Screen reader checklist

Accessibility - Screen reader checklist epic ticket: STP-1400 - Accessibility - Screen reader checklist TO DO

Please note

Screen readers need to be told what each element is and if/how the user can interact with it. This is where semantics and ARIA attributes come in. Screen readers use an accessibility tree which is similar to the DOM tree but with fewer nodes. Screen readers need to understand the role, name /label, value, and state for elements (though not every element will have all of these properties/attributes). (WCAG A 1.3.1 and A 1.3.2 and AAA 1.3.6 Only level A and AA compliance is required, level AAA is optional)

Required

1.General*

WG TODO: Viz team to double check priority on lang="en" marker with UW accessibility team, viz team seeing this as a translation bug, not an accessibility bug

- Use **HTML5 semantic elements** as they have built-in ARIA roles (except in older browsers). Here's an exhaustive list of other semantic elements. (WCAG A 4.1.2 and AA 4.1.3)
- Use aria-label, aria-labelledby, or aria-describedby to describe non-text content to screen readers. (WCAG A 1.1.1)
- When using **React**, try to return a semantic element or a React Fragment from your render functions instead of a `<div>` . (WCAG A 1.3.1)
- Web pages have <title> element in <head> that describe topic or purpose. (WCAG A 2.4.2)
- Use headings that describe the topic or purpose of the section. Screen reader users can navigate via headings alone, so ensure the headings make sense out of context. (WCAG AA 2.4.6)
- Use **sequential <h1> <h6> tags**. Don't skip levels. Screen readers offer users the ability to read through and navigate by the headings on the page. (WCAG A 1.3.1 and Hierarchical Heading Organization)
- Use ALL CAPS sparingly; screen readers have difficulty reading it. (Draft recommendation for readers with low vision, not part of WCAG 2.1 AA)

2.ARIA roles and attributes (WCAG A 4.1.2)*

WG TODO: UW Accessibility to suggest a new UI library and viz team to decide on new UI library

- Ensure any focusable item that is not a semantic HTML element has the correct ARIA roles and attributes associated with them
- There are many, many **ARIA roles, attributes, and properties**, and, as the WAI-ARIA Authoring Practices document states, "no ARIA is better than bad ARIA". Confusing screen reader users with incorrect ARIA attributes is very possible. Instead of making custom interactive elements, it's safer to use elements from an established UI library like Material UI or Antd. However, if you do make a custom interactive element, follow the WAI-ARIA design patterns to ensure proper keyboard interaction and proper aria roles, states, and properties.
- If you have the choice between an ARIA attribute or an HTML attribute, **use the HTML attribute**.
- ARIA attributes always need an **explicit value**, `aria-checked="true"` not `aria-checked` .
- **React ARIA attributes** are hyphen-cased as they are in html, `aria-label={labelText}` not camelCased.

3.ARIA live regions (WCAG AA 4.1.3)*

WG TODO: Viz team to see HighCharts uses aria-live to announce data changes with multiple charts

- For dynamic content changes that occur without a page reload, set aria-live regions (see below). Content that changes on screen in single page apps needs to be announced to screen reader users or else they will not be aware it has changed.
- It is most common to announce changes with `aria-live="polite"`. The default is 'off'. Polite won't interrupt the user to announce the change. When set to "assertive", the screen reader will interrupt the user to make the announcement, and is most appropriate to use for an urgent message or error.
- Designate the controls for the changes with `aria-controls=[IDLIST].` Associate a control with the regions that it controls by the region's id.
- Tell the screen reader how much of the change to read with `aria-atomic="false".` This is the default, and only reads the change. When set to true, the entire aria-live section will be read.
- Determine what the screen reader should read with `aria-relevant="additions text".` This is the default. It sets the types of changes that are relevant to a live region, in this case additions and text. You can also add 'removals' to the list or, instead, use 'all'.
- When controls change the content of the chart / view, announce some version of "Data has been updated" on data load
  - For controls that open a second chart on the screen and / or expose new charts or controls, an aria-live should be added to update the user that "[title of the chart now added to the screen" and "[title of chart] chart controls added to the control panel"
  - Aria-live "Data has been updated" announcement for views with multi-charts is non-conclusive for now
  - Edge case for controls that change the content in other controls, low priority for now; experience is the same between a sighted and screen reader user

In React applications:

- Because of how React works with the DOM, changes to the content will only be read out by screen readers if the entire live region already exists in the DOM. This means we cannot render the live region in the DOM for the first time at the moment we want to dispatch a message to the screen reader. On the initial render, the inside text is typically ignored by screen readers.
- Our current solution is to use the React-Aria-Live package to establish and control the update of aria-live regions. (React-Aria-Live uses React Context to communicate changes between the child and parent components to enable screen readers to read out those changes. You can learn more about the inner workings of React-Aria-Live here.)
- The below implementation is from WHO Rehabilitation. It assumes a different message will be sent to the screen reader every time. If you want to announce the same message repeatedly, there is a slightly different implementation that uses LiveMessenger.
- First, install React-Aria-Live: `npm i react-aria-live`

Aria-live regions with react-aria-live package

```
// In your root index file (index.jsx for WHO) wrap the
// app in the LiveAnnouncer component.
// It will render a visually hidden message area in your
// application that can broadcast aria-live messages.
import React from 'react';
import { LiveAnnouncer } from 'react-aria-live';
...
ReactDOM.render(
  <Provider store={store}>
    <ReactQueryConfigProvider config={queryConfig}>
      <LiveAnnouncer>
        <App />
      </LiveAnnouncer>
    </ReactQueryConfigProvider>
  </Provider>,
  document.getElementById('root'),
);

// In any component that renders a change, use LiveMessage
// to send the message you want announced to the screen reader.
// You can use the LiveMessage component to send polite or
// assertive messages. Messages are only triggered when the
// bound message prop changes.
// In WHO rehab, we are announcing the chart title changes
// once a user changes a setting via the controls.
// This is the best we can do currently. Once we figure out
// how to handle the chart content itself for screen readers, we may
// decide to expand the aria-live regions to the chart content.

// Example from ConditionView.jsx
// Visit the live Condition View for reference: http://ihmeuw.org/5cwz
import React, { useCallback, useEffect, useMemo, useState } from 'react';
import { LiveMessage } from 'react-aria-live';
...
// Need local state for aria-live message.
const [ariaLiveMessage, setAriaLiveMessage] = useState('');
// Update aria-live message when title changes due to control changes.
useEffect(() => {
  setAriaLiveMessage(
    `A stacked bar chart that is filterable by condition category, location, sex, measure, metric, and year.
It now shows ${title}`,
  );
}, [title]);
...
return (
    <ViewContainer className={`${classnames.ViewContainer} ${classnames.ConditionView}`}>
    // Render the LiveMessage with updated message prop.
        <LiveMessage aria-live="polite" message={ariaLiveMessage} />
      <ViewTitle ariaLabel={`A stacked bar chart showing ${title}`} title={title} />
      <div ref={parentRef} className={classnames.chartParent} role="graphics-document document">
        {status === 'success' && (
          <>
            <BarChart
...
```

In HTML applications:

Aria-live regions with html

```
// Example where plant info is displayed below the select box
<fieldset>
  <legend>Plant Information</legend>
  <label for="planetSelect">Planet</label>
  // aria-controls designates this select input element as controlling
  // the #planetInfo div.
  <select id="planetSelect" aria-controls="planetInfo">
    <option value="">Select a planet</option>
    <option value="mercury">Mercury</option>
    <option value="earth">Earth</option>
  </select>
  <button id="renderPlanetInfo">Go</button>
</fieldset>

// role designates the div as an aria region. aria-live alerts the
// user that content has changed.
<div role="region" id="planetInfo" aria-live="polite">
  <h2 id="planetTitle">No planet selected</h2>
  <p id="planetDescription">Select a planet</p>
</div>
```

4.Lists

Use lists correctly. (WCAG A 1.3.2 and List tutorial)

- <ul> is an unordered list
- <ol> is an ordered list
- <dl> is a definition list

5.Links

- As much as possible, use <a> for navigation to another webpage and <button> for completing an action.
- Use <a> with an href or <button> with a type and a click handler for links. (WCAG A 2.1.1)
    - If you need to use an <a> more like a button the href should be `href="#0"` and you can add a specific click handler.

Using <a> like a button

```
<a
    aria-label={title}
    as="a"
    href="#0"
    selected={language === id}
    onClick={() => onChange(id, title)}
>
    {title}
</a>
```

- Link text should be **meaningful**; it should explain what the link does. Don't use "learn more" or "click here". (WCAG A 2.4.4 , WCAG A 2.5.3, and WCAG AAA 2.4.9 Only level A and AA compliance is required, level AAA is optional)
- Differentiate links through **more than color**. Use underline, box-shadow, background-color, etc. (WCAG A 1.4.1)

6.Images

- Only use images of text if they are solely decorative. Screen readers can't read the text in images. If you have no alternative, include an aria-label with the text from the image. (WCAG AAA 1.4.9 Only level A and AA compliance is required, level AAA is optional)
- Use alt text, not a caption or title for images, charts, other purely visual content. (You can certainly still caption an image, just also include alt text.) (WCAG A 1.1.1)
  - Use consistent wording in the alternative text for charts and icons if the same type of chart or icon is repeated with in the tool (WCAG AA 3.2.4)
- Explain the content and function of the visual; there is no need to say "picture of..." or "image of...". (WCAG A 1.1.1)
- Purely decorative images should have an empty alt attribute: `alt=""` . (WCAG A 1.1.1, situation F)

Empty alt tag example

```
// This is the IHME logo link that appears at the top left of our viz tools.
// Since we include the aria-label on the <a> the alt for the logo image
// itself can be empty because it is decorative at that point for screen readers.
 <a
    aria-label="Go to the IHME website"
    className="header-logo"
    href="//healthdata.org"
    rel="noopener noreferrer"
    target="_blank"
 >
    <img alt="" src="/vizhub/static/images/ihme-logo.svg" />
 </a>
```

7.Pseudo-elements

An alt tag can be added to CSS generated content for screen readers. The W3 CSS Generated Content Module includes more specific information.

Alt tags in psuedo elements

```
// This is an example from the Evidence Score tool: https://stash.ihme.washington.edu/projects/VIZ/repos
/evidence/browse.
// The text that appears after the "/" in the content property is the alt tag for this ::before psuedo-
element.
// Add whatever text you want the screen reader to read for the psuedo-element as a string following the
slash.
// If it's an empty string (as below), the element is removed from the accessibility tree and not read by
// the screen reader.
.truncate-overflow::before {
  position: absolute;
  content: "..." / "";
  inset-block-end: 0; /* "bottom" */
  inset-inline-end: 0; /* "right" */
}
```

8.Tables (WCAG A 1.3.2)

- Include a <caption> for any data tables.
- Identify all table headers with <th>.

9.Audio/video

- Include a transcript or captions for audio and video. (WCAG A 1.2.2 and A 1.2.3)
- Provide an audio description for all prerecorded video content (WCAG AA 1.2.5)
- Provide an alternative for time-based media for prerecorded audio-only or pre-recorded video-only content (WCAG A 1.2.1)

10.Skip links

- Be sure you have added a skip links link. See Keyboard checklist > Skip links. (WCAG A 2.4.1)

11.Forms and control panels*

WG TODO: Viz team research required 1. to determine the correct semantic elements to use for controls and if the current Container, Label component abstractions will serve us in need for accessibility updates. 2. to ensure AntDesign or whatever component library we use correctly implements the label, input relationship

WG TODO: Lucy to add Jeremy/Melissa discussion outcome about invalid dropdown selections

- Ensure forms have labels, instructions, and validation/error messages. (WCAG A 3.2.2 and 3.3.1 and 3.3.2 and 1.3.3 and AA 1.3.5)
    - Add **labels above** form fields and **instructions below** form fields. Instructions *within* form fields can actually adversely impact form understandability and completion. A form field that is blank is easier for users to see.
    - instructions should describe controls by name not just by appearance or visual location (WCAG A 1.3.3)
- Group together similar elements, like checkboxes or radio buttons, using `<fieldset>` . (WCAG A 3.3.2)
- Elements that have identical functions should have identical text alternatives and labels (WCAG AA 3.2.4)
- Clearly identify required elements either through the html `required` attributed or through `aria-required`. (WCAG A 3.3.2)
- Indicate the purpose of common inputs by using the html `autocomplete, type` attributes or by adding icons. `autocomplete` only accepts a certain number of well-defined fixed values, use that to give more identification to a form field than the `type` attribute where possible (WCAG AA 1.3.5)

Required form inputs with React

```
<label htmlFor="dueDate">Due Date
  <span className="requiredField">(required)</span>
</label>
<input
  type="date"
  id="dueDate"
  name="dueDate"
  // If a more explicit label is required, add it through aria-label
  aria-label="Due date for book return"
  // Ensure screen readers understand this is a required field.
  aria-required="true"
  // If additional information is needed to explain a form input,
  // add a div with that info and then use aria-describedby to link
  // to that div's id.
  aria-describedby="dueDateConstraint"/>
<div id="dueDateConstraint">Due date must be within 6 months of the current date.</div>
```

- Use proper form validation**.** If a form has an error, alert the user immediately with text explaining the error right next to/below the input field, don't rely on color of the form input element alone. (WCAG A 3.3.1)
    - If a form has an error and suggestions for correction are known, then suggestions are provided to the user, unless it would jeopardize the security or purpose of the content (WCAG AA 3.3.3)
- For legal commitments or financial transactions that modify or delete controllable data in data storage systems or that submit user test responses one of the 3 must be true: submissions are reversible, the data is checked for input errors and the user is provided an opportunity to correct them, or a mechanism is available for review, confirming, and correcting info before finalization (WCAG AA 3.3.4)

Add an error message with React

```
<label htmlFor="ccNumber">Credit Card Number
  <span className="requiredField">(required)</span>
</label>
<input
  type="text"
  id="ccNumber"
  name="ccNumber"
  aria-label="Credit card number"
  aria-required="true"
  aria-describedby="ccError"
  aria-invalid="true"/>
// Use CSS to create visual styling that puts the error message
// close to the input and makes it look like an error so that
// sighted users also understand what the message means
<div
  className="errorMessage"
  id="ccError">
  Please enter a valid credit card number
</div>
```

- Making the error element an aria-live region will be helpful for alerting screen reader users to the error. (WCAG AA 4.1.3)
- For **React**, use refs to place focus at the input element where the error occurred when validating forms: (WCAG AAA 3.3.6, partial, Only level A and AA compliance is required, level AAA is optional)
  - Using class components
  - Using the useRef hook
- Use labels for items in an interface or control by pairing an <input> with a <label> or by nesting the <input> within the <label>. For the label of the full control, various semantic elements could be used based on the control(WCAG A 3.3.2 and WCAG A 2.5.3).*

Options for labels

```
// Html version pairing via for/id
<label for="firstName">First Name</label>
<input type="text" name="firstName" id="firstName">

// Or, input nested within label
<label>
   <input type="text" name="lastname" />
   Last Name
</label>

// React version of pairing via htmlFor/id
<label htmlFor="namedInput">Name:</label>
<input id="namedInput" type="text" name="name"/>
```

Testing

Testing*

WG TODO: Viz team to determine fastest development cycle using a PC and notes about setting up a VM below

- Screen reader/browser combination
  - Use a screen reader to test your viz tool. If you have a Windows machine, you can use NVDA on Chrome.
  - **We are investigating setting up virtual machines for everyone that will allow us to test on NVDA on Chrome.**
  - We can also get JAWS license by entering our UW email address at this JAWS license portal, but we do want to focus on only one combination and NVDA/Chrome is a good option for now.
  - For mobile testing, use VoiceOver/Safari on an actual iOS device, the mobile emulator in dev tools is NOT good for accessibility testing.
  - DO NOT worry about testing large screens with VoiceOver/Chrome on our Macs. It's just not a reliable/telling screen reader browser combination and is not highly used.
- Ensure what is read aloud makes sense; adjust your alt tags or aria-labels as needed. Be sure you can interact with all viz controls.
- The screen reader has its own controls (e.g., control+option right/left to navigate and control+option space to select), here are some common keyboard shortcuts. Use these to test. You will find that you can also use the keyboard controls (tab and enter) while using the screen reader, but using the keyboard controls does NOT test the screen reader functionality fully.
- VoiceOver will work with Chrome, Safari, and to some degree Firefox. If you have time, you should test in at least Chrome and Safari.
- See this screen reader > browser compatibility table for more information.
- Please note:
  - You have to turn on tab use in **Safari**. Preferences > Advanced > Accessibility, then check on "Press tab to highlight...".
  - In **Firefox,** you may also need to check off an option to get the tab to work. Preferences > General > Browsing, then check off "Always use the cursor keys to navigate...".
- We can also try testing Windows browsers via LambdaTest.
- You should not expect to see our focus ring when using a screen reader. Generally, a screen reader either adds its own focus ring or add no focus ring at all!

How to use NVDA (for PCs only)

- Download and install NVDA
- The default NVDA key is the `insert key,` which is kind of hard to use on a laptop. From the startup window, you can easily check on the option to use the `caps lock` key instead. **The shortcuts below are written using `caps lock`.**
- Several keyboard shortcuts appear below. You can find more shortcuts here and here
- Start nvda: control + alt + n
- Quit nvda: caps lock + q
- Navigate for reading: down arrow
- Navigate to next focusable area/interactive area: tab
- To click a link: enter
- To click a button: enter or spacebar
- To go to the next heading: h
- To navigate/toggle between radio buttons, select items, tree select items, tab items, menu items: up, down, right, left arrows

How to use VoiceOver

- Ensure VoiceOver is enabled. System Preferences > Accessibility > VoiceOver (from left menu). Check on Enable VoiceOver.
- There are several keyboard shortcuts, here is a list of common VoiceOver shortcuts
- Refresh the webpage, keep your mouse on that page, then press cmd+fn+f5 (if you have a touch bar) or just cmd+f5 (if you don't have a touch bar).
- It will take a couple of long seconds and then VoiceOver will open and begin reading the page.
- (If you want to stop VoiceOver reading at any point, just press control.)
- You can allow the screen reader to automatically read the page. You'll notice a bit more repetition this way than if you purposefully advance through the page (see directions below).
- Press control+option+right arrow to purposefully move the screen reader around the page (left arrow goes backwards).
- Press control+option+space bar to select an interactive element.
- Press control+option+down arrow to move down into a dropdown.
- Though pressing cmd+fn+f5 should turn VoiceOver off, it doesn't always work. In this case, click the close button on the VoiceOver window.

Optional/In development

Chart content

While we have (mostly, depending on the UI component library used) enabled screen reader users to change the chart controls, we don't yet have a solution for giving screen readers access to the chart content itself. It might be best to only provide a data download for screen readers users. However, we might also employ the WAI-ARIA Graphics Module to give access to the chart contents themselves (e.g., the lines or bars and the actual values at each coordinate point).

If we do give access to the chart content itself, we'll need to add aria-live regions so that changes made to the chart content are announced.