ARIA Accessibility

ARIA general
https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA

"Accessible Rich Internet Applications **(ARIA)** is a set of attributes that define ways to make web content and web applications (especially those developed with JavaScript) more accessible to people with disabilities...Many of these widgets were later incorporated into HTML5, and **developers should prefer using the correct semantic HTML element over using ARIA**, if such an element exists."

"In HTML5, all ARIA attributes validate. The new landmark elements (<main>, <header>, <nav> etc) have built-in ARIA roles, so there is no need to duplicate them."

From WAI ARIA guidelines, key point:

> "[ARIA code] is a promise that the author of that <div> has also incorporated JavaScript that provides the keyboard interactions expected for a button. Unlike HTML input elements, ARIA roles do not cause browsers to provide keyboard behaviors or styling.
>
> Using a role without fulfilling the promise of that role is similar to making a "Place Order" button that abandons an order and empties the shopping cart."

https://www.w3.org/TR/wai-aria-practices-1.1/#no_aria_better_bad_aria

We have to be very careful using ARIA and not overriding existing HTML5 accessibility functions:

> "This is the power of ARIA. It enables authors to describe nearly any user interface component in ways that assistive technologies can reliably interpret, thus making components accessible to assistive technology users.
>
> This is also the danger of ARIA. Authors can inadvertently override accessibility semantics."

ARIA Mobile?

Important for us! **ARIA is not fully adapted for mobile devices.** This working Google Sheet shows the key functions of ARIA and how they currently do or do not work with various Android and iOS devices and native accessibility features: https://docs.google.com/spreadsheets/d/1gN9oRZPdrJxLDNtT6nVO4fn7E7sn1061L9XI3__sIZ4/edit#gid=0

Applications and widgets
https://developer.mozilla.org/en-US/docs/Web/Accessibility/An_overview_of_accessible_web_applications_and_widgets

The ARIA specification is split up into three different types of attributes: roles, states, and properties.:

1. **Roles** describe widgets that aren't otherwise available in HTML 4, such as sliders, menu bars, tabs, and dialogs.
2. **Properties** describe characteristics of these widgets, such as if they are draggable, have a required element, or have a popup associated with them.
3. **States** describe the current interaction state of an element, informing the assistive technology if it is busy, disabled, selected, or hidden.

Developers should use ARIA states to indicate the state of UI widget elements and use CSS attribute selectors to alter the visual appearance based on the state changes (rather than using script to change a class name on the element).

Keyboard navigation: basic necessities for page elements

Here's a summary of how keyboard navigation should work in an ARIA-enabled web application:

- The Tab key should provide focus to the widget as a whole. For example, tabbing to a menu bar should put focus on the menu's first elem.
- The arrow keys should allow for selection or navigation within the widget. For example, using the left and right arrow keys should move focus to the previous and next menu items.
- When the widget is not inside a form, both the Enter and Spacebar keys should select or activate the control.
- Within a form, the Spacebar key should select or activate the control, while the Enter key should submit the form's default action.
- If in doubt, mimic the standard desktop behavior of the control you are creating.

More details on implementing keyboard navigation for JS widgets: https://developer.mozilla.org/en-US/docs/Web/Accessibility/Keyboard-navigable_JavaScript_widgets

Native HTML elements that are interactive, like <a>, <input> and <select>, are **already accessible by keyboards**, so to use one of them is the fastest path to make components work with keyboards.

Authors can also make a <div> or <span> keyboard accessible by adding a tabindex of 0. This is particularly useful for components that use interactive elements that do not exist in HTML.

Other key points for focus re: our app

1. Ensure that mouse (in our case, touch interaction) works the same as keyboard navigation

2. Use **tabindex** to control how interactive elements receive focus. Tabindex takes integer values: we will largely use -1 or 0 to keep navigation relatively linear. **The SOS button, which is more urgent, may want to have a tabindex of 1** (or any positive number) so that it can be immediately navigable without going through the other page header and menu elements first.

3. We can implement menus and submenus by giving the parent menu a tabindex="0" and its submenus each a tabindex="-1"

https://developer.paciellogroup.com/blog/2013/02/using-wai-aria-landmarks-2013/

Keeping up with HTML5 accessibility improvements and **supplementing gaps with ARIA**:

> "The new sectioning elements in HTML5 have some overlap with ARIA landmark roles, but in a majority of cases there is **no equivalent for the ARIA landmark roles in HTML5**….for example if you want to use the **HTML5 nav element, add role="navigation"** to it, so supporting Assistive Technology (AT) can convey the semantic information to users. When HTML5 elements such as nav are supported by AT, you can then remove the role as it will no longer be required."

Landmarks will be very useful for our app. Important table comparing ARIA roles and HTML5 section options at the bottom of https://developer.paciellogroup.com/blog/2013/02/using-wai-aria-landmarks-2013/.