11 Animated social media share links

This chapter covers

- Using the OOCSS, SMACSS, and BEM architecture patterns
- Scoping CSS when working with components
- Working with social media icons
- Creating CSS transitions
- Using JavaScript to overcome CSS limitations

One of the core reasons why the internet was created was to share and distribute information. One way we do this today is through social media. In this chapter, we'll style and animate some links that can be used to share a web page via email or social media.

As in the previous chapters, we'll be using HTML and CSS for this project without any frameworks. We chose this ap-

proach to focus on the CSS itself without the complexity and intricacies of using external packages. But many applications in the wild do use frameworks, some of which include the concept of the component.

A common reason to turn a piece of functionality into a component is to reuse the piece of code or element in multiple places in applications. With reusability comes the possibility of naming collisions. Some systems automatically restrict the scope of the CSS of the component to itself, preventing any possible collision between component styles. But many systems don't restrict the scope, leaving it up to the developer to organize the code to prevent changing the styles in another component when styling a new one.

Regardless of the framework and how it does (or doesn't) handle CSS scoping, we have a variety of architecture options to help us organize and standardize our styles. Before we dive into this chapter's project, let's take a quick look at some CSS architecture options.

1.1 Working with CSS architecture

Some of the most popular CSS architecture methodologies are OOCSS, SMACSS, and BEM. We'll be using BEM in this chapter, but we'll take a look at all three options so that we'll understand the high-level differences among them.

11.1.1 OOCSS

Introduced at Web Directions North in Denver by Nicolle Sullivan, OOCSS (Object-Oriented CSS; https://github.com/stubbornella/oocss/wiki) aims to help developers create CSS that's fast, maintainable, and standards-based. Sullivan describes the *Object* part of OOCSS as "a repeating visual pattern, that can be abstracted into an independent snippet of HTML, CSS, and possibly JavaScript. That object can then be reused throughout a site"—in other words, what we might think of today as a component or widget. To achieve this reusability, OOCSS follows two main principles:

- *Separate structure and skin*—Keeps visual features (background, borders, and so on, sometimes referred to as the theme) in their own classes, which can be mixed and matched with objects to create a variety of elements.
- *Separate container and content*—By refraining from using location-dependent styles, we can ensure that the objects look the same no matter where they're placed in the application or on the website.

11.1.2 SMACSS

Developed by Jonathan Snook, SMACSS (Scalable and Modular Architecture for CSS; **http://smacss.com**), organizes CSS rules into five categories:

- *Base*—The defaults applied by using element, descendent, or child selectors and pseudo-classes
- *Layout*—Used to lay elements out on the page, such as headers, articles, and footers
- *Module*—More discrete parts of the layout, such as carousels, cards, and navigation bars

- *State*—Something that augments or overrides other styles, such as an error state or the state of a menu (open or closed)
- *Theme*—Defines the look and feel; doesn't have to be separated in its own classes if it's the only theme for the page or project

11.1.3 BEM

Developed by a company named Yandex, BEM (Block Element Modifier; **https://en.bem.info/methodology**) is a component-based architecture that aims to break the user interface into independent, reusable blocks:

- *Block*
  - Describes the block's purpose.
  - An example would be a class name for an element, such as `header`.
- *Element*
  - Describes the element's purpose.
  - The class name is the block name followed by two underscores and the element, such as `header__text`.

- *Modifier*
  - Describes the appearance, state, and behavior.
  - The class pattern is `block-name_modifier-name` (example: `header_mobile`) or `block-name__element-name_modifier-name` (example: `header__menu_open`).

Choosing an architectural approach for CSS is a team-dependent task. The needs of the project, the size and experience of the team, and the libraries and frameworks being used are factors to consider. No one-size-fits-all approach exists, so the decision needs to be made by the team. Because of BEM's component-based nature, we'll use it in this chapter to scope and style our social media share links.

1.2 Setting up

Now that we've chosen our methodology, which dictates the naming convention we'll use for the project, let's take a look at what we'll be building. We'll style a Share button that, when clicked, opens a set of links that let the user share the page via email or to Facebook, LinkedIn,

or Twitter. Then we'll use transitions to animate opening and closing the share options and the hover/focus effects of the individual links. Figure 11.1 shows our goal.

Figure 11.1 Goal

Our starting HTML (listing 11.1) consists of a container for our component, a Share button, and a menu that lets users choose how to share the page. The code includes a linked JavaScript file, which makes our component usable via keyboard navigation and triggers showing/hiding the links inside the component when the Share button is clicked. As we'll see in section 11.6, a few limitations apply to animating elements with CSS alone, so we'll rely on a couple of lines of JavaScript to support our CSS. We'll look at JavaScript in more detail later in the chapter (also in section 11.6); first, we'll focus on our HTML and CSS.

Listing 11.1 Starting HTML

```
<main>
  <div class="share" id="share">                    ①
```

```
    <button id="shareButton"                                                    ②
      class="share__button"                                                     ②
      type="button"                                                             ②
      aria-controls="mediaList"                                                 ②
      aria-expanded="false"                                                     ②
      aria-haspopup="listbox">                                                  ②
        <img src="./icons/share.svg" alt="" width="24" height="24">             ②
        Share                                                                   ②
    </button>                                                                   ②

    <menu aria-labelledby="share"                                               ③
          role="menu"                                                           ③
          id="mediaList"                                                        ③
          class="share__menu">                                                  ③

      <li role="menuitem" class="share__menu-item">                             ④
        <a href="mailto:?subject=Tiny%20..."                                    ⑤
          target="_blank"                                                       ⑤
          rel="nofollow noopener"                                               ⑤
          tabindex="-1"                                                         ⑤
          class="share__link"                                                   ⑤
        >
          <img src="./icons/email.svg"                                          ⑥
               alt="Email" width="24" height="24">                              ⑥
        </a>
      </li>
      <li role="menuitem" class="share__menu-item">
        <a href="https://www.facebook.com/sh..."                                ⑦
          target="_blank"                                                       ⑦
          rel="nofollow noopener"                                               ⑦
          tabindex="-1"                                                         ⑦
          class="share__link"                                                   ⑦
        >                                                                       ⑦
          <img src="./icons/facebook.svg"                                       ⑦
               alt="Facebook" width="24" height="24">                           ⑦
        </a>                                                                    ⑦
      </li>
      ...
     </menu>
  </div>
</main>
```

```
    <script src="./scripts.js"></script>                    ⑧
```

① Component container

② Share button to open and close the list of social media links

③ Media menu

④ Menu item

⑤ First link is a mailto to share via email rather than social media.

⑥ Media icon

⑦ Link to share via social media

⑧ Script used for keyboard interactions and supplementing CSS

We also have some basic starter CSS applied to the `main` element to move the component away from the edge of the screen: `main { margin: 48px; }`.

You can find all the starter code (HTML, CSS, and JavaScript) on GitHub at **http://mng.bz/KeR4** or CodePen at **https://codepen.io/michaelgearon/pen/YzZzpWj**.

Our starting point looks like figure 11.2.

Figure 11.2 Starting point

As you can see, the icons have been provided, but let's discuss where and how we got them.

1.3 Sourcing icons

Any time we use iconography from someone else's brand, we need to answer the following questions:

- Are we authorized to use the icon?
- Are there any restrictions on how the icon can be used?

When we use social media icons, those brands are being represented in our work, so we must follow their guidelines on when, how, and in what context we can use the brand. When we use icons that don't represent a brand (such as the icons we used for the `mailto` link and

Share button), unless we created the icon ourselves, we're subject to copyright laws, just as we would be for any other piece of media (image, sound, video, and so on) that we use in our projects.

NOTE We're not lawyers, and we don't intend to offer legal advice in this chapter. When in doubt, contact a legal professional.

11.3.1 Media icons

An effective way to find how a branded icon can be used is to look for that brand's guide by doing a web search for terms such as *style guide* and *brand guide*. Many social media outlets have specific instructions on how the brand can be represented, including icon and logo downloads. Table 11.1 lists the social media platforms we included in our component and the links to their brand information. For this project, we sourced our social media icons directly from the respective brand guides.

Table 11.1 Social media brand resources

Brand	Icon	Link to assets
Facebook		http://mng.bz/9Dza
LinkedIn		https://brand.linkedin.com/downloads
Twitter		http://mng.bz/jPry

11.3.2 Icon libraries

Looking for icons can be a bit tedious, especially in large projects, so it's common practice to use icon fonts and libraries, which also are subject to terms of use. Each library and icon font has its own rules about where and how icons can be used. Some also require attribution. Therefore, we must be aware of any rules we need to follow while sourcing our icons.

For this project, we sourced our non-brand-related icons from Material Symbols (https://fonts.google.com/icons). Because we needed only two—share  and email @—we downloaded the individual SVGs and included them in our

icon folder rather than importing the entire library into the project. The icons have been provided in the starter code, so we're ready to start styling.

1.4 Styling the block

Because we're using BEM for our naming convention, our block name will be `"share"`. Therefore, the container `<div>` that wraps the entire component will have a class of `share`. This block name will be included in all future classes that use the BEM naming convention (section 11.1.3), which scopes our CSS to that component and helps prevent any styling collisions between our component and any other parts of the application it may be used in.

As shown in listing 11.2, we define the `font-family`, `background`, and `border-radius` for the block. We also give the component a `display` value of `inline-flex`. `inline-flex` works the same way as `flex` but makes the element an inline-level element rather than a block-level element. By making our component behave like an inline element (the same as links, spans, buttons, and so on),

we give it the greatest versatility in terms of placement in an application. Furthermore, buttons are inline elements by default, and when closed, what's presented is essentially a button, so we'll give our component the same flow behavior as a button.

NOTE To find out how Flexbox works and discover its associated properties, check out chapter 6.

Listing 11.2 Styling the container

```
.share {
  font-family: Verdana, Geneva, Tahoma, sans-serif;
  background: #ffe46a;          ①
  border-radius: 36px;
  display: inline-flex;
}
```

① Yellow

With the block styled (figure 11.3), let's address the individual elements inside the block.

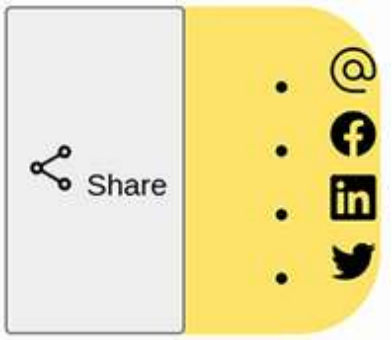

Figure 11.3 Styled container block

1.5 Styling the elements

Our block has three descendent elements, all of which we want to style:

- The Share button
- The menu containing the list of links
- The individual links inside the menu

Let's start with the Share button and work our way down the list.

11.5.1 Share button

The class name given to the button will include the block name followed by two underscores and then the element. In our case, we'll call this element `button`, so our class name will be `share__button`. By prefixing our class name with `share__`, we ensure that the only button we'll be styling is the one within our block.

We want to override the defaults provided by the browser and align the icon and text within the button (listing 11.3). We remove the background and border, adjust the font size and padding, and curve our corners.

To align the icon and text, we give the button a `display` value of `flex` and then use `align-items` to align the icon and text vertically. To add whitespace between the icon and text, we use the `gap` property.

Listing 11.3 Styling the Share button

```
.share__button {
  background: none;
  border: none;
  font-size: 1rem;
  padding: 0 2rem 0 1.5rem;
  border-radius: 36px;
  display: flex;
  align-items: center;
  gap: 1ch;
}
```

Figure 11.4 shows our output.

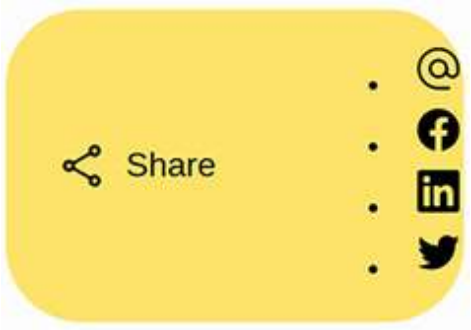

Figure 11.4 Styled Share button

Next, let's handle the hover and focus styles. We use the `:hover` and `:focus-visible` pseudo-classes to change the cursor style conditionally and add a black outline to the button. Then we offset the outline by `-5px` so that the outline places itself 5-pixels inside the button rather than on the outer edge.

The `outline-offset` property allows us to control where the outline is placed. Positive numbers move the outline farther out or away from the element; negative numbers inset the outline. The following listing shows our hover and focus CSS.

Listing 11.4 Share button hover and focus CSS

```
.share__button:hover,
.share__button:focus-visible {
  cursor: pointer;
  outline: solid 1px black;
  outline-offset: -5px;
}
```

Figure 11.5 shows our button being hovered over with a mouse.

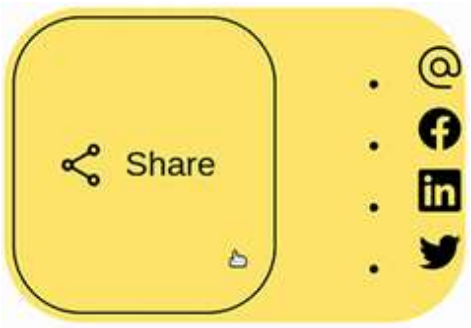

Figure 11.5 Share-button hover

11.5.2 Share menu

To style the menu and its items, we want to remove the bullets and then place the elements in a row beside the Share button. To remove the bullets, we give the list items a `list-style` value of `none` . Then we give the menu a `display` property value of `flex` . Finally, we remove the default margin and padding that the browser applies to the menu item automatically. The following listing shows our CSS.

Listing 11.5 Share menu and menu items

```
.share__menu-item { list-style: none; }

.share__menu {
  display: flex;
  margin: 0;
```

```
    padding: 0;
  }
```

When we look at our output (figure 11.6), we notice that we need some space between the edge of our container and our elements. We'll handle this task while styling the individual links.

Figure 11.6 Styled menu

11.5.3 Share links

To make sure that the links have a circle border on hover (rather than an ellipse), we set both their `height` and `width` to 48 pixels. Next, we curve their corners. This step also resolves our spacing problem because, as we see in listing 11.6, we've set the icon `height` and `width to 24`. Because we're making the links 48 pixels in both height and width, when the links are centered, we'll have 12 pixels of whitespace between each icon and the edge of its link.

Listing 11.6 List Item HTML

```
<li role="menuitem" class="share__menu-item">
  <a href="https:/ /www.facebook.com/sha..."
    target="_blank"
    rel="nofollow noopener"
    tabindex="-1"
    class="share__link"
  >
    <img src="./icons/facebook.svg" alt="Facebook" width="24" height="24">
  </a>
</li>
```

We also give the links a transparent border. Borders take up space, so to prevent the content from shifting on hover or focus when we expose the border, we add a transparent border by default and then color it when we want to show it. This approach ensures that the space needed for the border is allotted and prevents the content succeeding the element from shifting when the border is exposed.

To center the icon in the middle of the circle, we use `flex`, justifying the content and aligning the items to the center. Our CSS looks like the following listing.

Listing 11.7 Styling the links

```
.share__link:link,
.share__link:visited {
  height: 48px;
  width: 48px;
  border-radius: 50%;
  display: flex;
```

```
    align-items: center;
    justify-content: center;
    border: solid 1px transparent;
  }
```

With our links styled (figure 11.7), we can style the links for the hover and focus states.

Figure 11.7 Styled share links

11.5.4 scale()

On hover and focus, we're going to expose the border by changing its color from transparent to black. When we set the border on the links, we used the `border` shorthand property, which allows us to define the style, border width, and border color in one declaration. Because we're changing only the color, we'll use `border-color` rather than the `border` shorthand. By using `border-color`, we can edit the border's color without worrying about the rest of the already defined properties.

Next, we'll use the `scale()` function to increase the size of the icon to make it look as though it's magnified. In chapter 2, while expanding the loader bars, we used `scaleY()`

to grow and shrink the bar vertically. In this project, we want our links to grow proportionally, so we'll use `scale()`. When passed a single parameter, this function grows the element (both horizontally and vertically) proportionally by the same amount.

The `scale()` function is the shorthand for combining `scaleX()` and `scaleY()`. If only one value is passed, the `scale()` amount is applied both vertically and horizontally. If two parameters are passed, the first parameter defines horizontal scale, and the second defines vertical scale.

On hover or focus, we want the links to be 25% larger than when they're not being interacted with, so we'll give our function a single parameter of `1.25` and apply it to the `transform` property. Our CSS looks like the following listing.

Listing 11.8 Styling the links on hover and focus

```
.share__link:hover,
.share__link:focus-visible {
  border-color: black;
  outline: none;
```

```
    transform: scale(1.25);
  }
```

With the styles applied, our links grow on hover (figure 11.8), but because now the link is taller than the container, gaps at the top and bottom of the link don't have the yellow background.

Figure 11.8 Link on hover

To create our magnification effect, we want the entire link to remain yellow. We could add a yellow background to the link, which would accomplish that task, but the background needs to be yellow because the block's background color is yellow. If we changed the background color of the container, we'd want the link's background color to change as well. To make sure that the colors stay in sync, we could use a custom property (CSS variable) or make the element inherit the color from its parent.

11.5.5 The inherit property value

The `background-color` property isn't inherited by default.

We want to explicitly instruct the link to inherit the background color. To this inheritance from its parent, we can set the `background-color` property value for the link to `inherit`. Inheritance, however, goes up only to the parent. In our case, the element that controls the background color is the link's great-grandparent, as shown in figure 11.9.

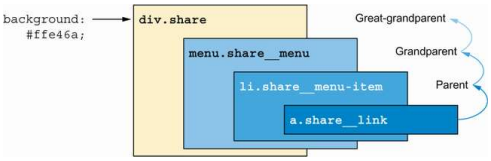

Figure 11.9 Ancestors of the media links

We need to make the `link`, `menu`, and `menu-item` rules inherit the `background-color` to make it trickle down to the link. After we give all three elements a `background-color` value of `inherit` (figure 11.10), we notice that although we've fixed the gaps in the link being hovered over, we've lost the curve on the right side of the component.

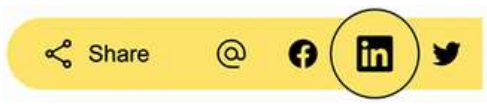

Figure 11.10 Inherited `background-color`

We lost our curve because, like `background-color`, `border-radius` isn't inherited. To fix the problem, we apply the same logic that we used for `background-color`. Listing 11.9 shows our edited CSS. Notice that the `border-radius` of the link wasn't edited. We want to keep the link's shape as a circle, so we keep the `border-radius: 50%` declaration on the link.

Listing 11.9 Inheriting property values

```
.share__menu-item {
  list-style: none;
  background: inherit;
  border-radius: inherit;
}

.share__menu {
  display: flex;        ①
  margin: 0;
  padding: 0;
  background: inherit;
  border-radius: inherit;
}

.share__link:hover,
.share__link:focus-visible {
  border-color: black;
  outline: none;
  transform: scale(1.25);
  background: inherit;
}
```

① Makes the link a circle

Although inheriting values in this manner can be a bit cumbersome, it allows us to make sure that the color is controlled from one place. This approach benefits maintainability in case we decide to change the background's color, and it sets us up to expand our component to support multiple themes. Another option would be to use a custom property for our color.

With the `border-radius` and `background-color` inherited, our hover and focus styles are complete (figure 11.11), but the change when we hover over the link is abrupt. Let's animate the size change.

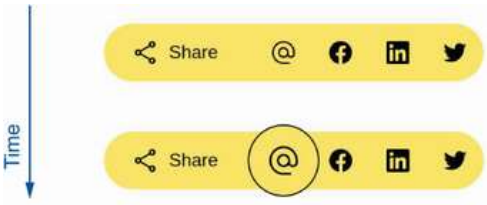

Figure 11.11 Share-link hover effect

1.6 Animating the component

In chapter 2, we used keyframes to create animation, which allowed us to define steps for our animation. For our hover state, we already have our start and end states defined. We're transitioning from one state (not hov-

ered or focused) to another (hovered or focused), whose styles are already defined in rules. So instead of using an animation, we're going to use a transition.

11.6.1 Creating a transition

A transition doesn't require a keyframe but still allows us to animate the change of styles from one state to another. The `transition` property allows us to define which property changes should be animated, as well as the duration and timing function. By adding `transition: transform ease-in-out 250ms;` to our `.share__link` rules, we tell the browser to animate the size change of our link (listing 11.10).

To choose the amount of time the transition needs to take, we choose something relatively fast: 250 milliseconds. We want to keep the animation slow enough to be visible but fast enough to be snappy. If we make the transition too slow, our project will look laggy and distract users from performing the task they're trying to accomplish (sharing the content).

Listing 11.10 Transitioning the link size change

```
.share__link:link,
.share__link:visited {
  text-decoration: none;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  height: 48px;
  width: 48px;
  border-radius: 50%;
  border: solid 1px transparent;
  transition: transform ease-in-out 250ms;
}
```

NOTE You may notice that after adding the transition, the outline gets chopped off on hover. The reason is that JavaScript drives the opening and closing of the component and toggles overflow and visibility. We go into detail on what the JavaScript is doing in section 11.6.2. Clicking the Share button toggles this behavior.

In our transition, we specifically tell the browser to animate the changes that occur on the `transform` property, but we don't have a `transform` property in our `.share__link:link, .share__link:visited` rule. When we run the code, however, we notice that our size

change is animated and that the code works. This behavior occurs because, when not defined, `scale()` equals `scale(1)` by default. Therefore, we're animating going from `scale(1)` to `scale(1.25)` when we hover or focus the link and then animating the scale back to `scale(1)` when we move away from the link.

Next, we're going to animate hiding and exposing the links when the button is clicked.

11.6.2 Opening and closing the component

Remember that our goal is for the component to hide our menu of links by default and expose it only when the Share button is clicked (figure 11.12).

Figure 11.12 Closed and expanded states

The first thing we need to do is hide the menu items by default. To achieve this task, we'll give the `menu a width of 0` and hide the `overflow`, as shown in listing 11.11.

Listing 11.11 Hiding the menu

```
.share__menu {
  display: flex;
  margin: 0;
  padding: 0;
  background: inherit;
  border-radius: inherit;
  width: 0;                    ①
  overflow: hidden;            ②
}
```

① Makes the width of the menu equal to 0

② Hides the overflow so that the links within are also hidden

With our menu hidden (figure 11.13), we need to toggle exposing and hiding the menu when the Share button is clicked.

Figure 11.13 Hidden menu

Our JavaScript handles part of the behavior for us. At the beginning of this chapter, we mentioned that we'll need some JavaScript for this project. When we open the JavaScript file, we notice that it contains a lot of code (listing 11.12).

Listing 11.12 JavaScript file

```
(() => {
  'use strict';

  let expanded = false;
  const container = document.getElementById('share');
  const shareButton = document.getElementById('shareButton');
  const menuItems = Array.from(container.querySelectorAll('li'));
  const menu = container.querySelector('menu');

  addButtonListeners();
  addListListeners();
  addTransitionListeners();

  function addButtonListeners() {                                    ①
    shareButton.addEventListener('click', toggleMenu);               ①
    shareButton.addEventListener('keyup', handleToggleButtonKeypress);  ①
  }                                                                  ①
  function addListListeners() {                                      ②
    menuItems.forEach(li => {                                        ②
        const link = li.querySelector('a');                          ②
        link.addEventListener('keyup', handleMenuItemKeypress);      ②
        link.addEventListener('keydown', handleTab);                 ②
        link.addEventListener('click', toggleMenu);                  ②
      })                                                             ②
  }                                                                  ②

  function addTransitionListeners() {                                ③
    menu.addEventListener('transitionstart', handleAnimationStart);  ③
    menu.addEventListener('transitionend', handleAnimationEnd);      ③
  }                                                                  ③

  function handleToggleButtonKeypress(event) {                       ④
    switch(event.key) {                                              ④
      case 'ArrowDown':                                              ④
      case 'ArrowRight':                                             ④
        if (!expanded) { toggleMenu(); }                             ④
        moveToNext();                                                ④
        break;                                                       ④
      case 'ArrowUp':                                                ④
      case 'ArrowLeft':                                              ④
        if (expanded) { toggleMenu(); }                              ④
        break;                                                       ④
    }                                                                ④
```

```
}                                                                    ④

function handleMenuItemKeypress(event) {                             ⑤
  switch(event.key) {                                                ⑤
    case 'ArrowDown':                                                ⑤
    case 'ArrowRight':                                               ⑤
      moveToNext();                                                  ⑤
      break;                                                         ⑤
    case 'ArrowUp':                                                  ⑤
    case 'ArrowLeft':                                                ⑤
      if (event.altKey === true) {                                   ⑤
        navigate(event);                                             ⑤
        toggleMenu();                                                ⑤
      } else {                                                       ⑤
        moveToPrevious();                                            ⑤
      }                                                              ⑤
      break;                                                         ⑤
    case 'Enter':                                                    ⑤
      toggleMenu();                                                  ⑤
      break;                                                         ⑤
    case ' ':                                                        ⑤
      navigate(event);                                               ⑤
      toggleMenu();                                                  ⑤
      break;                                                         ⑤
    case 'Tab':                                                      ⑤
      event.preventDefault();                                        ⑤
      toggleMenu();                                                  ⑤
      break;                                                         ⑤
    case 'Escape':                                                   ⑤
      toggleMenu();                                                  ⑤
      break;                                                         ⑤
    case 'Home':                                                     ⑤
      moveToNext(0);                                                 ⑤
      break;                                                         ⑤
    case 'End':                                                      ⑤
      moveToNext(menuItems.length - 1);                              ⑤
      break;                                                         ⑤
  }                                                                  ⑤
}                                                                    ⑤

function handleTab(event) {                                          ⑥
  if (event.key !== 'Tab') { return; }                               ⑥
  event.preventDefault();                                            ⑥
}                                                                    ⑥
```

```
function toggleMenu(event) {                                                    ⑦
  expanded = !expanded;                                                         ⑦
  shareButton.ariaExpanded = expanded;                                          ⑦
  container.classList.toggle('share_expanded');                                 ⑦
  if (expanded) {                                                               ⑦
    menuItems.forEach(li => li.removeAttribute('tabindex'));                    ⑦
  }                                                                             ⑦
  if (!expanded) {                                                              ⑦
    menuItems.forEach(li => {                                                   ⑦
      li.removeAttribute('data-current');                                       ⑦
      li.tabIndex = -1;                                                         ⑦
    })                                                                          ⑦
    shareButton.focus();                                                        ⑦
  }                                                                             ⑦
}                                                                               ⑦

function moveToNext(next = undefined) {                                         ⑧
  const selectedIndex = menuItems.findIndex(                                    ⑧
    li => li.dataset.current  === 'true'                                        ⑧
  );                                                                            ⑧
  let newIndex                                                                  ⑧
  if (next) {                                                                   ⑧
    newIndex = next;                                                            ⑧
  } else if (                                                                   ⑧
    selectedIndex === -1 || selectedIndex ===  menuItems.length - 1) {          ⑧
    newIndex = 0;                                                               ⑧
  } else {                                                                      ⑧
    newIndex = selectedIndex + 1;                                               ⑧
  }                                                                             ⑧

  if (selectedIndex !== -1) {                                                   ⑧
    menuItems[selectedIndex].removeAttribute('data-current');                   ⑧
  }                                                                             ⑧
  menuItems[newIndex].setAttribute('data-current', 'true');                     ⑧
  menuItems[newIndex].querySelector('a').focus();                               ⑧
}                                                                               ⑧


function moveToPrevious() {                                                     ⑨
  const selectedIndex = menuItems.findIndex(li => li.dataset.current);          ⑨
  const newIndex = selectedIndex < 1                                            ⑨
    ? menuItems.length - 1                                                      ⑨
    : selectedIndex - 1;                                                        ⑨
```

```
      if (selectedIndex !== -1) {                                    ⑨
        menuItems[selectedIndex].removeAttribute('data-current');    ⑨
      }                                                              ⑨
      menuItems[newIndex].setAttribute('data-current', 'true');      ⑨
      menuItems[newIndex].querySelector('a').focus();                ⑨
    }                                                                ⑨

    function navigate(event) {                                       ⑩
      const url = event.target.href;                                 ⑩
      window.open(url);                                              ⑩
    }                                                                ⑩

    function handleAnimationStart() {                                ⑪
      if (!expanded) { menu.style.overflow = 'hidden' };             ⑪
    }                                                                ⑪

    function handleAnimationEnd() {                                  ⑫
      if (expanded) { menu.style.overflow = 'visible' }              ⑫
    }                                                                ⑫
  })()
```

① Adds event listeners to the Share button for clicks and key-presses to open and close the menu via both keyboard and mouse

② Adds event listens to the links for clicks and keypresses to handle keyboard navigation within the menu

③ Adds event listeners to the menu to know when transitions start and end

④ Handles keyboard up- and down-arrow functionality or the Share button

⑤ Handles keypress on links for keyboard navigation within the menu, including exiting the menu

⑥ Prevents tab from navigating between the links because on tab, we want to return focus to the Share button rather than go to the next link

⑦ Opens and closes the menu

⑧ When next is defined, moves the focus to the specific item by index; otherwise, cycles through the links, returning to the top when the user reaches the last item in the menu

⑨ Moves focus to the previous link and returns the user to the bottom of the list when they reach the first item in the menu

⑩ Navigates the user when the action is keyboard-triggered and not the default click or key-press; used when the user presses the spacebar on a menu item

⑪ Hides overflow when the menu is closing

⑫ If open, shows overflow to allow the magnified icon to expand outside the container

Most of the code handles keyboard accessibility for the component, and listing 11.13 shows the parts that are relevant to the button click. When the page loads, we default the component to being closed and find the element's container, which we assign to the `container` variable. Then we add event listeners to the button so that when the button is clicked, the `toggleMenu()` function is triggered. When the button is clicked, we change the `expanded` variable to its inverse. If the setting was `true`, it becomes `false`, and vice versa. Finally, we add or remove the `share_expanded` class. `classList.toggle()` adds the class if it's not present and removes it if it is.

Listing 11.13 Opening and closing the menu (JavaScript)

```
(() => {
  ...
  let expanded = false;                                          ①
  const container = document.getElementById('share');            ②
  ...
  function addButtonListeners() {                                ③
    shareButton.addEventListener('click', toggleMenu);           ③
    ...                                                          ③
  }                                                              ③

  function toggleMenu(event) {
    expanded = !expanded;                                        ④
```

```
    ...
    container.classList.toggle('share_expanded');          ⑤
    ...
    ...
}
```

① Defines a variable to hold our current state

② Defines a variable for our HTML container element

③ Defines what happens when the button is clicked

④ Toggles the expanded variable value

⑤ Handles adding and removing share_expanded

NOTE Because this book is about CSS, the JavaScript is included in the starter code. If you're following along, you don't need to make any edits to the JavaScript to make it work.

All put together, this code adds the `share_expanded` class to the container when the Share button is clicked. If `share_expanded` is already open, the code removes it. We had hidden our menu items, but now we'll show them when the `share_expanded` class is present.

NOTE Remember that we decided to use BEM for our class-name convention. Our class name has only one underscore because `expanded` is our modifier. We use a modifier because we're changing (modifying) the style based on the state (open/closed). We have the block (`share`) and the modifier (`expanded`); therefore, our class name is `block_modifier` or `share_expanded` .

To show the links when the component is marked as `expanded` , we must increase the width of the menu, as shown in listing 11.14. We also add a little horizontal padding to create some room around the menu.

To calculate the width of the menu, we multiply the number of links by their width. The link's width is 48 pixels (which we hard-set) plus the border (1 pixel on each side). Therefore, the menu's width is `width` = 4 ×(48 + 2) = `200px` .

Listing 11.14 Showing the menu

```
.share_expanded .share__menu {
  width: 200px;
  padding: 0 2rem 0 1rem;
}
```

After clicking the button and hovering over the first link, we see that our link no longer expands outside the menu (figure 11.14). We also see that after we hover over the links and close the menu, our menu items continue to display until we hover over them again.

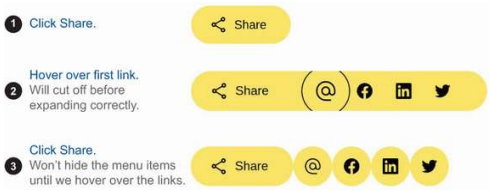

Figure 11.14 Expanded component on click

Remember that our JavaScript triggers when transitions start and end and is responsible for controlling our overflow. Although we've already animated the style changes for hovering over the individual menu items, we haven't added the transition for opening and closing the menu yet. When we add that transition, overflow will be set correctly when the transition activates and finishes, making these problems go away.

The next task we need to accomplish is to maintain the button outline that's usually present on hover when the component is open. Because we

already have a rule to add the border on hover and focus, we're going to edit the rule to trigger when the component is open. By reusing the rule, we ensure that the styles will be consistent in the hover and focus states and when the list is visible. To add the condition, we add the `.share_ expanded .share__button` selector to the rule, as shown in the following listing.

Listing 11.15 Adding button border to Share button when list is displayed

```
.share__button:hover,
.share__button:focus-visible,
.share_expanded .share__button {
  cursor: pointer;
  outline: solid 1px black;
  outline-offset: -5px;
}
```

With the selector added, our button keeps its border after the component is expanded (figure 11.15); and when the component is closed and not focused or hovered, the border stays absent.

Figure 11.15 Maintaining the Share-button border when list is displayed

11.6.3 Animating the menu

Now that we've set our styles for both the open and closed states, let's animate the showing and hiding of the menu. We want the link list to expand from the left, as depicted in figure 11.16.

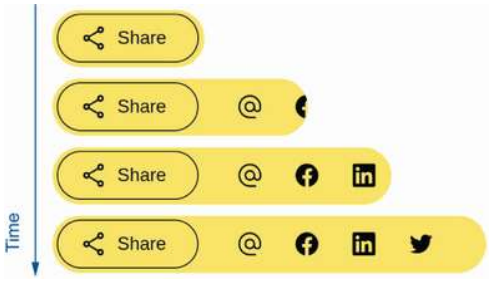

Figure 11.16 Breakdown of opening animation

When the menu closes, we'll want to perform the inverse of the opening animation, retracting the menu and hiding the link. We'll do the same for the magnification effect on the links, using a transition. We don't need to use keyframes because the animation is going to be performed only once (when the button is clicked) and we already have the two states defined.

We'll add the `transition` declaration to the menu as follows: `transition: width 250ms ease-in-out`. Again, we want to keep the transition snappy, so

we give it a duration of 250 milliseconds.

After we add the transition, we realize that icons are becoming visible before they should. Figure 11.17 breaks down the effect.

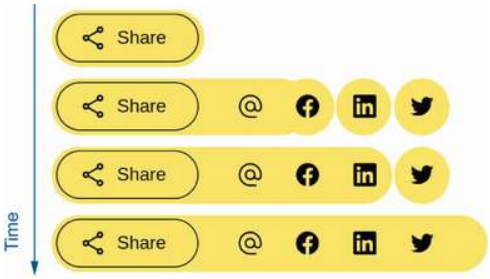

Figure 11.17 Icons displaying too soon

Even if we change the transition to transition all properties instead of only `width`, the same problem occurs. The cause is overflow. When the menu is closed, we want the menu's overflow to be hidden; when it's open, we want it to be visible. But overflow can't be changed gradually, like width. It's either visible or it's not. There's no in-between state.

When opening the menu, we want to wait until the transition is done before we change `overflow` to `visible`. When we close, we want the overflow to be hidden immediately. This task is where we turn to JavaScript to support our CSS.

We'll remove `overflow: visible` from our `.share_expanded` `.share__menu` class and handle adding it via JavaScript.

Listing 11.16 singles out the relevant JavaScript for handling the overflow. The magic lies in the `transitionstart` and `transitionend` event listeners. Attached to the menu, they listen for when the transition is triggered and when it's done performing the change. When the event happens, they trigger their functions to handle the overflow for the menu.

Listing 11.16 JavaScript for handling overflow

```
(() => {
  'use strict';

  let expanded = false;
  const container = document.getElementById('share');
  const menu = container.querySelector('menu');
...
  addTransitionListeners();
...
  function addTransitionListeners() {
    menu.addEventListener('transitionstart', handleAnimationStart);
    menu.addEventListener('transitionend', handleAnimationEnd);
  }
...
  function handleAnimationStart() {                          ①
    if (!expanded) { menu.style.overflow = 'hidden'; }       ②
  }
...
```

```
    function handleAnimationEnd() {                          ③
      if (expanded) { menu.style.overflow = 'visible'; }     ④
    }
  })()
```

① Triggers when the transition starts

② If in the process of closing, hides the menu's overflow

③ Triggers when the transition ends

④ If just opened, shows the overflow

NOTE As we mention earlier in the chapter, the JavaScript is included in the starter code. If you're following along, you don't need to edit the JavaScript; it should work.

The next listing shows the CSS that makes the animation work.

Listing 11.17 Updated CSS for open and close animation

```
  .share__menu {
    display: flex;
    margin: 0;
    padding: 0;
    background: inherit;
    border-radius: inherit;
    width: 0;
    overflow: hidden;
```

```
  transition: width 250ms ease-in-out;      ①
}
```

① Adds the animation

With these last edits made to make the animation smooth, we've finished our animated social media share component. The final product is shown in figure 11.18.

Figure 11.18 Final product

ummary

- We have several ways to organize CSS. Three common patterns are OOCSS, SMACSS, and BEM.
- Icons are subject to copyright, so follow brand guidelines when using social media icons.
- We can make elements displayed via Flexbox behave like inline-level elements by using `inline-flex`. `inline-flex` uses the same properties as `flex`.
- The position of an outline can be controlled via `outline-offset`.

- The `scale()` function allows us to grow or shrink an element proportionally.
- The `inherit` property value allows us to inherit values from the parent element that generally wouldn't be inherited.
- Transitions don't require keyframes but still allow us to animate CSS changes from one state to another.
- The `overflow` property allows us to control whether elements that extend beyond their container are displayed or hidden.
- When using JavaScript to extend our transitions' functionality, we can use the `ontransitionstart` and `ontransitionend` event listeners to trigger JavaScript change in response to the transition's life cycle.