Building a theme switch component

- Overview
- Markup
  - The button
    - aria-label and aria-live polite
  - The scalable vector graphic (SVG) icon
    - The sun
    - The sun beams
    - The moon
    - What happens if CSS doesn’t load?
- Layout
- Styles
  - .theme-toggle styles
    - Adaptive sizing with the hover media query
  - Sun and moon SVG styles
    - Light theme
    - Dark theme
- Animation
  - Sharing media queries and importing easings
  - The sun
  - The moon
  - Prefers reduced motion
- JavaScript
  - The page load experience
  - The toggling experience
  - Synchronizing with the system
- Conclusion

- Community remixes

In this post I want to share thinking on a way to build a dark and light theme switch component. Try the demo.

> Demo button size increased for easy visibility

If you prefer video, here's a YouTube version of this post:

https://www.youtube.com/watch?v=kZiS1QStIWc

Overview

A website may provide settings for controlling the color scheme instead of relying entirely on the system preference. This means that users may browse in a mode other than their system preferences. For example, a user's system is in a light theme, but the user prefers the website to display in the dark theme.

There are several web engineering considerations when building this feature. For example, the browser should be made aware of the preference as soon as possible to prevent page color flashes, and the control needs to first sync with the system then allow client-side stored exceptions.

> Diagram shows a preview of JavaScript page load and document interaction events to overall show there's 4 paths to setting the theme

Markup

A `<button>` should be used for the toggle, as you then benefit from browser-provided interaction events and features, such as click events and focusability.

The button

The button needs a class for use from CSS and an ID for use from JavaScript. Additionally, since the button content is an icon rather than text, add a title attribute to provide information about the button's purpose. Last, add an `[aria-label]` to hold the state of the icon button, so screen readers can share the state of the theme to folks who are visually impaired.

```
<button
  class="theme-toggle"
  id="theme-toggle"
  title="Toggles light & dark"
  aria-label="auto"
>
  …
</button>
```

`aria-label` and `aria-live` polite

To indicate to screen readers that changes to `aria-label` should be announced, add `aria-live="polite"` to the button.

```
<button
  class="theme-toggle"
  id="theme-toggle"
  title="Toggles light & dark"
  aria-label="auto"
  aria-live="polite"
>
  …
</button>
```

This markup addition signals to screen readers to politely, instead of `aria-live="assertive"`, tell the user what changed. In the case of this button, it will announce "light" or "dark" depending on what the `aria-label` has become.

The scalable vector graphic (SVG) icon

SVG provides a way to create high-quality, scalable shapes with minimal markup. Interacting with the button can trigger new visual states for the vectors, making SVG great for icons.

The following SVG markup goes inside the `<button>`:

```
<svg class="sun-and-moon" aria-hidden="true" width="24" height="24" viewBox="0 0 24 24">
  …
</svg>
```

`aria-hidden` has been added to the SVG element so screen readers know to ignore it as it's marked presentational. This is great to do for visual decorations, like the icon inside a button. In addition to

the required `viewBox` attribute on the element, add height and width for similar reasons that images should get inline sizes.

The sun

The sun icon shown with the sunbeams faded out and a hotpink arrow pointing to the circle in the center.

The sun graphic consists of a circle and lines which SVG conveniently has shapes for. The `<circle>` is centered by setting the `cx` and `cy` properties to 12, which is half of the viewport size (24), and then given a radius (`r`) of `6` which sets the size.

```
<svg class="sun-and-moon" aria-hidden="true" width="24" height="24" viewBox="0 0 24 24">
  <circle class="sun" cx="12" cy="12" r="6" mask="url(#moon-mask)" fill="currentColor" />
</svg>
```

Additionally, the mask property points to an SVG element's ID, which you will create next, and finally given a fill color that matches the page's text color with `currentColor`.

The sun beams

The sun icon shown with the sun center faded out and a hotpink arrow pointing to the sunbeams.

Next, the sunbeam lines are added just below the circle, inside of a group element `<g>` group.

```
<svg class="sun-and-moon" aria-hidden="true" width="24" height="24" viewBox="0 0 24 24">
  <circle class="sun" cx="12" cy="12" r="6" mask="url(#moon-mask)" fill="currentColor" />
  <g class="sun-beams" stroke="currentColor">
    <line x1="12" y1="1" x2="12" y2="3" />
    <line x1="12" y1="21" x2="12" y2="23" />
    <line x1="4.22" y1="4.22" x2="5.64" y2="5.64" />
    <line x1="18.36" y1="18.36" x2="19.78" y2="19.78" />
    <line x1="1" y1="12" x2="3" y2="12" />
    <line x1="21" y1="12" x2="23" y2="12" />
    <line x1="4.22" y1="19.78" x2="5.64" y2="18.36" />
    <line x1="18.36" y1="5.64" x2="19.78" y2="4.22" />
  </g>
</svg>
```

This time, instead of the value of fill being `currentColor`, each line's stroke is set. The lines plus the circle shapes create a nice sun with beams.

Icon SVG sourced from https://feathericons.com/?query=sun with minor adjustments made.

The moon

To create the illusion of a seamless transition between light (sun) and dark (moon), the moon is an augmentation of the sun icon, using an SVG mask.

```
<svg class="sun-and-moon" aria-hidden="true" width="24" height="24" viewBox="0 0 24 24">
  <circle class="sun" cx="12" cy="12" r="6" mask="url(#moon-mask)" fill="currentColor" />
  <g class="sun-beams" stroke="currentColor">
    …
  </g>
  <mask class="moon" id="moon-mask">
    <rect x="0" y="0" width="100%" height="100%" fill="white" />
    <circle cx="24" cy="10" r="6" fill="black" />
  </mask>
</svg>
```

> *Graphic with three vertical layers to help show how masking works. The top layer is a white square with a black circle. The middle layer is the sun icon. The bottom layer is labeled as the result and it shows the sun icon with a cutout where the top layer black circle is.*

Masks with SVG are powerful, allowing the colors white and black to either remove or include parts of another graphic. The sun icon will be eclipsed by a moon `<circle>` shape with an SVG mask, simply by moving a circle shape in and out of a mask area.

What happens if CSS doesn’t load?

> *Screenshot of a plain browser button with the sun icon inside.*

It can be nice to test your SVG as if CSS didn't load to ensure the result isn't super large or causing layout issues. The inline height and width attributes on the SVG plus the use of `currentColor` give minimal style rules for the browser to use if CSS doesn't load. This makes for nice defensive styles against network turbulence.

Layout

The theme switch component has little surface area, so you don’t need grid or flexbox for layout. Instead, SVG positioning and CSS transforms are used.

Styles

`.theme-toggle` styles

The `<button>` element is the container for the icon shapes and styles. This parent context will hold adaptive colors and sizes to pass down to SVG.

The first task is to make the button a circle and remove the default button styles:

```
.theme-toggle {
  --size: 2rem;

  background: none;
  border: none;
  padding: 0;

  inline-size: var(--size);
  block-size: var(--size);
  aspect-ratio: 1;
  border-radius: 50%;
}
```

Next, add some interaction styles. Add a cursor style for mouse users. Add `touch-action: manipulation` for a fast reacting touch experience. Remove the semi-transparent highlight iOS applies to buttons. Last, give the focus state outline some breathing room from the edge of the element:

```
.theme-toggle {
  --size: 2rem;

  background: none;
  border: none;
  padding: 0;

  inline-size: var(--size);
  block-size: var(--size);
  aspect-ratio: 1;
  border-radius: 50%;
```

```
  cursor: pointer;
  touch-action: manipulation;
  -webkit-tap-highlight-color: transparent;
  outline-offset: 5px;
}
```

The SVG inside of the button needs some styles as well. The SVG should fit the size of the button and, for visual softness, round out the line ends:

```
.theme-toggle {
  --size: 2rem;

  background: none;
  border: none;
  padding: 0;

  inline-size: var(--size);
  block-size: var(--size);
  aspect-ratio: 1;
  border-radius: 50%;

  cursor: pointer;
  touch-action: manipulation;
  -webkit-tap-highlight-color: transparent;
  outline-offset: 5px;

  & > svg {
    inline-size: 100%;
    block-size: 100%;
    stroke-linecap: round;
  }
}
```

The CSS @nest feature, used with PostCSS Nesting, allows authoring new styles without leaving a selectors context. The above nested selector creates `.theme-toggle > svg` because the `&` character is dynamically represented as `.theme-toggle`.

Adaptive sizing with the `hover` media query

The icon button size is a bit small at `2rem`, which is fine for mouse users but can be a struggle for a coarse pointer like a finger. Make the button meet many touch size guidelines by using a hover

media query to specify a size increase.

```
.theme-toggle {
  --size: 2rem;
  …

  @media (hover: none) {
    --size: 48px;
  }
}
```

Sun and moon SVG styles

The button holds the interactive aspects of the theme switch component while SVG inside will hold the visual and animated aspects. This is where the icon can be made beautiful and brought to life.

Light theme

ALT_TEXT_HERE

For scale and rotate animations to happen from the center of SVG shapes, set their `transform-origin: center center`. The adaptive colors provided by the button are used here by the shapes. The moon and sun use the button provided `var(--icon-fill)` and `var(--icon-fill-hover)` for their fill, while the sunbeams use the variables for stroke.

```
.sun-and-moon {
  & > :is(.moon, .sun, .sun-beams) {
    transform-origin: center center;
  }

  & > :is(.moon, .sun) {
    fill: var(--icon-fill);

    @nest .theme-toggle:is(:hover, :focus-visible) > & {
      fill: var(--icon-fill-hover);
    }
  }

  & > .sun-beams {
    stroke: var(--icon-fill);
```

```
    stroke-width: 2px;

    @nest .theme-toggle:is(:hover, :focus-visible) & {
      stroke: var(--icon-fill-hover);
    }
  }
}
```

`@nest` is used again, this time to make a new selector that targets a parent button being hovered or focused. `@nest .theme-toggle:is(:hover, :focus-visible) &` in this usage is the same as `@nest .theme-toggle:is(:hover, :focus-visible) .sun-and-moon > .sun-beams`.

Dark theme

ALT_TEXT_HERE

The moon styles need to remove the sunbeams, scale up the sun circle and move the circle mask.

```
.sun-and-moon {
  @nest [data-theme="dark"] & {
    & > .sun {
      transform: scale(1.75);
    }

    & > .sun-beams {
      opacity: 0;
    }

    & > .moon > circle {
      transform: translateX(-7px);

      @supports (cx: 1) {
        transform: translateX(0);
        cx: 17;
      }
    }
  }
}
```

Notice the dark theme has no color changes or transitions. The parent button component owns the colors, where they're already adaptive within a dark and light context. The transition information

should be behind a user's motion preference media query.

Animation

The button should be functional and stateful but without transitions at this point. The following sections are all about defining **how** and **what** transitions.

Sharing media queries and importing easings

To make it easy to put transitions and animations behind a user's operating system motion preferences, the PostCSS plugin Custom Media enables usage of the drafted CSS specification for media query variables syntax:

```
@custom-media --motionOK (prefers-reduced-motion: no-preference);

/* usage example */
@media (--motionOK) {
  .sun {
    transition: transform .5s var(--ease-elastic-3);
  }
}
```

For unique and easy to use CSS easings, import the easings portion of Open Props:

```
@import "https://unpkg.com/open-props/easings.min.css";

/* usage example */
.sun {
  transition: transform .5s var(--ease-elastic-3);
}
```

The sun

The sun transitions will be more playful than the moon, achieving this effect with bouncy easings. The sunbeams should bounce a small amount as they rotate and the center of the sun should bounce a small amount as it scales.

The default (light theme) styles define the transitions and the dark theme styles define customizations for the transition to light:

```
□□.sun-and-moon {
  @media (--motionOK) {
    & > .sun {
      transition: transform .5s var(--ease-elastic-3);
    }

    & > .sun-beams {
      transition:
        transform .5s var(--ease-elastic-4),
        opacity .5s var(--ease-3)
      ;
    }

    @nest [data-theme="dark"] & {
      & > .sun {
        transform: scale(1.75);
        transition-timing-function: var(--ease-3);
        transition-duration: .25s;
      }

      & > .sun-beams {
        transform: rotateZ(-25deg);
        transition-duration: .15s;
      }
    }
  }
}
```

In the Animation panel in Chrome DevTools, you can find a timeline for animation transitions. The duration of the total animation, the elements, and the easing timing can be inspected.

Light to dark transition

Dark to light transition

The moon

The moon light and dark positions are already set, add transition styles inside of the `--motionOK` media query to bring it to life while respecting the user's motion preferences.

The timing with delay and duration are critical in making this transition clean. If the sun is eclipsed too early, for example, the transition doesn't feel orchestrated or playful, it feels chaotic.

```
.sun-and-moon {
  @media (--motionOK) {
    & .moon > circle {
      transform: translateX(-7px);
      transition: transform .25s var(--ease-out-5);

      @supports (cx: 1) {
        transform: translateX(0);
        cx: 17;
        transition: cx .25s var(--ease-out-5);
      }
    }

    @nest [data-theme="dark"] & {
      & > .moon > circle {
        transition-delay: .25s;
        transition-duration: .5s;
      }
    }
  }
}
```

Light to dark transition

Dark to light transition

Prefers reduced motion

In most GUI Challenges I try to keep some animation, like opacity cross fades, for users who prefer reduced motion. This component felt better with instant state changes however.

JavaScript

There's a lot of work for JavaScript in this component, from managing ARIA information for screen readers to getting and setting values from local storage.

The page load experience

It was important that no color flashing occurs on page load. If a user with a dark color scheme indicates they preferred light with this component, then reloaded the page, at first the page would be dark then it would flash to light. Preventing this meant running a small amount of blocking JavaScript with the goal to set the HTML attribute `data-theme` as early as possible.

```
<script src="./theme-toggle.js"></script>
```

To achieve this, a plain `<script>` tag in the document `<head>` is loaded first, before any CSS or `<body>` markup. When the browser encounters an unmarked script like this, it runs the code and executes it before the rest of the HTML. Using this blocking moment sparingly, it's possible to set the HTML attribute before the main CSS paints the page, thus preventing a flash or colors.

The JavaScript first checks for the user's preference in local storage and fallback to check the system preference if nothing is found in storage:

```
const storageKey = 'theme-preference'

const getColorPreference = () => {
  if (localStorage.getItem(storageKey))
    return localStorage.getItem(storageKey)
  else
    return window.matchMedia('(prefers-color-scheme: dark)').matches
      ? 'dark'
      : 'light'
}
```

A function to set the user's preference in local storage is parsed next:

```
const setPreference = () => {
  localStorage.setItem(storageKey, theme.value)
  reflectPreference()
}
```

Followed by a function to modify the document with the preferences.

```
const reflectPreference = () => {
  document.firstElementChild
    .setAttribute('data-theme', theme.value)

  document
    .querySelector('#theme-toggle')
    ?.setAttribute('aria-label', theme.value)
}
```

An important thing to note at this point is the HTML document parsing state. The browser doesn't know about the "#theme-toggle" button yet, as the `<head>` tag hasn't been completely parsed. However, the browser does have a `document.firstElementChild` , aka the `<html>` tag. The function attempts to set both to keep them in sync, but on first run will only be able to set the HTML tag.

The `querySelector` won't find anything at first and the optional chaining operator ensures no syntax errors when it's not found and the setAttribute function is attempted to be invoked.

Next, that function `reflectPreference()` is immediately called so the HTML document has its `data-theme` attribute set:

```
reflectPreference()
```

The button still needs the attribute, so wait for the page load event, then it will be safe to query, add listeners and set attributes on:

```
window.onload = () => {
  // set on load so screen readers can get the latest value on the button
  reflectPreference()

  // now this script can find and listen for clicks on the control
  document
    .querySelector('#theme-toggle')
    .addEventListener('click', onClick)
}
```

The toggling experience

When the button is clicked, the theme needs to be swapped, in JavaScript memory and in the document. The current theme value will need to be inspected and a decision made about its new state. Once the new state is set, save it and update the document:

```
const onClick = () => {
  theme.value = theme.value === 'light'
    ? 'dark'
    : 'light'

  setPreference()
}
```

Synchronizing with the system

Unique to this theme switch is synchronization with the system preference as it changes. If a user changes their system preference while a page and this component are visible, the theme switch will change to match the new user preference, as if the user had interacted with the theme switch at the same time it did the system switch.

Achieve this with JavaScript and a `matchMedia` event listening for changes to a media query:

```
window
  .matchMedia('(prefers-color-scheme: dark)')
  .addEventListener('change', ({matches:isDark}) => {
    theme.value = isDark ? 'dark' : 'light'
    setPreference()
  })
```

Changing the MacOS system preference changes the theme switch state

Conclusion

Now that you know how I did it, how would you? □□

Let's diversify our approaches and learn all the ways to build on the web. Create a demo, tweet me links, and I'll add it to the community remixes section below!

Community remixes

- @NathanG on Codepen with Vue
- @ShadowShahriar on Codepen

- @tomayac as a custom element
- @bramus with vanilla JavaScript
- @JoshWComeau with react

□□□ #3
□ □□□ □□□ 2026-10-31 21:41:39 UTC
□ cocobook-sync □□□ 2026-11-04 17:36:41 UTC