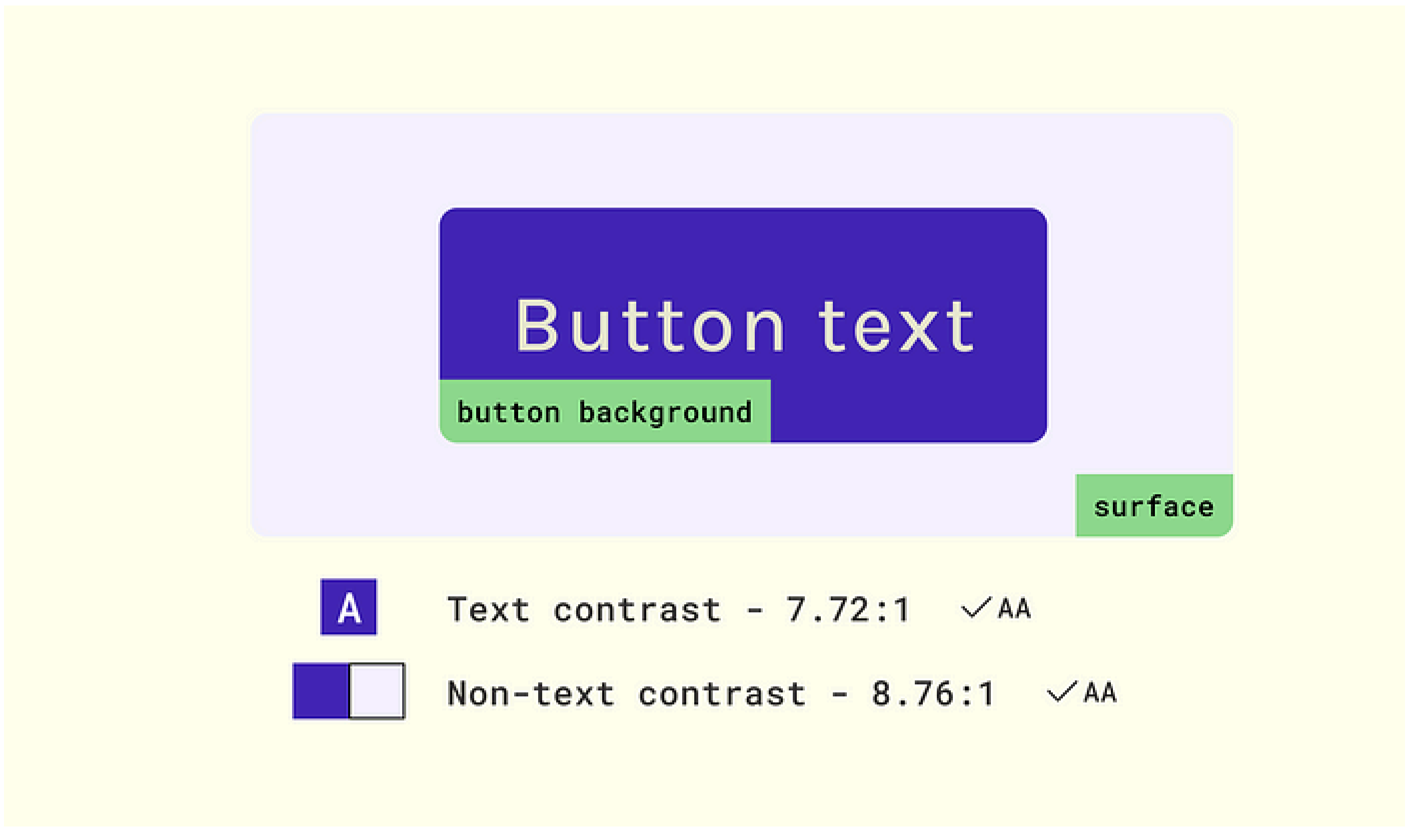
< Go to the original



Creating an accessible button component

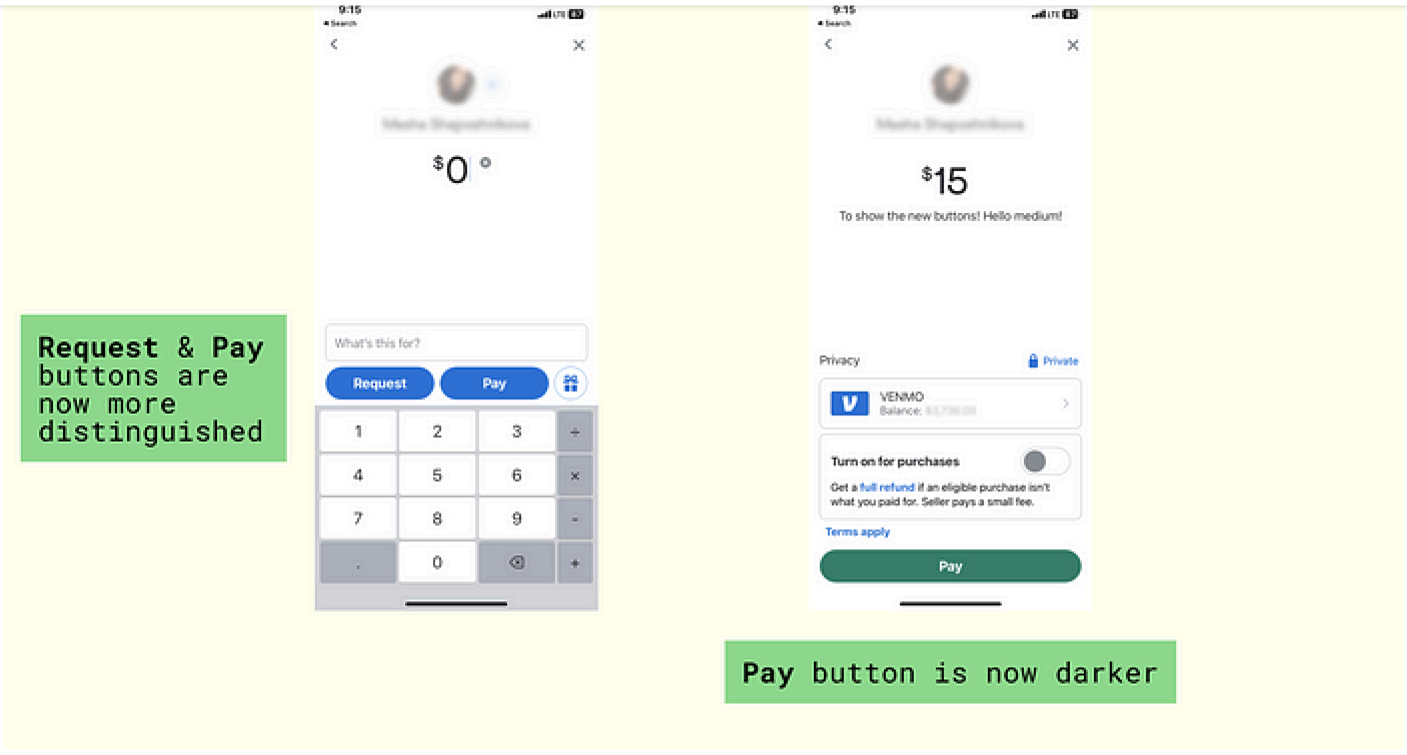
Everything you need to know to ensure your buttons are WCAG AA compliant.

Alex Zlatkus
Follow

UX Collective a11y-light ~6 min read · April 21, 2023 (Updated: April 25, 2023) · Free: No

Background

2023 has already seen a continuation of redesigns initiated by accessibility — and no component is affected more than the button.

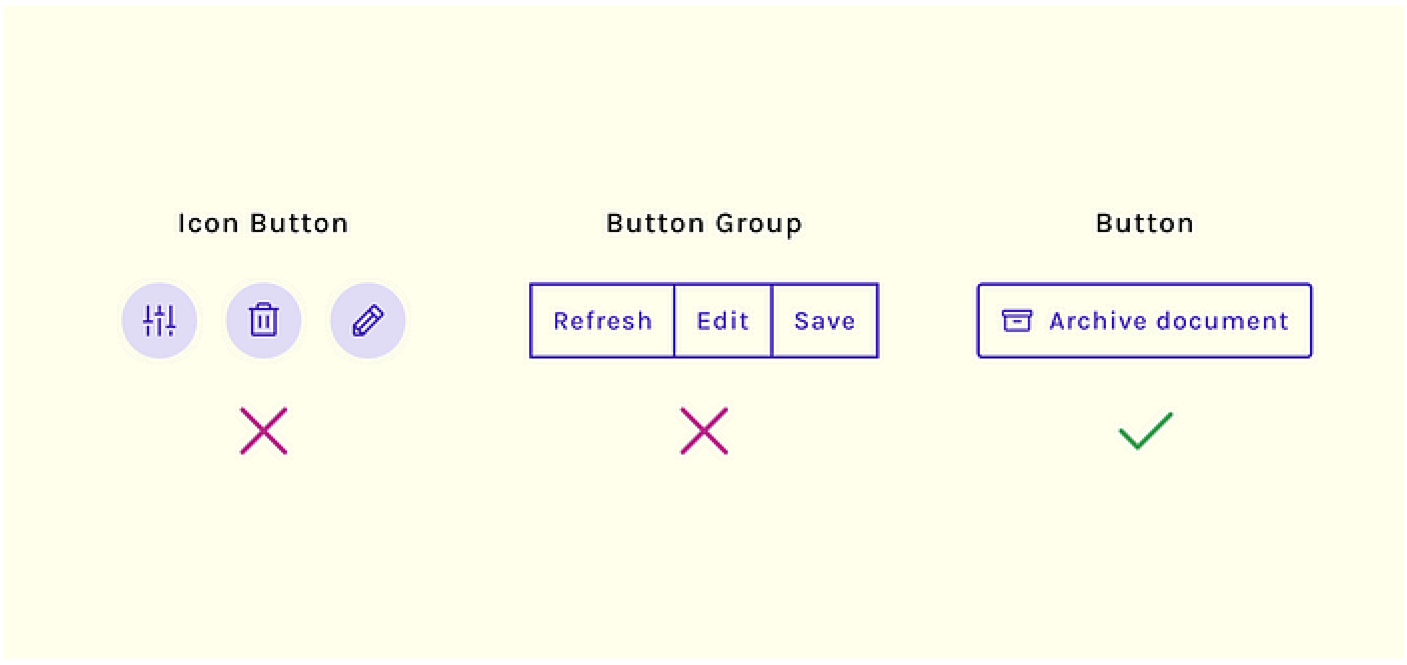


UI changes for the Venmo app.

Now, more than ever, is the time to ensure your most interactable components can be used by everyone.

Scope

This article will not focus on button groups, or icon buttons. We will dive into the standard form of a button, which is a rectangular component that performs an action (not navigation) when triggered.

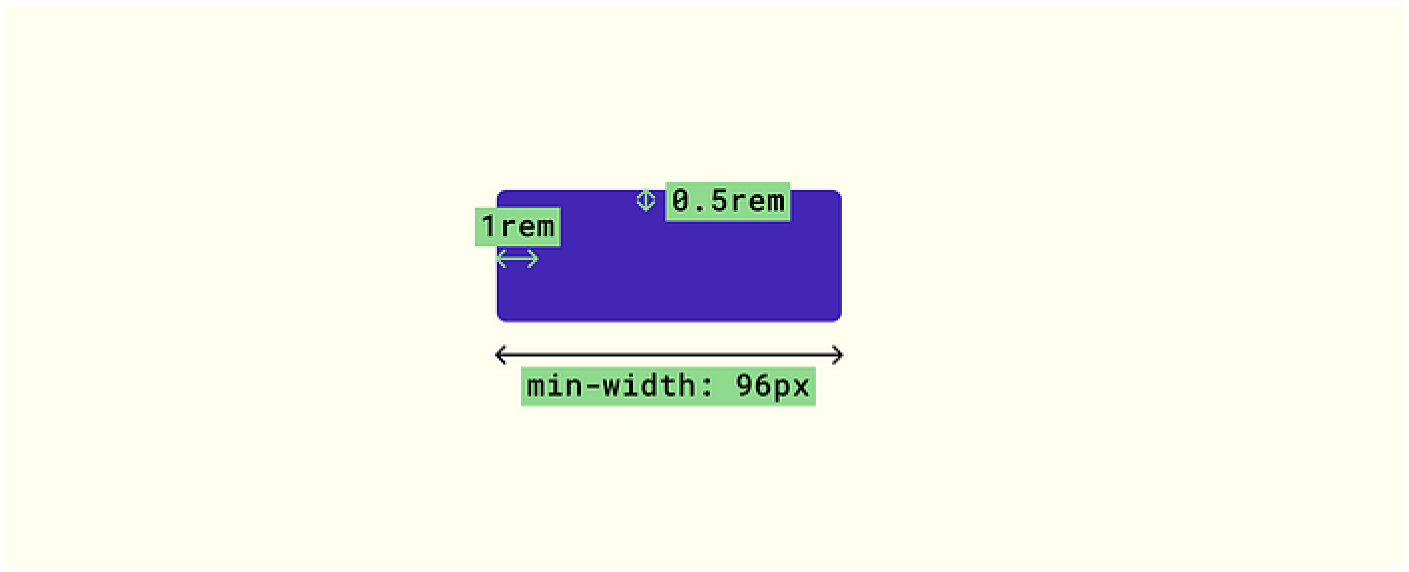


Size

We'll keep this one pretty short. Success Criterion 2.5.5 states interactable components have to be at least 44px x 44px in size. Although this is a **AAA** criteria, I think standard buttons can easily be 44px in height.

I also like to put a specified `min-width` so that buttons with short titles, like **Go** and **Save** are not significantly skinnier than other buttons. I like to make this `min-width` 96px (in this specific case I do not want to use rem).

How I'd style it



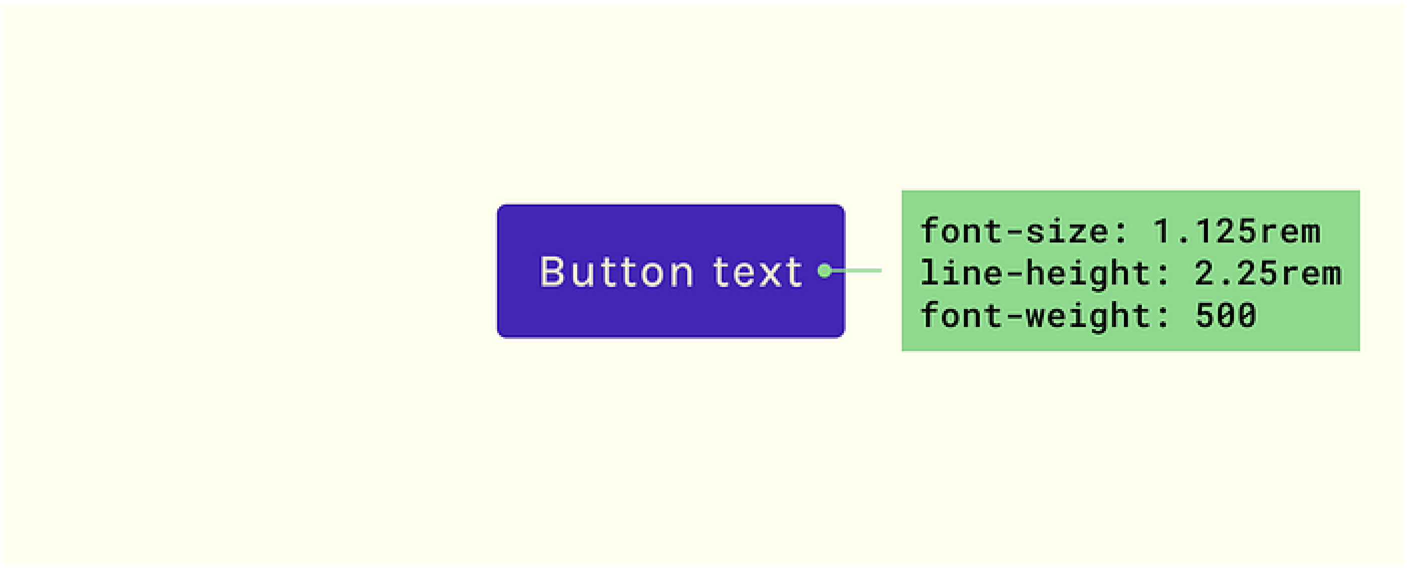
Font Size

There is no official minimum font size for the web, although the way the industry is going I wouldn't do anything lower than 0.875rem.

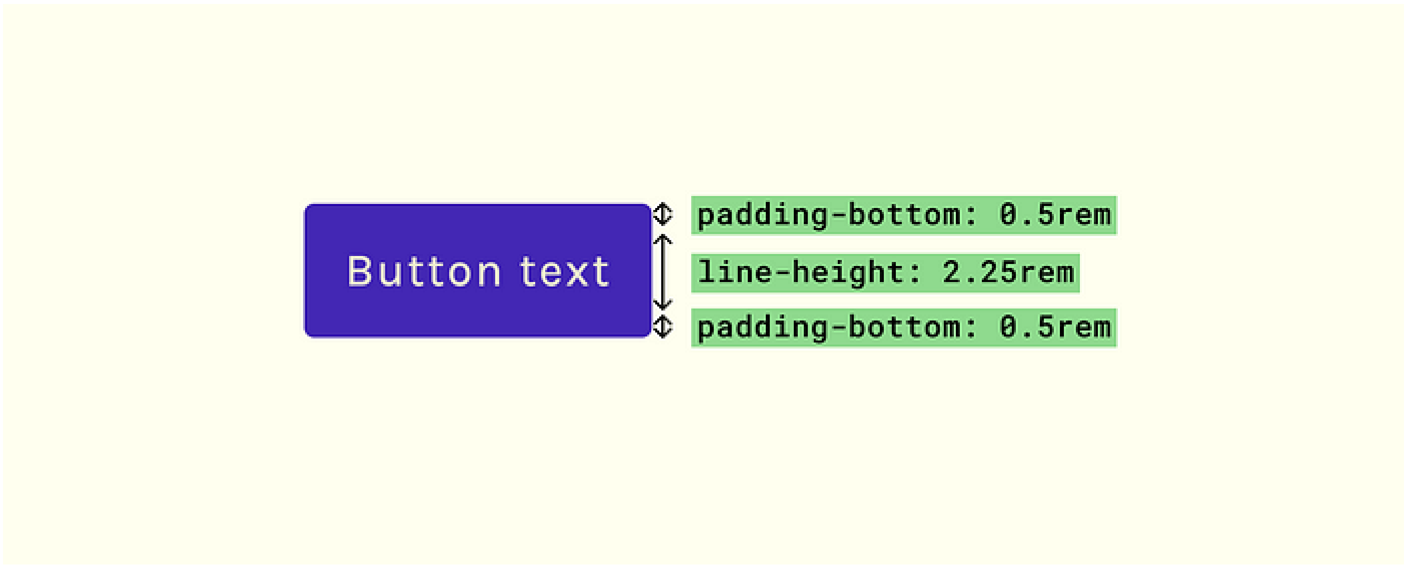
Side note: I especially like to avoid mentioning pixels when referring to typography because font variables need to be defined in rems. Font variables defined in pixels will fail Success Criterion 1.4.4 which states text should be able to be resized by 200%. (There are two main ways to resize text — browser zoom & changing base font size. If you do not define font size in rem then changing the base font size will do nothing.)

How I'd style it

I like to make the font 1.125rem so that this leaves room for text of lower importance to be 1rem. I'd leave 0.875rem for tools with a lot of textual hierarchy or for components that usually are small in stature, e.g. badges, tags, etc.

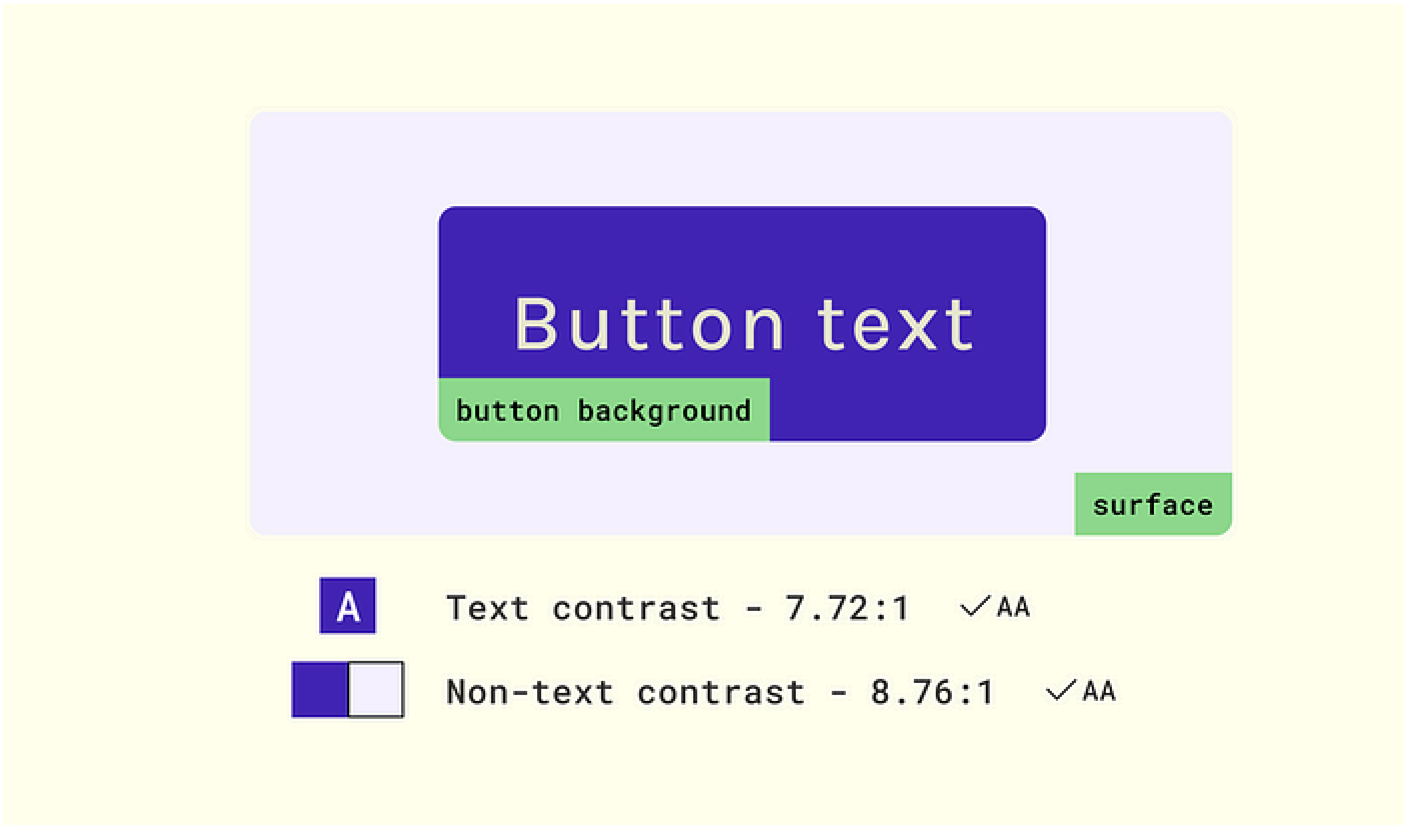


Combining all height-specific info we'd have the following:



Color

A button's label (and icon) needs a 4.5:1 contrast with the background of the button (Success Criterion 1.4.3), and the background of the button needs a 3:1

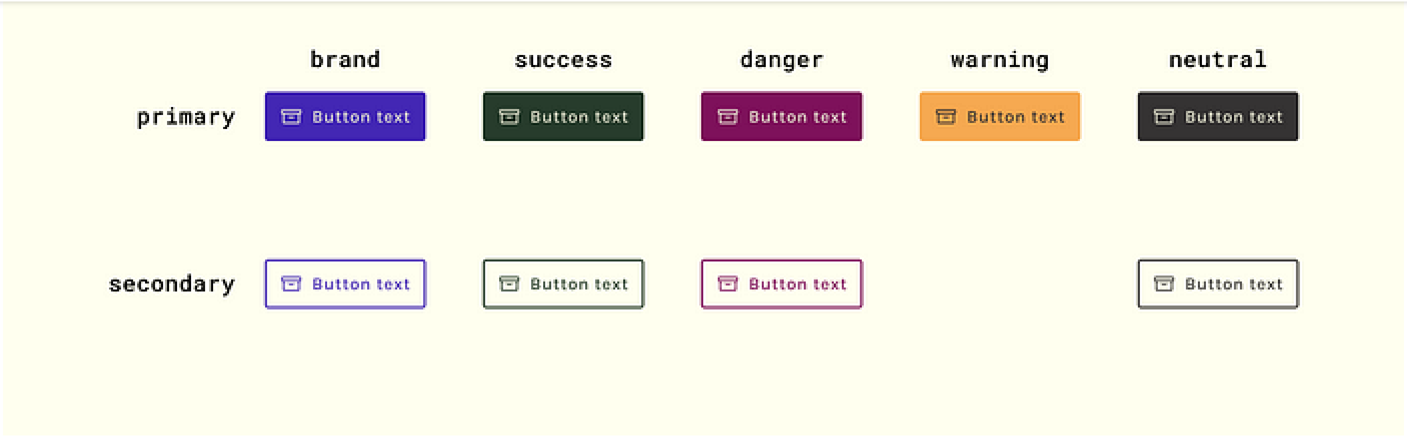


This one can be the marketing team's kryptonite as brand colors are often not accessible when using it as a background to white text.

Furthermore, this means ghost buttons, or even tonal buttons would not pass Success Criterion 1.4.11 and thus would need a button-defining feature — to be discussed below. This topic seems to be the most highly-contested, and I've had some good discussions in Stark's slack channel on it 😃

How I'd style it

I would just avoid ghost and tonal buttons outright and use outline buttons for secondary emphasis. If anything needs to have "tertiary" emphasis a button styled like a link can be used.



The *warning* button does not pass 1.4.11 so it is important that it is accompanied

The *warning* button does not pass 1.4.11 so it is in the same camp as ghost/tonal buttons in that it needs at least one of the following button-defining features:

1. It contains an icon.
2. It is positioned in an area that usually contains button(s), like the bottom corner of a modal.
3. Its name follows the <action> <object> format.

Lastly, the brand color does NOT have to be the same shade in digital products. Many companies opt for darker shades in their app(s) without changing what they use for their e.g. posters, business cards, etc.

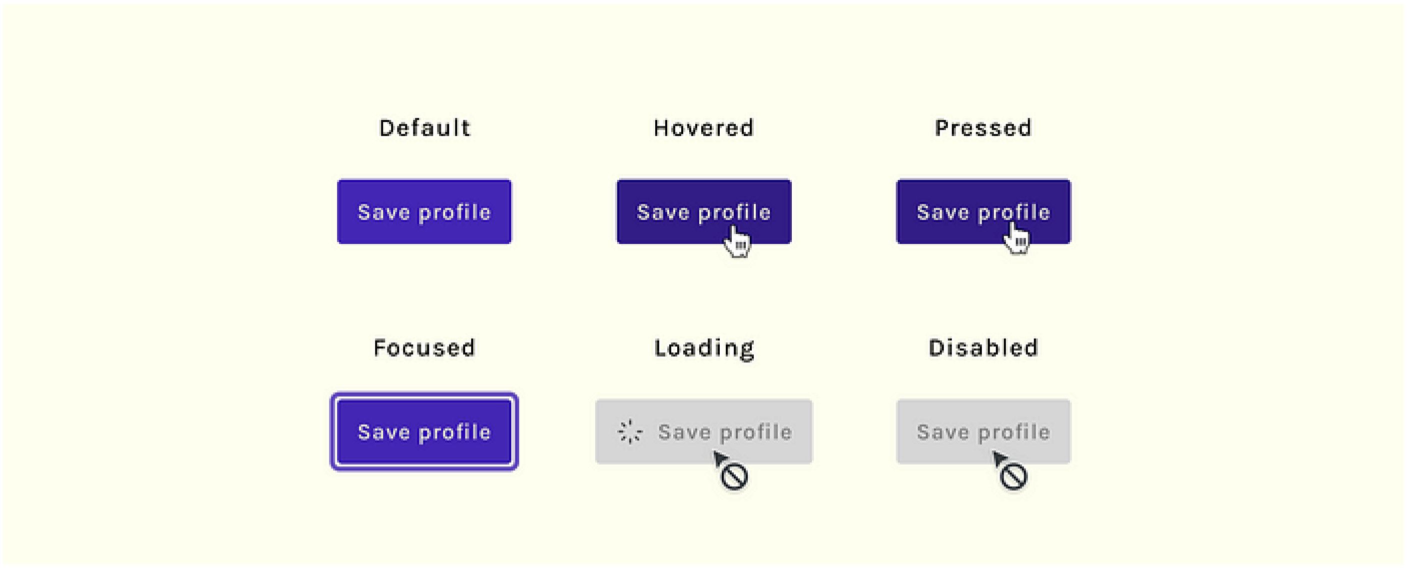
States

This section will be using the word "state" in a general sense, encompassing the `: hover`*, &* `: focus` pseudo-classes and the `loading` & `disabled` states.

Success Criterion 1.4.3 for color contrast applies across all button states, however, it does not apply *between* states. This means there is no required contrast ratio between the background of an default state button vs the background of a hovered button.

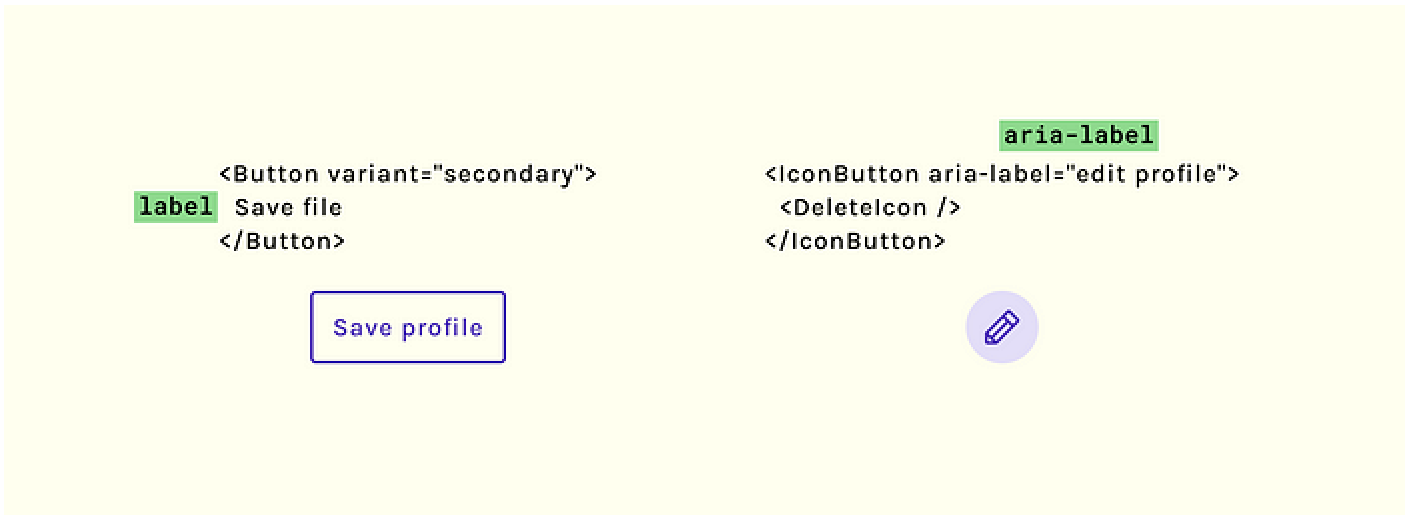
That being said, we should still be cognizant of why there are color changes in the first place and design accordingly.

- I'd make the *default* state following the contrast rules described above.
- I'd make the *hover* state with even more contrast to (a) draw attention to what is being hovered and (b) align with a physical buttons experience a shadow as a hand closes in on it.
- I'd style the *pressed* state the same as the hover state and ensure the focus ring (mentioned below) does not appear.
- I'd style the *focused* state the same as the *default* state except for the fact that it needs an additional focus indicator. For buttons I'd use a focus ring with an outline offset so it is not touching the edge of the button. This ensures the ring will only need a 3:1 contrast with the surface, and not the wide-ranging button background.
- The *disabled* state does not need to meet either color contrast criteria as it is an inactive user interface component. I lean more toward making is visually look disabled than worrying about any contrast. When hovering over a disabled button the cursor should be not-allowed so that users leveraging a cursor have an extra affordance that the button is not clickable.
- I'd style the *loading* state the same as the *disabled* state and would just add a loading animation that has higher contrast than the text. If there is an icon, replace it with the animation. `cursor: 'not-allowed'` also applies for this state. Changing the button's label while loading is not worth the effort.



aria-label

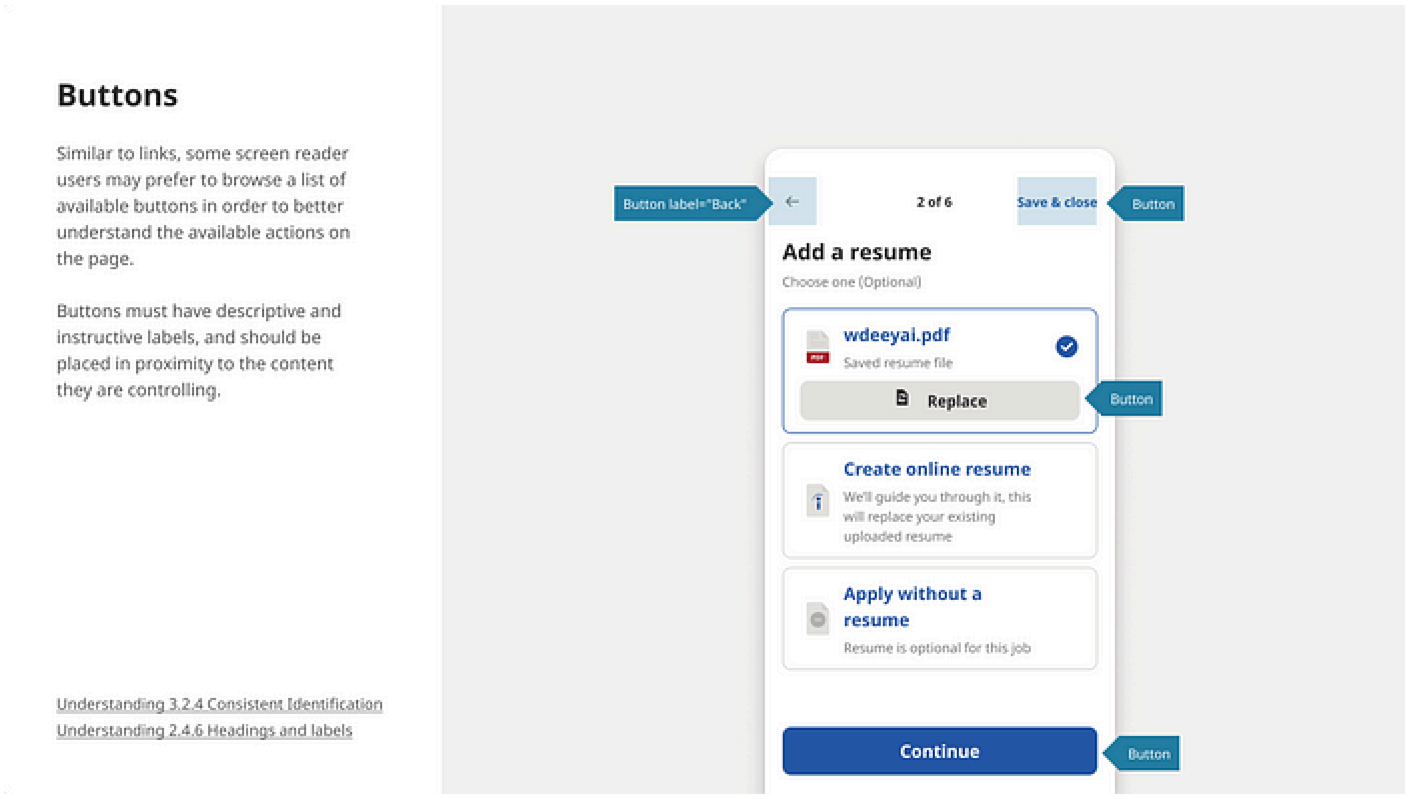
Invisible in the UI, aria-labels are tied to interactive components and are read by assistive technologies (e.g. screen readers). These are critical for components like icon buttons, which may be easy to understand if you can visually identify the icon, but would be impossible to discern if a screen reader just said "button".



```
<Button variant="secondary">
  Save file
</Button>
```

```
<IconButton aria-label="edit profile">
  <DeleteIcon />
</IconButton>
```

A11y's Figma community file perfectly showcases this where only the *back* (icon) button has an annotated `aria-label`.



Buttons

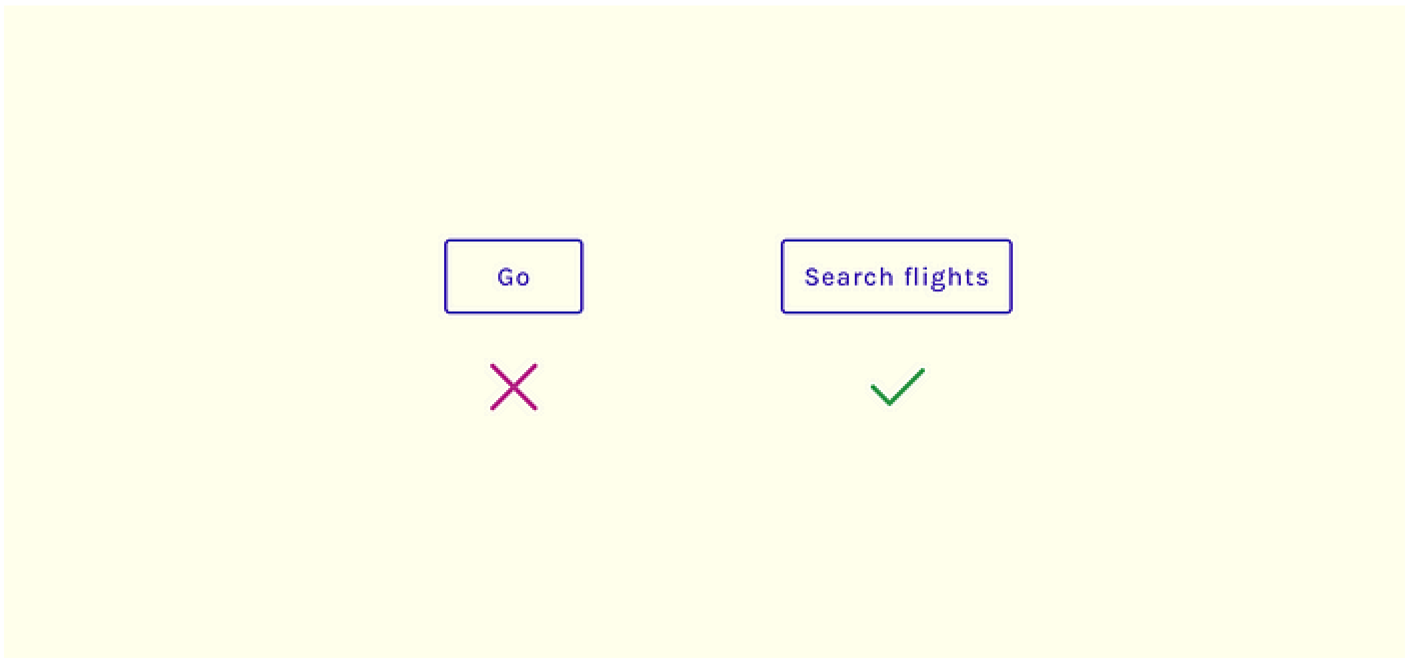
Similar to links, some screen reader users may prefer to browse a list of available buttons in order to better understand the available actions on the page.

Buttons must have descriptive and instructive labels, and should be placed in proximity to the content they are controlling.

Understanding 3.2.4 Consistent Identification
Understanding 2.4.6 Headings and labels

Criteria 3.2.4 & Criteria 2.4.6, as seen in the image.

Keeping the scope to just textual buttons, ensure labels are descriptive and instructive. Then an `aria-label` is not needed, and should be null.



In code **aria label** should not be present at all.

And that's it 🎉

I am not an accessibility expert and am still learning, so please provide me with any/all feedback.

#accessibility #wcag #ux #ui #product-design