Mozilla: JavaScript and CSS Accessibility for Mobile

Mobile Accessibility, General

General Mozilla doc on Mobile Accessibility, a good supplement to the WAI guide: https://developer.mozilla.org/en-US/docs/Learn/Accessibility/Mobile

This page includes a good introduction to Android and iOS specific screen reader apps (TalkBack and VoiceOver, respectively) and global device options.

Things to note

Mouse events like onmousedown are not inherently accessible: you need to implement different but equivalent events. Touch events can be used for touchscreen devices.

Do not disable zoom on mobile apps: never disable using viewport.

> Many people rely on zoom to be able to see the content of your website, so taking this functionality away is a really bad idea. There are certain situations where zooming might break the UI; in such cases, if you feel that you absolutely need to disable zoom, you should provide some other kind of equivalent, such as a control for increasing the text size in a way that doesn't break your UI.

Make sure **"hamburger menus"** and other ways we may "shrink" text for mobile screens remain accessible. A good web example with different menus is here: http://fritz-weisshart.de/meg_men/#

> When implementing such a menu, you need to make sure that the control to reveal it is accessible by appropriate control mechanisms (normally touch for mobile), as discussed in Control mechanisms above, and that the rest of the page is moved out of the way or hidden in some way while the menu is being accessed, to avoid confusion with navigating it.

Minimize the amount of typing required. Mobile devices are much harder to use for inputting data. For us, this should include things like easy ways to set the timer without typing every number or using tedious step arrow keys. Mozilla also recommends providing a **select menu** with common options to minimize typing and reduce errors. This is something we can potentially implement in several places: on alert settings, contacts, and general account settings.

Mozilla on CSS and JS accessibility
https://developer.mozilla.org/en-US/docs/Learn/Accessibility/CSS_and_JavaScript

Important rule: **Do not use CSS to override already-good HTML5 semantics**. Users rely on known styling conventions, and putting too much unique design over the elements or changing expected looks (like for links, forms, etc) can reduce usability and functionality for users.

Color and Contrast

Make sure foreground/text contrasts strongly with background (we have already seen some issues with our contrast checks on the cool tones color scheme).

> When choosing a color scheme for your website, make sure that the text (foreground) color contrasts well with the background color. Your design might look cool, but it is no good if people with visual impairments like color blindness can't read your content.

Another important tip for us: **do not let color be the only conveyer of information.**

> Another tip is to not rely on color alone for signposts/information, as this will be no good for those who can't see the color. Instead of marking required form fields in red, for example, mark them with an asterisk and in red.

High contrast helps all users, not just those with visual impairment. Good contrast makes the pages easier to read outdoors or in brightly-lit environments.

It's okay to "stack" elements over each other, but be wary of hidden elements: **visibility:hidden** and d**isplay:none** hide content from screen readers!

JavaScript risks and guides

Generating pages all from JavaScript can provide opportunity for a lot of accessibility gaps. Mozilla recommends not over-using JS, especially to generate HTML content. Keep JS unobtrusive and **provide alternatives**.

Limit how often you update the UI (or refresh a page) so as not to confuse people using screen readers and other tools.

Double mouse-specific events (for us this may be touch-specific) with **device-independent event handlers**. For example, we want to provide a way to activate the "Text Location" button on the Contacts page with both touch and with straightforward keyboard navigation.

We also want to make sure the double-click on the SOS button is implemented identically with two keyboard or voice-activated “clicks”.

https://developer.mozilla.org/en-US/docs/Learn/Accessibility/HTML#Building_keyboard_accessibility_back_in

Use **meaningful text labels**: for example, we may need to label our home screen button - currently just to logo - with “Home” or a similar label, both in alt text and on screen.

Never include text content inside an image: this is important for our SOS button and potentially the Contacts page. We need to make sure each piece of text is coded as text and fully readable/navigable as such.

Useful Medium post on JavaScript accessibility:
https://medium.com/@matuzo/writing-javascript-with-accessibility-in-mind-a1f6a5f467b9

Takeaways:

As we’ve seen, keyboard navigation and focusable elements is the first most important accessibility function. Some common JS elements can pose issues:

> Elements like **`<p>`, `<h2>` or `<div>` cannot be focused by default**. We often use tags like these to create custom components powered by JavaScript, which might be problematic for keyboard users.

More on the tabindex options we saw from Mozilla:

> It’s possible to make non-focusable elements focusable by adding the `tabindex` attribute with an integer value. If the value is set to `0` the element becomes focusable and reachable via keyboard. If the value is a negative number, the element is focusable (e.g. with JavaScript) as well, but not reachable via keyboard. You can also use a value greater than `0`, but that changes the natural tab order and is considered an anti-pattern.

Focusable elements must be in the **correct DOM order**.

Use buttons for buttons. Don’t try to use generic or custom elements. Use the true button class.

Dynamic Content Changes

Explicitly inform screen readers when content changes without a page refresh. This will be very important for us with functions like the on/off alert list in the Contacts page, the call/text location buttons, and the alert set/disabled features. We will likely want to implement these without refreshing the page, and it is really important to **inform our users of success or failure.**

Use case example:

> Let’s say you have a profile settings page where you’re able to edit personal data and save it. When the save button is clicked changes are saved without reloading the page. An alert informs the user whether the changes where successful or not. This may happen immediately or take some time.

How to implement it:

> By adding a `role` of `status` or `alert` to the message box screen readers will listen for content updates in that element.

```
<div class="message" role="status">Changes saved!</div>
```

> If the text of the message changes the new text will be read out.

Alert Urgency and Interruptions

This is especially important for our app, which has some features that are urgent and immediate, and some that are more routine and administrative.

Status vs. alert:

> The difference between `status` and `alert` is that an `alert` will interrupt the screen reader if it's in the course of announcing something else. `status` will wait until the screen reader has finished announcing.
> Use `alert` only for critical changes. `status` is better in most cases, because it's *politer*.

ARIA live values:

> There’s another attribute called *aria-live*, which can take three possible

values `off`, `polite` or `assertive`. `off` is the default value, `aria-live="polite"` is equivalent to `role="status"` and `aria-live="assertive"` to `role="alert"`.