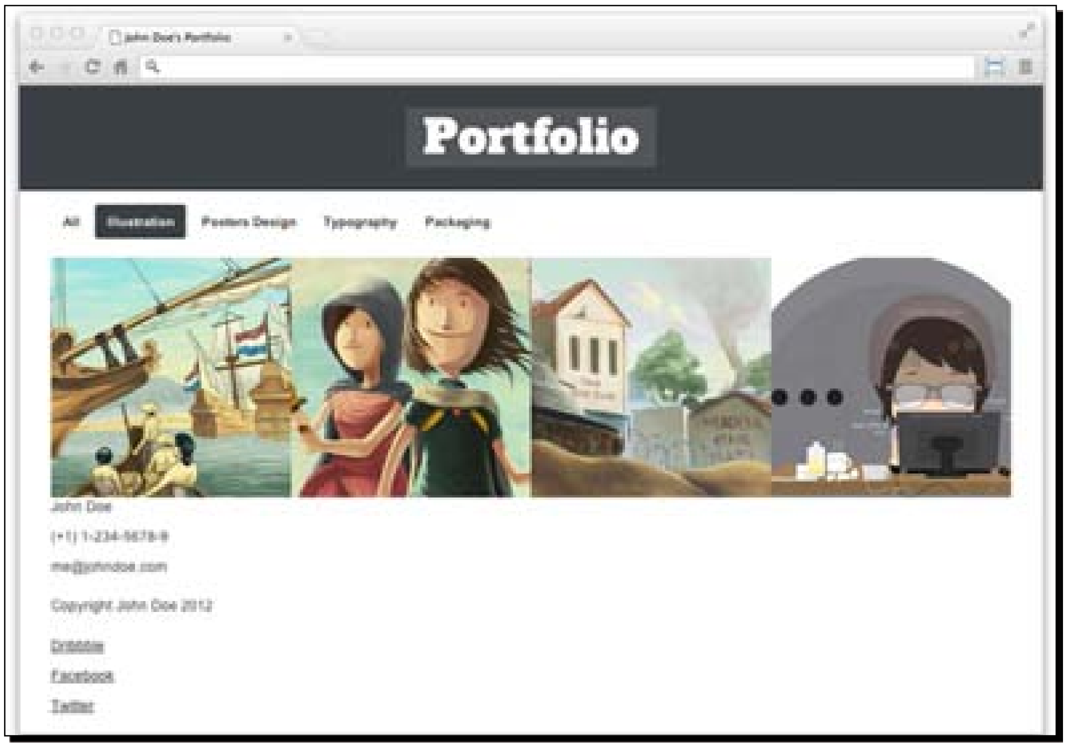
   ```
   #all:checked ~ .portfolio .all,
   #posters:checked ~ .portfolio .poster,
   #illustrations:checked ~ .portfolio .illustration,
   #typography:checked ~ .portfolio .typography,
   #packaging:checked ~ .portfolio .package {
     display: block;
   }
   ```

What just happened?

We've just added styles for the purpose of navigation and built the filter functionality with a combination of some CSS selectors.

Despite being officially announced as part of CSS3 specification, the `:checked` pseudo class has actually been supported since as early as Firefox 1. To see how it works, you can view the demo from the W3C website that is available at `http://www.w3.org/Style/CSS/Test/CSS3/Selectors/current/html/full/flat/css3-modsel-25.html`.

At this point, you can sort the portfolios based on their categories with the navigation menu, as shown in the following screenshot:



Footer section

In this section, we will add styles to the footer section. This section contains the link to our social presence and contact information, such as our phone number, e-mail, and name.

Time for action – styling the footer section

To style the footer section, perform the following steps:

1. We are still working within the `styles.css` file. Our website footer is defined with the HTML5 `<footer>` element and assigned to the class `footer`. First of all, we add some decorative styles, such as a margin, padding, and border line, as follows:

```
.footer {
  border-top: 1px solid #ccc;
  margin-top: 28px;
  padding: 28px 0;
}
```

2. Next, we place the social links' profiles on the left side of the page, as follows:

   ```
   .social {
     float: left;
   }
   ```

3. Inside the social link section, we have a small space to place the website's copyright text. We will change the color of this text to gray and add a small gap with `margin-bottom`.

   ```
   .social .copyright {
     color: #ccc;
     margin-bottom: 10px;
     font-size: 1em;
   }
   ```

4. The social links are structured with the `<li>` element. We need to display them side by side.

   ```
   .social ul li {
     display: inline;
   }
   ```

5. In this step, we will add styles for the links. First, in order to be able to set the width and height, we need to set the anchor element `display` to `inline-block`. Then, we set the `width` to `42px` and `height` to `36px`.

   ```
   .social ul li a {
     display: inline-block;
     width: 36px;
     height: 42px;
   }
   ```

6. We also add what's called CSS image replacement styles to hide the text inside the anchor element and replace it with an image later through the `background-image` property.

   ```
   .social ul li a {
     display: inline-block;
     width: 48px;
     height: 48px;

   /*below is the css image replacement styles*/
     text-indent: 100%;
     white-space: nowrap;
     overflow: hidden;
   }
   ```

7. We add the social icons, which we have concatenated into a single sprite file in *Chapter 2, Constructing a Responsive Portfolio Page with Skeleton*, to the <a> element with the `background-image` property:

```
.social-dribbble a,
.social-facebook a,
.social-twitter a {
  background-image: url('../images/social.png');
  background-repeat: no-repeat;
}
```

8. Next, we need to edit the CSS rules that were generated when we concatenated the social icon images. These CSS rules define the icon image's position:

```
.social-dribbble-hover{
  background-position: 0 0;
  width: 48px;
  height: 48px;
}
.social-dribbble{
  background-position: 0 -58px;
  width: 48px;
  height: 48px;
}
.social-facebook-hover{
  background-position: 0 -116px;
  width: 48px;
  height: 48px;
}
.social-facebook{
  background-position: 0 -174px;
  width: 48px;
  height: 48px;
}
.social-twitter-hover{
  background-position: 0 -232px;
  width: 48px;
  height: 48px;
}
.social-twitter{
  background-position: 0 -290px;
  width: 48px;
  height: 48px;
}
```

First, we change the class name that defines the hover state (`.social-dribbble-hover`) with `:hover` and assign it to the link icons, as follows:

```
.social-dribbble a:hover {
  background-position: 0 0;
  width: 48px;
  height: 48px;
}
.social-dribbble{
  background-position: 0 -58px;
  width: 48px;
  height: 48px;
}
.social-facebook a:hover{
  background-position: 0 -116px;
  width: 48px;
  height: 48px;
}
.social-facebook{
  background-position: 0 -174px;
  width: 48px;
  height: 48px;
}
.social-twitter a:hover{
  background-position: 0 -232px;
  width: 48px;
  height: 48px;
}
.social-twitter{
  background-position: 0 -290px;
  width: 48px;
  height: 48px;
}
```

9. Since we have set the width and height in the previous step, we can remove the height and width definition from these CSS rules, as follows:

```
.social-dribbble a:hover {
  background-position: 0 0;
}
.social-dribbble{
  background-position: 0 -58px;
}
.social-facebook a:hover{
  background-position: 0 -116px;
}
.social-facebook{
  background-position: 0 -174px;
}
.social-twitter a:hover{
  background-position: 0 -232px;
}
.social-twitter{
  background-position: 0 -290px;
}
```

10. ,We place the `contact` section to the right of the page:

```
.contact {
  float: right;
}
```

11. And change the color of the text and text link to gray.

```
.contact, .contact a {
  color: #ccc;
}
```

12. Then, we add the icons for the contacts, which we have concatenated into one sprite file in *Chapter 2*, *Constructing a Responsive Portfolio Page with Skeleton*, through the `background-image` property. But this time we will add them in the `:before` pseudo element.

    The `:before` pseudo element adds the element before the content of the element that has been specified. It has a sibling named `:after`, which adds the element after the content of the specified element.

    In an HTML structure, this can be illustrated as follows:

```
<div>
  <span> </span> <!-- :before -->
  Content
  <span> </span> <!-- :after -->
</div>
```

    But, a pseudo element does not add an actual or physical element; that is why, it is called **pseudo**. When we add a pseudo element, it will be interpreted as if the element exists in the document (but it does not).

    In CSS3, the pseudo element's syntax is revised. The syntax is defined with double colons (`::before` or `::after`) to differentiate it from a pseudo class, which uses the single-colon syntax (`:hover` or `:checked`).

    In the following code snippet, we add the `:before` pseudo element to `<li>`. We change `display` to `inline-block` in order to be able to set `width` and `height`.

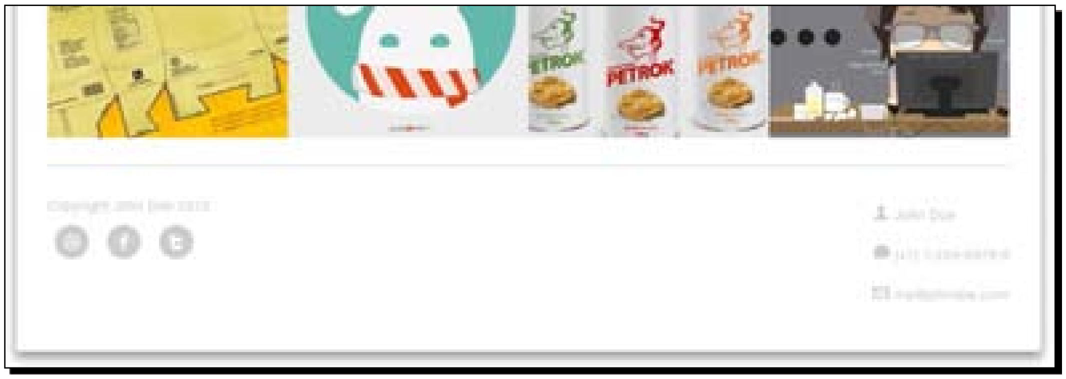
```
.contact ul li:before {
  content: '';
  display: inline-block;
  width: 24px;
  height: 24px;
  background-image: url('../images/contact.png');
  margin-right: 0.1em;
}
```

13. Lastly, we adjust the background position for the contact's icon image.

```
.contact-name:before {
  background-position: 0 -29px;
}
.contact-phone:before {
  background-position: 0 -63px;
}
.contact-email:before{
  background-position: 0 5px;
}
```

What just happened?

We just added styles to the footer section and included the social and contact sections inside it. The following screenshot shows how the footer section will appear:



Adjusting website styles in a smaller viewport

In this section, we are going to add styles for a specific viewport using CSS3 media queries.

Now, before we start adding styles, we need to copy all the media queries defined in Skeleton's `layout.css` into our `styles.css` file. The following code snippet shows how they appear in the file:

```
/* Smaller than standard 960 (devices and browsers) */
@media only screen and (max-width: 959px) {

}
/* Tablet Portrait size to standard 960 (devices and browsers) */
@media only screen and (min-width: 768px) and (max-width: 959px) {

}
```

```
/* All Mobile Sizes (devices and browser) */
@media only screen and (max-width: 767px) {

}
/* Mobile Landscape Size to Tablet Portrait (devices and browsers) */
@media only screen and (min-width: 480px) and (max-width: 767px) {

}
/* Mobile Portrait Size to Mobile Landscape Size (devices and
browsers) */
@media only screen and (max-width: 479px) {

}
```

Time for action – viewport size less than 960 px

We are about to add styles that are applicable to a viewport size that is less than 960 pixels:

1. First, we are going to put these styles inside the following media query. This media query specifies the styles for viewports that are smaller than 960 px.

   ```
   @media only screen and (max-width: 959px) {
   }
   ```

2. Since the device is getting smaller, we need to change the column and container widths to their relative units. In this case, the container width would be `100%` while the width of each column within the container would be 25%, as the container is divided into four columns.

   ```
   .container {
     width: 100%;
   }
   .portfolio .four.columns {
     width: 25%;
   }
   ```

3. Then, we hide the navigation menu to let the users navigate by scrolling with their fingers:

   ```
   label {
     display: none;
   }
   ```

4. We add a little gap at the bottom of each row by adding `margin-bottom` to the figure element.

   ```
   .portfolio .all {
     margin-bottom: 15px;
   }
   ```

5. Since we have hidden the navigation, we need to shift the category information somewhere else.

   In this case, we will place it at the top of the image. We can add it using the `:before` pseudo element and grab the category information from the HTML5 `data-*` attribute, as follows:

```
.portofolio > figure:before {
  content: attr(data-category);
  font-size: 1em;
  padding: 8px;
  width: 100%;
  color: #fff;
  display: block;
  font-weight: bold;
  text-transform: capitalize;
  background-color: rgba(42,47,51,0.8);
  position: absolute;
}
```

6. In this smaller viewport, we will show the image caption instead of hiding it. So we'll set `position` to `relative` and the `translateX` to `0%`. We'll also set the default background color for the caption.

```
.portfolio figcaption {
  position: relative;

  -webkit-transform: translateX(0%);
  -moz-transform: translateX(0%);
  -ms-transform: translateX(0%);
  -o-transform: translateX(0%);
  transform: translateX(0%);

  background-color: #3a3f43;
}
```

7. We use the `nth-child` pseudo element to select the `<figure>` element that is set with an odd order and set the background color darker than the default that we set in Step 6. In this way, each portfolio caption is distinguishable from the other.

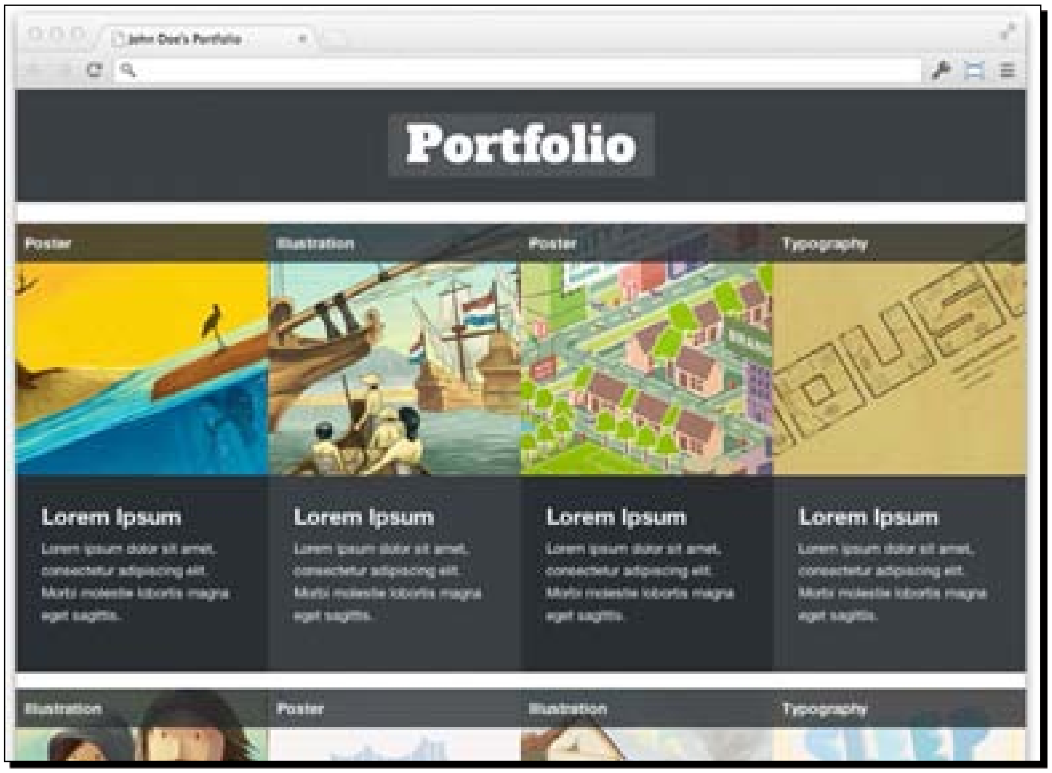
```
.portoflio figure:nth-child(odd) figcaption {
  background-color: #2a2f33;
}
```

8. Lastly, we need to adjust the margin of the footer section.

```
.footer {
  border-top: 1px solid #ccc;
  margin-top: 42px;
  padding: 28px;
}
```

What just happened?

We just added styles for a viewport with a width less than 960 px. Here is the result:



Time for action – viewport size between 767 px and 480 px

This time we are going to add styles when the viewport size is between 767 px and 480 px. This size is most likely the size of mobile and tablet devices.

1. We will add styles inside the following media query:

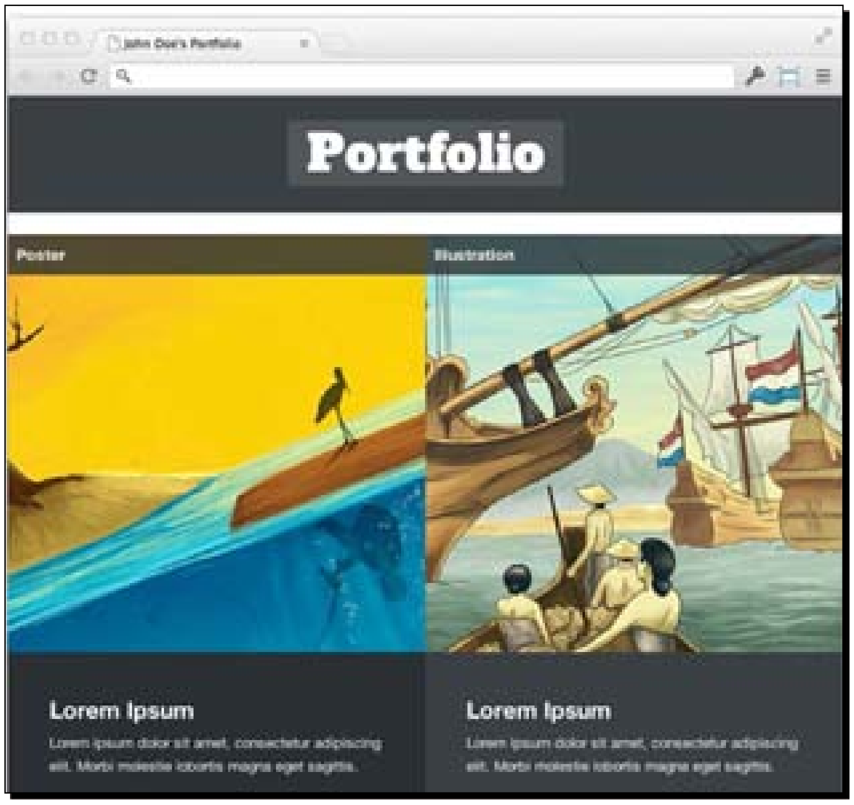
```
@media only screen and (min-width: 480px) and (max-width: 767px) {

}
```

2. Since the viewport is getting smaller, we need to divide the columns into bigger sizes. In this case, we divide the `100%` width of the container by 2, so each column will have `width` set to `50%` of the viewport size.

```
.portfolio .four.columns {
  width: 50%;
}
```

What just happened?

We just added styles for a website that has a viewport size between 767 px and 480 px, and the following screenshot shows how it will appear:


John Doe's Portfolio
Portfolio
Poster
Illustration
Lorem Ipsum
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi molestie lobortis magna eget sagittis.
Lorem Ipsum
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi molestie lobortis magna eget sagittis.

Time for action – viewport size less than 480 px

This time we are going to add styles for a viewport size that is less than 480 px:

1. The styles for a viewport with a size less than 480 px will be added inside the following media query:

```
@media only screen and (max-width: 479px) {
}
```

2. Since the viewport size is really small, we will set the column's `width` attribute to `100%` as well so the image is more visible.

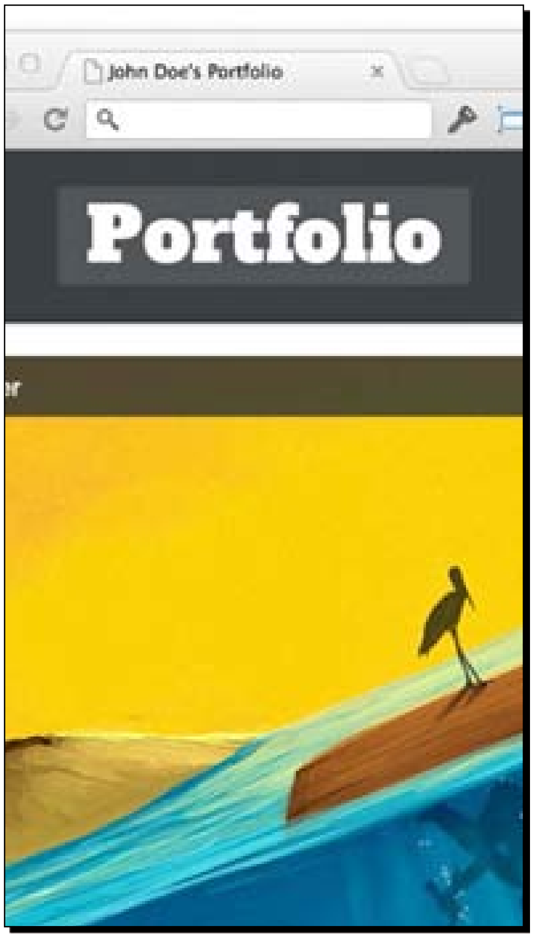
```
.portfolio .four.columns  {
  width: 100%;
}
```

3. In the footer section, we will remove the float definition and set `text-align` to `center`.

```
.footer {
  text-align: center;
}
.contact, .social {
  float: none;
  display: block;
}
```

What just happened?

We just added the styles for our website when it is viewed in a viewport size less than 480 px. The following screenshot shows how it appears:



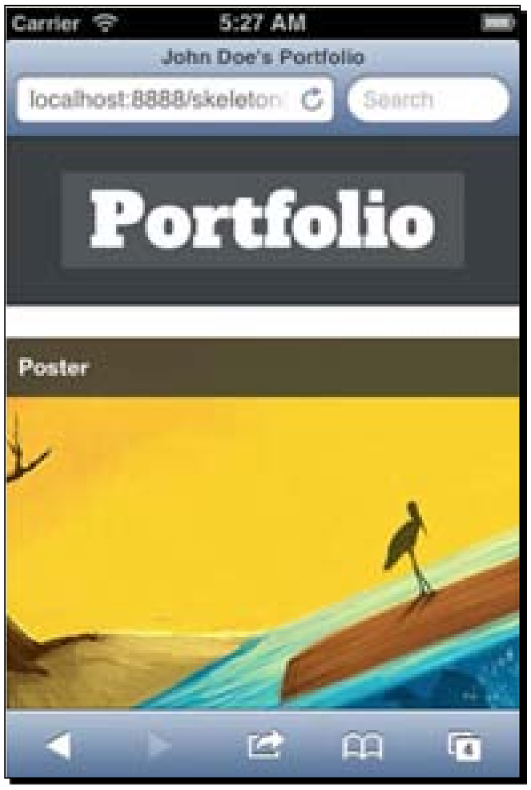
Testing the website in a different viewport size

We are done with the website, and it is ready for testing. During this process, we test the website in a desktop browser and in a smaller viewport size only by minimizing the browser window. Alternatively, we can also test it with some other tools, such as the following:

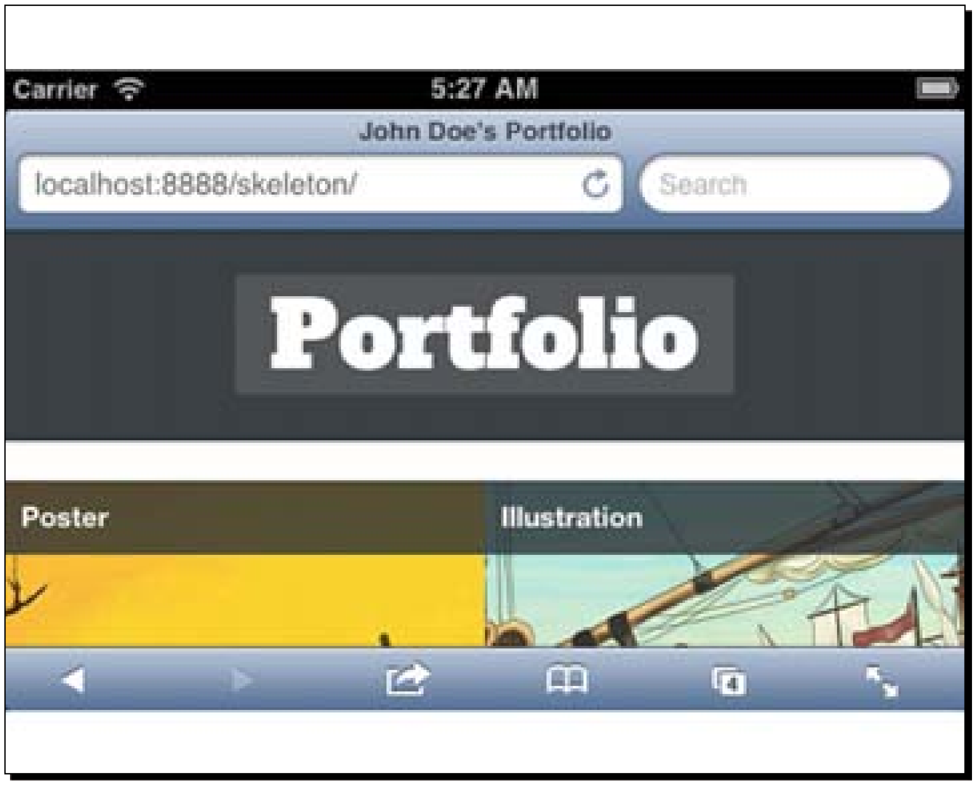
- Firefox's built-in Responsive Tool (`https://developer.mozilla.org/en-US/docs/Tools/Responsive_Design_View`)
- Responsinator (`http://www.responsinator.com/`)
- Screenqueries (`http://screenqueri.es/`)

It is better to test the website in real devices—phones, tablets, or readers—to see how the website actually responds. The following screenshots shows how our website from this first project is displayed in iPhone and iPad.

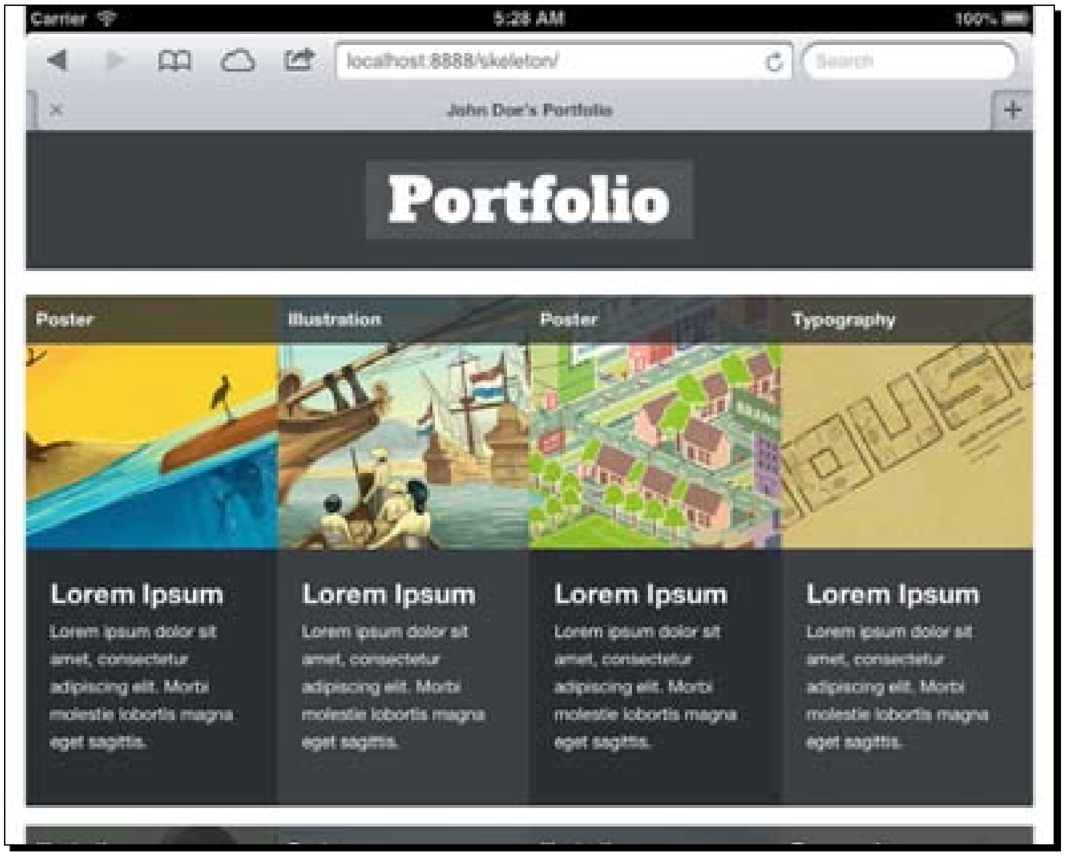
The following screenshot shows our website when viewed in iPhone with the portrait screen orientation. In this screen orientation, where the viewport size is really small (320 px x 480 px), the navigation is hidden and replaced by the category name that is shown above each of the portfolio thumbnails. The image caption is also viewable next to each portfolio thumbnail.



The following screenshot shows how our website will appear when viewed in iPhone with the landscape screen orientation. Like in portrait orientation, the navigation is hidden. But since the viewport width is wider—480 px x 320 px—we can display two portfolio image thumbnails in a row, side by side.



The following screenshot shows how our website will appear in iPad with the portrait screen orientation. As the viewport size is much wider than that of the iPhone (768 px x 1024 px), we are able to accommodate four portfolio image thumbnails in a row, and the caption is also visible below each thumbnail.



Lorem Ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi molestie lobortis magna eget sagittis.

Lorem Ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi molestie lobortis magna eget sagittis.

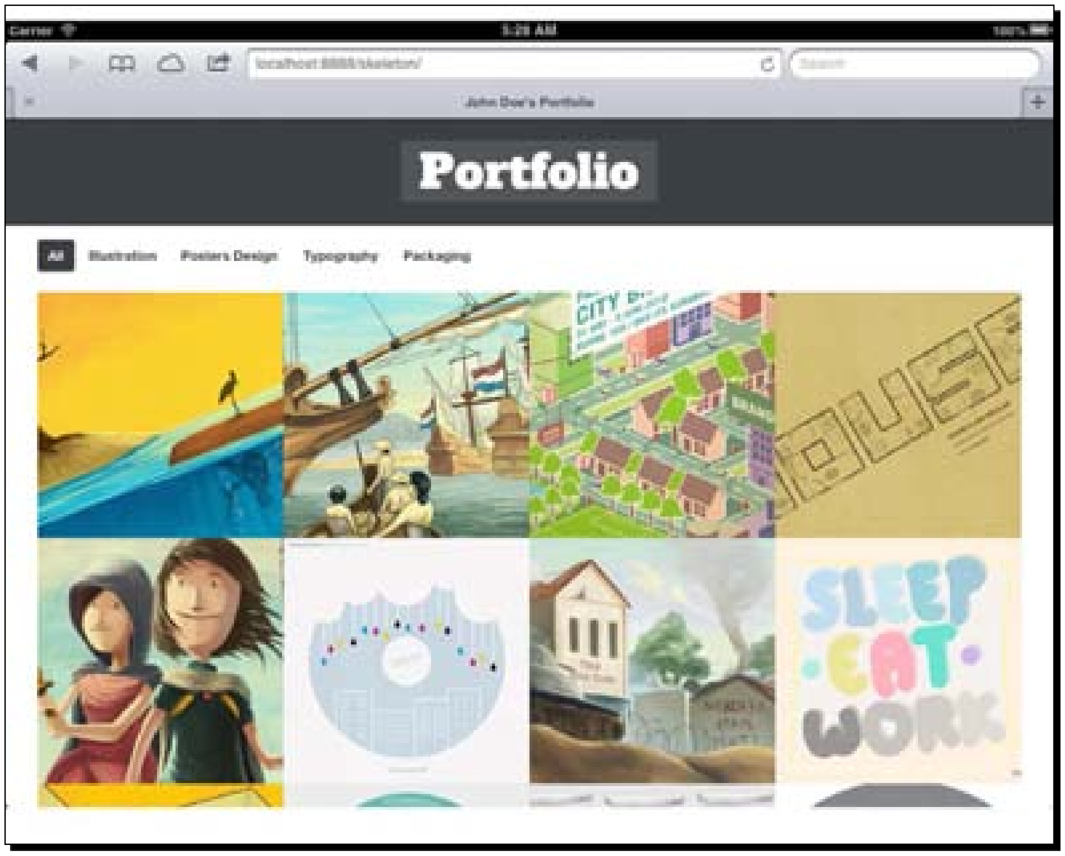
Lorem Ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi molestie lobortis magna eget sagittis.

Lorem Ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi molestie lobortis magna eget sagittis.

Now, we will see how our website appears in iPad with the landscape screen orientation. In this orientation, the viewport size is 1024 px x 768 px; this space is wide enough to accommodate four image thumbnails in one row. The navigation for this viewport size is visible; we can tap on the navigation menu to sort the portfolio.



Summary

We have just finalized our first responsive website with CSS3. In this chapter, we performed the following tasks:

- Polished our website with some new properties introduced in CSS3, such as `box-sizing`, `border-radius`, and `box-shadow`
- Created a fancy image hover effect with CSS3 Transforms and Transitions
- Created a portfolio filter function with a combination of CSS selectors
- Adjusted our website's styles in different viewport sizes with CSS3 media queries

Now that we are done with our first project with Skeleton, we are going to explore another framework to create a responsive website in the next chapter.