6 Creating a profile card

This chapter covers

- Using CSS custom properties
- Creating a background using `radial-gradient`
- Setting image size
- Positioning elements using a flexbox

In this chapter, we'll create a profile card. In web design, a card is a visual element that contains information on a single topic. We're going to apply this concept to someone's profile information, essentially creating a digital business card. This type of layout is often used on social media and blog sites to give readers an overview of who wrote the content. It sometimes has links to a detailed profile page or opportunities to interact with the person to whom the profile belongs.

To create the layout, we'll do a lot of work revolving around positioning, specifically, using the CSS Flexbox Layout Module to align and center elements. We'll also look at how to make a rectangular image fit into a circle without distorting the image. By the end of the chapter, our profile card will look like figure 6.1.

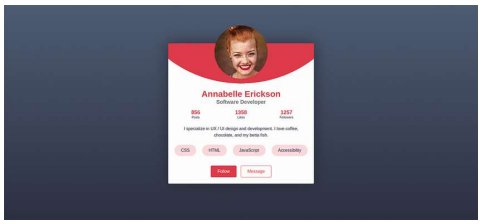

Figure 6.1 Final output

.1 Starting the project

Let's dive right in and take a look at our starting HTML (listing 6.1), which you can find in the GitHub repository at **http://mng.bz/5197** or on CodePen at **https://codepen.io/michaelgearon/pen/NWyByWN**. We have a `<div>` with a class of `card` that contains all the elements being presented in the profile card. To set our blog post information, we'll use a description list. Our technologies (CSS, HTML, and so on) are presented in a list.

Description list

A *description list* contains groups of terms, including a description term (`dt`) and any number of descriptions (`dd`). Description lists are often used to create glossaries or to display metadata. Because we're pairing terms (posts, likes, and followers) with their counts (the number), this project is a great use case for a description list.

Listing 6.1 Project HTML

```
<body>
  <div class="card">                                                    ①
    <img class="portrait" src="./img/portrait.jpg" alt="">              ②
    <h1>Annabelle Erickson</h1>                                         ③
    <div class="title">Software Developer</div>                         ④
    <dl>                                                                ⑤
      <div>                                                             ⑤
        <dt>Posts</dt>                                                  ⑤
        <dd>856</dd>                                                    ⑤
      </div>                                                            ⑤
      <div>                                                             ⑤
        <dt>Likes</dt>                                                  ⑤
        <dd>1358</dd>                                                   ⑤
      </div>                                                            ⑤
      <div>                                                             ⑤
        <dt>Followers</dt>                                              ⑤
        <dd>1257</dd>                                                   ⑤
      </div>                                                            ⑤
    </dl>                                                               ⑤
    <p class="summary">I specialize in UX / UI...</p>                   ⑥
    <ul class="technologies">                                           ⑦
      <li>CSS</li>                                                      ⑦
      <li>HTML</li>                                                     ⑦
```

```
        <li>JavaScript</li>                                          ⑦
        <li>Accessibility</li>                                       ⑦
      </ul>                                                          ⑦
      <div class="actions">                                          ⑧
        <button type="button" class="follow">Follow</button>         ⑧
        <a href="#" class="message">Message</a>                      ⑧
      </div>                                                         ⑧
    </div>                                                           ⑨
  </body>
```

① Start of the card

② Profile Image

③ Profile holder's name

④ Profile holder's job title

⑤ Post information

⑥ Personal summary/about

⑦ Technologies

⑧ Actions

⑨ End of the card

As we begin styling our card, our page looks like figure 6.2.

.2 Setting CSS custom properties

In our layout, specifically, when we style the profile image and colored portion at the top of the card below the image, we're going to need the

image-size value for several calculations. In languages such as JavaScript, when we have a value that we're going to be referencing multiple times, we use *custom properties,* sometimes referred to as *CSS variables*.

To create a custom property, we prefix the variable name with two hyphens (`--`) immediately followed by the variable name. We assign the value to a custom property the same way that we do any other property: with a colon (`:`) followed by the value. A CSS variable declaration, therefore, looks like this: `--myVariableName: myValue;`.

As with any other declaration, we need to define our variables inside a rule. For our project, we're going to define our colors and image size and then declare them inside a `body` rule, as shown in listing 6.2. Because we define our variables on the `body` , the `<body>` element and any of its descendants will have access to the variables.

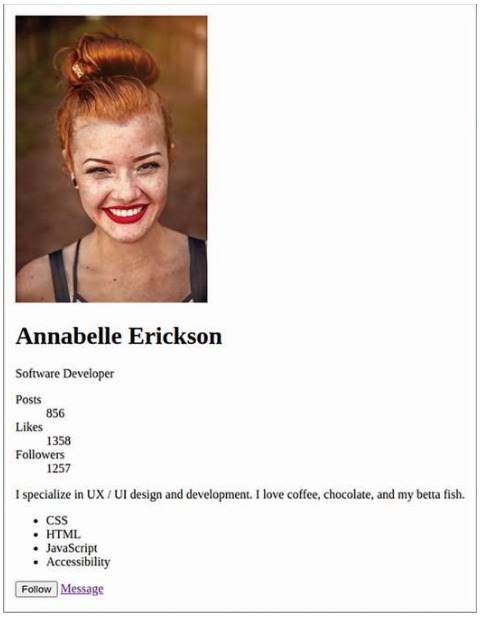

Figure 6.2 Starting point

Listing 6.2 Defining CSS custom properties

```
body {
  --primary: #de3c4b;                    ①
  --primary-contrast: white;
  --secondary: #717777;                  ②
  --font: Helvetica, Arial, sans-serif;
  --text-color: #2D3142;                 ③
  --card-background: #ffffff;
  --technologies-background: #ffdadd;
  --page-background: linear-gradient(#4F5D75, #2D3142);
  --imageSize: 200px;

  background: var(--page-background);
  font-family: var(--font);
  color: var(--text-color);
}
```

① Red

② Gray

③ Dark blue-gray

NOTE Our linear gradient will go from top to bottom, fading from dark blue to darker blue. For an in-depth explanation of linear gradients, check out chapter 3.

Notice that we can assign different types of values to our variables. We assign colors, such as in our `--primary` variable (probably one of the most common uses for CSS custom properties), but we also define a size (`--image-Size`), a font family (`--font`), and a gradient (`--page-background`).

To reference the variable and use it as part of a declaration, we use the syntax `var(--variableName)` . Therefore, to assign our text color, we declare `color: var (--text-color);` . With our background and the font color and family applied (figure 6.3), we notice that our background repeats at the bottom of the page.

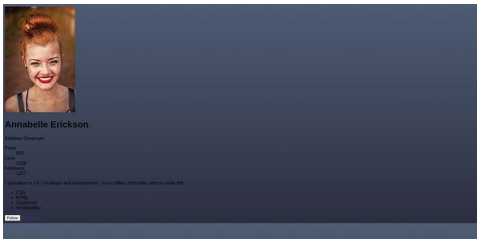

Figure 6.3 Adding the background to the `<body>`

.3 Creating full-height backgrounds

A *linear gradient* is a type of image. When we apply an image as a background to an element in CSS, if the image is smaller than the element, the image will repeat, or *tile*. In this particular case, we don't want the image to repeat. We have two ways to fix this situation:

- We can tell the background that we don't want it to repeat by using `background-repeat: no-repeat;`. Because our `<body>` element is only as tall as its contents, however, if the window is taller than the content, we'll be left with an unsightly white bar at the bottom of the page—which is not ideal.

- Our second option (the one we'll use) is to make the `<html>` and `<body>` elements take the full height of the screen rather than size to their contents.

We'll add the rule in listing 6.3 to our stylesheet. We reset the margin and padding to `0` because we want to ensure that we go edge to edge inside the window.

Listing 6.3 Making the background full height

```
html, body {
  margin: 0;
  padding: 0;
  min-height: 100vh;
}
```

To set the height, we use `min-height` because should the content length be greater than the height of the window, we want the user to have access to the content, and we want the background to be behind that content. By using `min-height` , we instruct the browser to make the `<body>` and `<html>` elements at least the height of the window. If the content forces the elements to be taller, the browser

will use the height of the content.

The value we set for `min-height` is `100vh`. Viewport height (`vh`), a unit based on the height of the viewport itself, is percentage-based. So assigning a value of `100vh` to `min-height` means that we want the element to have, at minimum, a height equal to 100% of the viewport height. Now that we have our background set (figure 6.4), let's style the card.

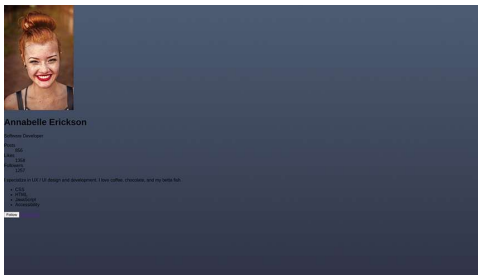

Figure 6.4 Full-screen gradient background

.4 Styling and centering the card using Flexbox

Let's start with styling the card itself. We'll give it a white background and shadow to give our layout some depth. Notice that instead of using the color value for the back-

ground, we use our `background` variable.

We're also going to set the width of the card to `75vw`. Viewport width (`vw`) is the horizontal counterpart to the viewport height (`vh`) unit we used earlier. It's also percentage based, so by setting our width to `75vw`, we're setting the width of the card to be 75% of the total width of the browser window.

Next, we'll further constrain the width of the card to a maximum 500 pixels wide. By using both the `width` and `max-width` properties, we allow the card to shrink when the screen size is narrow but constrain it from becoming too wide and unruly on larger screens. Last, we curve the corners of the card by using `border-radius` to soften the design. The following listing shows our card rule.

Listing 6.4 Styling the card

```
.card {
  background-color: var(--card-background);
  box-shadow: 0 0 55px rgba(38, 40, 45, .75);
  width: 75vw;
  max-width: 500px;
```

```
    border-radius: 4px;
  }
```

Figure 6.5 shows the styles applied to our project. With some basic styles added to the card (we'll continue adding to them later in the chapter), let's place the card in the middle of the screen both vertically and horizontally.

Figure 6.5 Starting to style the card

To center the card in the exact middle of the screen, we're going to use a flex layout (sometimes referred to as *flexbox*), which allows us to place elements across a single axis either vertically or horizontally. Although we could position the card by using `grid` (and whether we should is a matter of personal preference), in this instance, we're concerned only with centering the item, not with its position in terms of columns and rows, so Flexbox seems to be a better choice.

The `display` property with a value of `flex` is used on the parent item of the child elements that should be placed on the screen with Flexbox. In our project, the element being positioned is the card, and its parent is the `<body>` element, so we'll add the `display: flex` declaration to our `body` rule.

Next, we define how we want the elements within the `<body>` to behave. In our case, we have one child (the card), and we want it to be centered. To center the card horizontally, we add a `justify-content: center` declaration to the `body` rule. This property allows us to dictate how elements are distributed across our axis. Figure 6.6 breaks down the options.

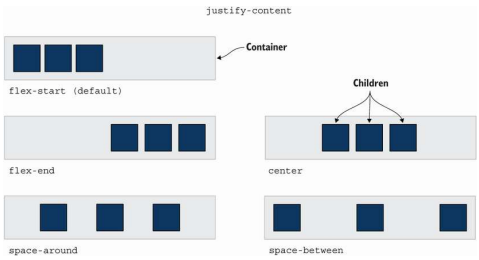

Figure 6.6 Values for the `justify-content` property

We also want to center the card vertically. For the vertical positioning, we'll use `align-`

`items: center`. The `align-items` property enables us to dictate how elements should be positioned relative to one another and to the container, as shown in figure 6.7.

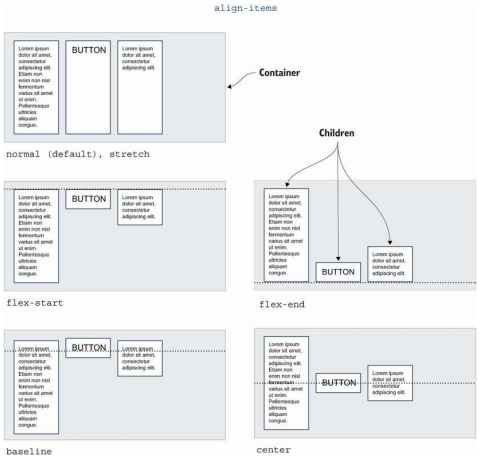

Figure 6.7 Values for the `align-items` property

The following listing shows our updated body rule. Remember that the parent of the element being positioned is the one to which we apply flexbox-related declarations.

Listing 6.5 Centering the card

```
body {
  ...
  display: flex;
  justify-content: center;     ①
  align-items: center;         ②
}
```

① Centers the card horizontally

② Centers the card vertically

Now that our card is centered (figure 6.8), let's focus on the content of the card, starting with the profile picture.

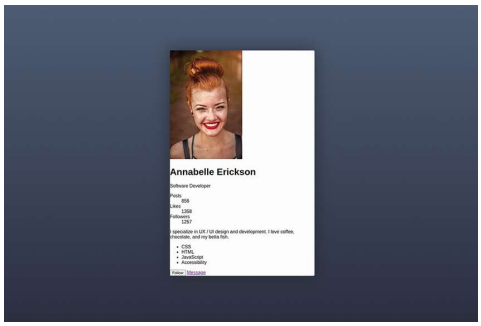

Figure 6.8 Centered card

.5 Styling and positioning the profile picture

We currently have a rectangular image. We want to make the image circular. We also want to center it on the card and have it stick out the top a little bit. Let's start by converting the image to a circle.

6.5.1 The object-fit property

A circle's height is equal to its width, so as we can see in figure 6.9, if we set the height and width of the picture to equal our image-size variable, the picture will distort.

Figure 6.9 Distorted profile picture

To prevent the image from distorting, we must also dictate how the image behaves in relation to the size it's given. To do this, we'll use the `object-fit` property. By setting `object-fit`'s value to `cover`, we instruct the image to maintain its initial aspect ratio but fit itself to fill the space available. In this case, we'll lose a little of the top and bottom of the image due to the image being taller than it is wide.

When we use `object-fit`, the image is centered by default, and if parts of the image are clipped, those parts are the edges, which works well for our current use case and picture. But if we wanted to adjust the position of the image within its allotted size and clip only from the bottom, we would add an `object-position` declaration.

To make our image a circle 200 pixels wide, we use the CSS in listing 6.6. Remember

that we set the image size as a CSS custom property in the `body`, so we set the width and height of the image equal to the `--imageSize` variable. We add the `object-fit` declaration to prevent the image from distorting. Finally, we give the image a 50% `border-radius` to make it a circle.

Listing 6.6 Centering the card

```
body {
  ...
  --imageSize: 200px;
}

img.portrait {
  width: var(--imageSize);
  height: var(--imageSize);
  object-fit: cover;          ①
  border-radius: 50%;         ②
}
```

① Prevents distortion

② Makes the image a circle

Now our image looks like figure 6.10.

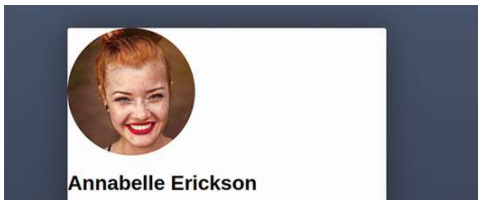

Figure 6.10 Circle profile picture

Next, we need to position our picture.

6.5.2 Negative margins

To position our image to stick out above the card, we're going to use a negative margin. To move an element down and away from the content above it, we can add a positive `margin-top` value to the element. But if we add a negative margin, instead of being pushed down, the element will be pulled up. We're going to use margin in conjunction with text centering to position the image. Looking at the final design in figure 6.11, we notice that all the text is also centered.

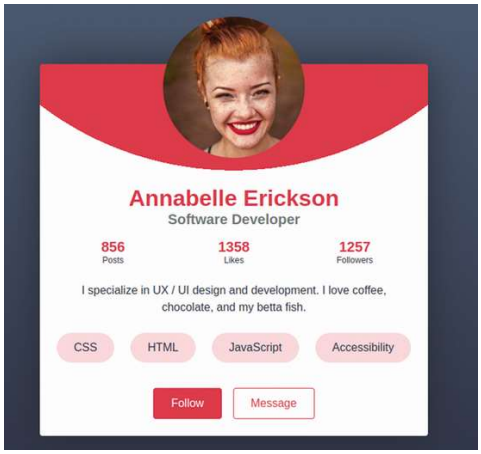

Figure 6.11 Final design

Because all the text is centered, let's add a `text-align: center` declaration to the

card rule. Images are inline elements by default, so we notice that by centering the text, the image also gets centered (figure 6.12).

Figure 6.12 Centered text

Now all that's left to do is add the negative top margin to move the image upward. We want one third of the image to stick out from the top, and we'll use the `calc()` function to do the math for us. Our function is `calc(-1 * var(--imageSize) / 3);`. We divide the image size by 3 to get a third of the height of the image and then multiply by –1 to make it negative. Our margin will make a third of the image stick out from the top of the card, as shown in figure 6.13.

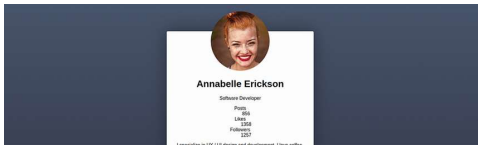

Figure 6.13 Positioned image

Next, we need to give our card some margin. Due to the nega-

tive margin we added to the image, if we have a short screen (figure 6.14), the top of the image disappears offscreen.

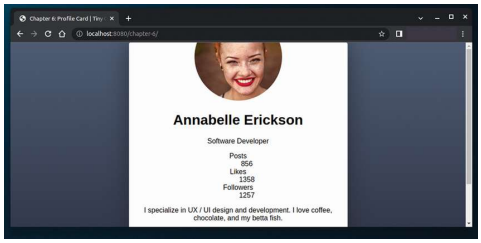

Figure 6.14 Clipping the top of the image when the window height is small

To prevent cutting off part of the picture when the window isn't especially tall, we want to add some vertical margin to the card itself—a margin that's greater than or equal to the amount of the picture that's sticking out of the card. To calculate the amount sticking out, we used `calc(-1 * var(--imageSize) / 3);`. For our card margin, we're going to use a similar concept, taking one third of the image height and then adding 24 pixels to move the card and image away from the edge. Our final function will be `calc(var(--imageSize) / 3 + 24px)`. The following listing shows the CSS we added to position the image.

Listing 6.7 Positioning the image

```
.card {
  ...
  text-align: center;
  margin: calc(var(--imageSize) / 3 + 24px) 24px;    ①
}

img {
  width: var(--imageSize);
  height: var(--imageSize);
  object-fit: cover;
  border-radius: 50%;
  margin-top: calc(-1 * var(--imageSize) / 3);       ②
}
```

① Vertical margin of one third the image size + 24px and horizontal of 24px

② Negative top margin to make the image stick out of the card

With our image positioned and margins added so that the top of the image doesn't get cut off on small screens (figure 6.15), let's turn our attention to the curved red background below the picture.

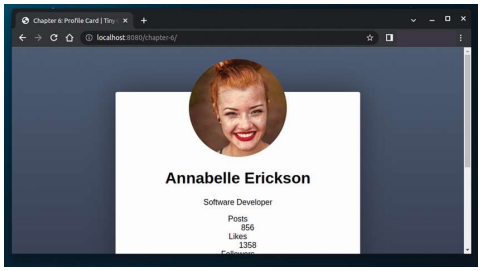

Figure 6.15 Added card margin

.6 Setting the background size and position

To add the red curved background behind the picture, we're going to add the declaration in the following listing to our card rule.

Listing 6.8 Positioning the image

```
.card {
  background-color: var(--card-background);
  ...
  background-image: radial-gradient(
    circle at top,
    var(--primary) 50%,
    transparent 50%,
    transparent
  );
  background-size: 1500px 500px;
  background-position: center -300px;
  background-repeat: no-repeat;
}
```

Let's break down what this code does. First, we add a `background-image` consisting

of a `radial-gradient`, as shown in figure 6.16.

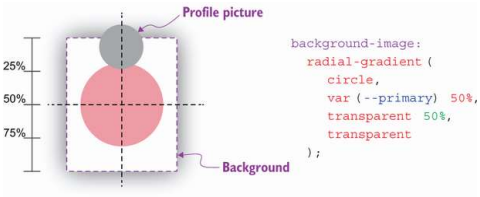

Figure 6.16 Adding background with `radial-gradient`

Combining background color and image

We can add both a background color and a background image to the same element. We assign the color to the `background-color` property and the image to the `background-image` property. Or we can apply both in the `background` shorthand property as follows: `background: white url(path-to-image);`.

The `radial-gradient` takes an ending shape (circle or ellipse) and then defines where we want each color to start and stop to form the gradient. We define ours as `radial-gradient(circle, var(--primary) 50%, transparent 50%, transparent);`.

Our primary color is red, so our gradient will create a cir-

cle that's red until it reaches 50% of its container. At 50% of the container size, the color immediately shifts to transparent. Because the shift in color is immediate, no fade occurs, so we get a nice clean circle.

By default, radial gradients emanate from the center of their container, so next we add `circle at top` to the beginning of our `radial-gradient` function to shift the origin of the circle from the center of the background to the top. Our updated `radial-gradient` function is `radial-gradient(circle at top, var(--primary) 50%, transparent 50%, transparent);` (figure 6.17).

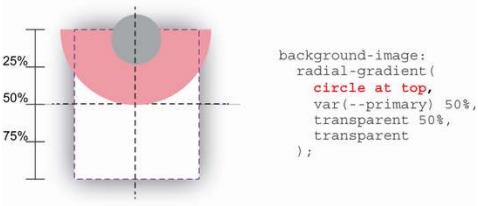

Figure 6.17 Making the gradient emanate from the top center of the container

Now we want to move the circle up so that the bottom of the circle is directly below the image. Figure 6.18 shows that if we move the background up -150 pixels and our card is rather short (our profile

doesn't have a lot of content), we'll end up with gaps in the top corners between our circle and the edge of the card, which we don't want.

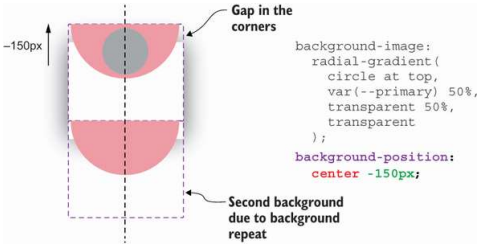

Figure 6.18 Altering the background position

To prevent this from happening, we're going to make the background image three times wider than the maximum card size: `(3×500 =1500)`. When we create a `background-image` using gradients, the background image produced will grow and shrink with the container, so we're also going to give the background a set height. That way, no matter how much content is in the card, the shape of our background will be predictable (figure 6.19).

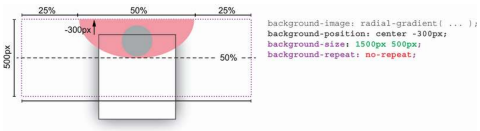

Figure 6.19 Editing the `background-size` and handling the `background-repeat`

After changing the dimensions of the background, we also increase the amount by which we move the background up so that it ends directly below the profile image. Finally, as mentioned earlier in the chapter, background images repeat by default. By moving the image up, we leave room for the background to tile. We want to have only one semicircle, so we add a `background-repeat` declaration with a value of `no-repeat`. Now our card background is defined as shown in the following listing.

Listing 6.9 Positioning the image

```
.card {
  background-color: var(--card-background);
  ...
  background-image: radial-gradient(     ①
    circle at top,                       ①
    var(--primary) 50%,                  ①
    transparent 50%,                     ①
    transparent                          ①
  );                                     ①
  background-size: 1500px 500px;         ②
  background-position: center -300px;    ③
  background-repeat: no-repeat;          ④
}
```

① Creates a semicircle whose flat side is the top of the card

② Sets the dimensions of the background image to 1500px wide and 500px tall

③ Positions the background to be horizontally centered and starting 300px above the card

④ Prevents the background from tiling

Figure 6.20 shows the background added to the card. With the top of our card starting to look good, let's focus on the rest of the content.

Figure 6.20 Finished background image

.7 Styling the content

Our card currently doesn't have any padding, which means that if the name were longer, it could potentially go edge to edge on our card. In most cases, we would create a card as a component or template to reuse for multiple clients, so let's add some left and right padding to ensure that our text doesn't run to the

edge of the card. We'll also add some bottom padding to move the links and bottom away from the bottom edge of the card.

Listing 6.10 shows our updated card rule, and figure 6.21 shows the new output. We use the padding shorthand property, which defines three values: it states that the top padding is `0`, that the left and right are `24px`, and that the bottom padding is `24px`. We specifically don't add padding to the top because it would push the image down, forcing us to readjust our image positioning.

Listing 6.10 Adding padding to the card

```
.card {
  ...
  padding: 0 24px 24px;
}
```

Figure 6.21 Added card padding

6.7.1 Name and job title

Going down the card, we see that the first piece of content is the name. As an `<h1>`, it has some default styles provided by the browser, including some margin. We're going to edit the margin to increase the amount of room between the header and the image, and remove the bottom margin so that the job title appears directly below the name. We'll also change the color to red and set the font size to `2rem`.

The rem unit

A *rem* is a relative unit based on the font size of the root element—in our case, HTML. For most browsers, the default is `16px`. We didn't set a font size on the `html` element in our project; therefore, when we set the `<h1> font-size` to `2rem`, the output size is `32px`, assuming a `16px` default.

The benefit of using relative font sizes such as `rem` and `em` is accessibility. These sizes help ensure that the text scales gracefully regardless of the user's settings or device.

For the job title, we'll increase the size and weight of the font, and we'll change the font color to our secondary color, which is gray. The following listing shows our new rules, and figure 6.22 shows the output.

Listing 6.11 Styling the name

```
h1 {                          ①
  font-size: 2rem;            ①
  margin: 36px 0 0;           ①
  color: var(--primary);      ①
}                             ①

.title {                      ②
  font-size: 1.25rem;         ②
  font-weight: bold;          ②
  color: var(--secondary);    ②
}                             ②
```

① Styles for the name

② Styles for the job title

Figure 6.22 Styled name and job title

Next, we're going to style the post, like, and follower information.

6.7.2 The space-around and gap properties

In our HTML, the description list (`dl`) contains the post, like, and follower counts (listing 6.12). Each grouping is contained within a `<div>` , so we'll apply a `display` value of `flex` to the definition list to align all three groups horizontally. Then we'll set the `justify-content` property to `space-around` to spread them out across the card.

Listing 6.12 Description-list HTML

```
<dl>
  <div>
    <dt>Posts</dt>
    <dd>856</dd>
  </div>
  <div>
    <dt>Likes</dt>
    <dd>1358</dd>
  </div>
  <div>
    <dt>Followers</dt>
    <dd>1257</dd>
  </div>
</dl>
```

The `space-around` value distributes the elements evenly across our axis by providing an equal amount of space between each element and half

as much on each edge. Figure 6.23 shows how the spacing is applied.

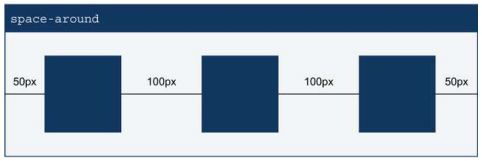

Figure 6.23 The `space-around` property

Listing 6.13 shows our styles for the description list. Notice that we included a `gap: 12px` declaration, which ensures that the minimum amount of space between our elements will be 12 pixels. We could have given our `<div>` s inside the description list a margin, but a margin would have affected the outer edges. The `gap` property affects only the space between elements.

NOTE The `gap` property is supported in iOS version 14.5 and later. At this writing, many people still use earlier versions. To check global use of this property, see **https://caniuse.com/flexbox-gap**.

Listing 6.13 Styling the name

```
dl {
  display: flex;
```

```
    justify-content: space-around;
    gap: 12px;
  }
```

As shown in figure 6.24, now our profile stats are in a row and evenly spaced across the card.

Figure 6.24 Aligned profile stats

The numbers are offset, however. This offset comes from the description, which has some margins that come from the browser defaults. Let's get rid of those settings and style the text to be bold, bigger, and red, using the CSS in the following listing.

Listing 6.14 Description details rule

```
  dd {
    margin: 0;
    font-size: 1.25rem;
    font-weight: bold;
    color: var(--primary);
  }
```

With the margin removed (figure 6.25), we notice that the

likes still aren't centered on our card.

Figure 6.25 Description list alignment

The reason that the likes aren't centered is that three elements don't have exactly the same width. When the elements are distributed, the browser calculates the total amount of space each element needs and redistributes the leftovers equally. Therefore, because the `<div>` containing followers is larger than the `<div>` containing posts, the likes `<div>` doesn't land in the middle.

6.7.3 The flex-basis and flex-shrink properties

To center the likes, we'll assign the same width to all three `<div>`s. Instead of using the `width` property, however, we'll use `flex-basis` and set its value to `33%`. `flex-basis` sets the initial size the browser should use when calculating the amount of space

the element needs. We'll also set `flex-shrink` to `1`.

`flex-shrink` dictates whether an element is allowed to shrink smaller than the size assigned by the `flex-basis` value if there's not enough room for the element in the container. If the `flex-shrink` value is `0`, the size isn't adjusted. Any positive value allows for resizing.

We set our `flex-basis` to `33%`. But remember that we also set a `gap` of 12 pixels between each of our elements. Therefore, the `flex-basis` size we set is too wide for the container when the `gap` setting is taken into consideration. By allowing the elements to shrink, we tell the browser to start its positioning calculations with each `<div>` taking up 33% of the width of the container and to shrink the `<div>`s evenly to fit the available space. This situation prevents us from having to do math, figuring out exactly how wide the `<div>`s should be and still be of equal sizes.

To write our rule (listing 6.15), we target the `<div>`s that are

immediate children of the description list (`dl`) by using a child combinator (`>`), and we apply the `flex-basis` and `flex-shrink` declarations.

Listing 6.15 Centering the likes

```
dl > div {
  flex-basis: 33%;
  flex-shrink: 1;
}
```

With our likes centered (figure 6.26), let's turn our attention to the definition terms (`dt`).

Figure 6.26 Centered likes

6.7.4 The flex-direction property

In our original design, we have the description details (the numbers) above the description terms. To flip them visually, we're going to use the `flex-direction` property. We asserted that Flexbox can place elements across a single axis. So far, we've done our work across the horizontal axis, or x-axis.

To move the details above the terms, we're going to use Flexbox on the vertical (y-axis), sometimes called the *block* or *cross* axis. To change which axis we want Flexbox to operate on, we use the `flex-direction` property. By default, that property has a value of `row`, which makes Flexbox operate on the x-axis. By changing the value to `column`, we make it operate on the y-axis.

Furthermore, the `flex-direction` property allows us to dictate how the elements should be ordered. Setting the value to `column-reverse` tells the browser that we want to operate on the y-axis and that we want the elements to be placed in reverse HTML order, making the description details (`<dd>`) appear first and the description term (`<dt>`) second.

As before, we want to set the behavior on the parent—in this case, the `<div>`. We'll add to our previous `<div>` rule to reorder the elements (listing 6.16). We also decrease the size of the description term

(`<dt>`) to emphasize the number over its term.

Listing 6.16 Reversing content order

```
dl > div {
  flex-basis: 33%;
  flex-shrink: 1;
  display: flex;
  flex-direction: column-reverse;
}
dt { font-size: .75rem; }
```

Accessibility concerns and content display order

For accessibility reasons, we want to make sure that the order in which our HTML is written follows the order in which it's displayed onscreen. A user who has their computer read the contents of the page to them as they follow along visually would be easily disoriented or confused if the content that's being read to them doesn't match what they're seeing. Use caution when using properties such as `flex-direction` to reorder content.

Figure 6.27 shows our styled description list (`<dl>`).

Figure 6.27 Styled description list

Continuing down the card, let's turn our attention to the summary paragraph below the profile stats.

6.7.5 Paragraph

The paragraph already looks good. The only thing we're going to do to it is add some vertical margin for breathing room and increase the line height for better legibility, as shown in listing 6.17.

Notice that the line height doesn't take a unit. By not setting a unit, we allow the line height to scale with the font size. This unitless value is specific to the `line-height` property. If we'd set it to a `12px` value, for example, the line height would remain 12 pixels regardless of the font size. So if the font size were increased radically, our letters would overlap vertically. It's always safest *not* to declare a unit.

Listing 6.17 Paragraph rule

```
p.summary {
  margin: 24px 0;
  line-height: 1.5;
}
```

With our paragraph taken care of (figure 6.28), let's style the list of technologies.

Figure 6.28 Styled summary paragraph

6.7.6 The flex-wrap property

First, we're going to style the list elements themselves. We'll use a design pattern sometimes referred to as a *pill*, *chip*, or *tags*, in which the element has a background color and rounded edges. Our CSS will look like listing 6.18. We also include some padding so that the text doesn't come right up against the edge of the tag.

Listing 6.18 Styling the list elements

```
ul.technologies li {
  padding: 12px 24px;
  border-radius: 24px;
  background: var(--technologies-background);
}
```

With the individual elements styled (figure 6.29), we can focus on the list's layout.

Figure 6.29 Styled list items

First, we'll remove the bullets by using `list-style: none`. Then we'll remove all padding, and set the margins to `24px` vertically and `0` horizontally.

To position the items, we'll use Flexbox, adding a `gap` of `12px` and setting the `justify-content` property value to `space-between`. `space-between` works similarly to `space-around` except that it doesn't add space to the beginning and end of the container, as shown in figure 6.30.

Figure 6.30 Comparing `space-around` and `space-between`

Our rule to lay out our chips will look like the next listing.

Listing 6.19 Styling the list of technologies

```
ul.technologies {
  list-style: none;
  padding: 0;
  margin: 24px 0;
  display: flex;
  justify-content: space-between;
  gap: 12px;
}
```

We notice that when we reduce the screen width (figure 6.31), however, our last tag extends beyond our card.

Figure 6.31 Tag extending beyond card width

On narrow screens, our list is wider than our card. To prevent the content from overflowing the card, we can use the `flex-wrap` property.

By default, flex items display in a straight line even if the container is too small, as we're experiencing with our list of technologies. To force the last element onto a new line when we run out of room, we can set the `flex-wrap` property to `wrap`. This setting tells the browser to start a new line of

items below when it runs out of room.

Like `flex-direction`, `flex-wrap` can change the order in which the elements are displayed, but we won't need to change it here. The following listing contains our updated rule.

Listing 6.20 Adding `flex-wrap`

```
ul.technologies {
  list-style: none;
  padding: 0;
  margin: 24px 0;
  display: flex;
  justify-content: space-between;
  gap: 12px;
  flex-wrap: wrap;
}
```

Notice the gap between the CSS and Accessibility tags in figure 6.32, even though our list element doesn't have any margin. Our list has a `gap` property value of `12px`, which means not only that we'll have a minimum 12 pixels horizontally between our items, but also that when we wrap, we'll add a 12-pixel gap between the items vertically.

Figure 6.32 Wrapping the chips on narrow screens

.8 Styling the actions

The last things we need to style in our profile card are the two actions the user can take at the bottom of the card: message or follow the profile owner. Even though these actions are semantically different—one is a link, and the other is a button—we're going to style both of them to look like buttons. Let's start with some basics that will apply to both elements. We create one rule with selectors for both elements to ensure that both element types are visually consistent. Then we create individual rules for use where they diverge.

We also create a `focus-visible` rule that will be applied to all elements by means of the universal selector (`*`) and the pseudo-class `:focus-visible` so that when a user navigates to our links and buttons

via the keyboard, a dotted outline appears around the element, and they can clearly see what they're about to select. The following listing shows our styles.

Listing 6.21 Adding `flex-wrap`

```
.actions a, .actions button {          ①
  padding: 12px 24px;
  border-radius: 4px;
  text-decoration: none;               ②
  border: solid 1px var(--primary);
  font-size: 1rem;
  cursor: pointer;
}

.follow {
  background: var(--primary);
  color: var(--primary-contrast);
}

.message {
  background: var(--primary-contrast);
  color: var(--primary);
}

*:focus-visible {
  outline: dotted 1px var(--primary);
  outline-offset: 3px;
}
```

① Applies to both the link and the button

② Removes the underline

Notice that in our base styles, we changed the `cursor` to `pointer` for both links and buttons. In most browsers, links will use the `pointer` by default but not the `button`. Because we want both elements to have a similar experience, we'll define the `cursor` to ensure consistency. Figure 6.33 shows our styled link and button.

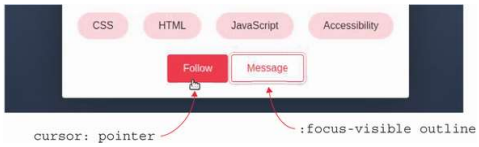

Figure 6.33 Styled actions

As these two buttons are quite close together, however, we're going to want to add some space between them. Let's use `flex` and `gap` one last time to position our action elements.

We're going to give the list a `display` property value of `flex` and add a `gap` of `16px`. To keep the two elements centered, we'll use the `justify-content` property with a value of `center`. Finally, we'll add some space between the list of technologies and our actions by giving the list a `margin-`

`top` value of `36px`, as shown in the following listing.

Listing 6.22 Positioning the link and button

```
.actions {
  display: flex;
  gap: 16px;
  justify-content: center;
  margin-top: 36px;
}
```

With this last rule, we've finished styling our profile card. The final product is shown in figure 6.34.

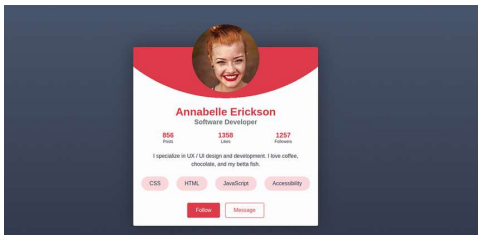

Figure 6.34 Finished profile card

ummary

- CSS custom properties allow us to set variables that can be reused throughout our CSS.
- The CSS Flexbox Layout Module allows us to position elements on a single axis either horizontally or vertically.

- `flex-direction` sets which axis Flexbox will operate on.
- Both `flex-direction` and `flex-wrap` can alter the order in which the elements are displayed.
- The `align-items` property sets how the elements are aligned on the axis relative to one another.
- The `justify-content` property dictates how the elements are positioned; leftover space will be distributed within the element to which it's applied.
- `flex-basis` sets a starting element size for the browser to use when laying out flexed content.
- `flex-shrink` dictates whether and how content can shrink when an element is being flexed.
- We can prevent images from distorting when we use fixed heights and widths that don't match the image's aspect ratio by using the `object-fit` property.