Styling checklist

Accessibility - Styling checklist epic ticket: STP-1410 - Accessibility - Styling checklist TO DO

Please note

Style the tool such that it has readable text, allows for resizing, has clear visual focus indicators, and sufficient color contrast.

Required

1. General

- Consistently order navigation that repeats across multiple pages - the locations of the control panels, legends, etc. - to help pages be predictable for users with cognitive limitations, low to no vision and intellectual disabilities (WCAG AA 3.2.3)

2. Viewport

- If possible, **don't restrict the size of the viewport** as it prevents low vision users from zooming. (WCAG A 1.4.4 and Meta viewport rule)
- Don't set `maximum-scale=1` or `user-scalable=no` on the viewport as these prevent low vision users from pinching the screen to zoom in on mobile devices.

Viewport meta tag

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

3. Responsive design*

WG TODO: Viz team to investigate reflow at zoom breakpoints using media queries, and using css to maintain the visualization container size.

- Use **responsive design** as this will help some assistive device users who need to zoom in to see. Their devices will use small screen formatting, but blow the image up to a large screen size. (WCAG AA 1.4.10)
- Allow users to zoom in and out to 400% using browser zoom controls. (WCAG AA 1.4.10)
    - Text should stay in it's container when zoomed to 200%. Use Firefox text-only zoom to test.
    - Visualizations can have 2-dimensional scroll to maintain visibility at zoom
    - All other content must reflow to one dimensional scroll at zoom
- Don't lock content to either portrait or landscape presentation. Website must be functional at both orientations, including mobile. (WCAG AA 1.3.4)
- For **mobile touch targets**, ensure elements are sufficiently large.
    - Minimum target size: 24px (WCAG AA 2.5.8 Only level A and AA are required, level AAA is optional)
    - Target size (preferred): 44px by 44px (set padding to increase target area) (WCAG AAA 2.5.5 Only level A and AA are required, level AAA is optional)

4. Font

- Use IHME standard font as described in our VizHub Style Guide.
- Use a **minimum font size** of 12px, preferably a font size of 14px.
- Use **relative size units** like %, em, or rem, especially for text.
- Try to ensure **font can be increased** by 150-200% without loss of content or functionality. **Use a firefox browser > Zoom > Zoom text only to zoom in to test this.** (WCAG AA 1.4.4)
- Use a **reasonable line length**, maybe somewhere between 75 and 100 characters per line. (WCAG AA 1.4.12 and WCAG AAA 1.4.8 partial Only level A and AA are required, level AAA is optional)
- Ensure that when a user overrides the authored text spacing to the following updates, content or functionality is not lost. **Use your browser devtools in to test this**. (WCAG AA 1.4.12):
    - Line height (line spacing) to at least 1.5 times the font size
    - Spacing following paragraphs to at least 2 times the font size
    - Letter spacing (tracking) to at least 0.12 times the font size
    - Word spacing to at least 0.16 times the font size
- Font treatment is also covered under the Style Guide template ticket STP-964.

5. Video/Animation

- Be wary of animation that has a pulsing or strobing effect as it might cause seizures. No more than 3 flashes per any one second period. Implement prefers-reduced-motion. (WCAG AAA 2.3.2 and WCAG AAA 2.3.3 Only level A and AA are required, level AAA is optional)
- Video or animations longer than 2 seconds must be able to be paused or stopped. (WCAG A 2.2.2)

6.Color*

WG TODO: Check with team if they'd prefer on hover or on click to see where the datum falls on the axes

- Use IHME standard colors as described in our Viz Hub Style Guide.
- Never use **color** alone to convey information. (WCAG A 1.4.1)
- The Design Team has also created color palette guidelines which are very helpful to reference.
- You can check individual color contrast through the WebAIM color contrast checker: https://webaim.org/resources/contrastchecker/

Color contrast

- Use **sufficient color contrast** of at least 3:1 against adjacent colors. An Axe or Lighthouse audit will check individual elements. (WCAG AA 1.4.11)
    - Interface components and meaningful graphics must be distinguishable (this does not apply to inactive user interface components)
    - Examples to look out for: checkmark vs its' filled in background, checkbox border, colors adjacent to a control for each state, how the focus state outline interacts with the control, gradients (hover state is out of scope as long as the hover state styling maintains contrast)
- You can also check individual color combinations with the WebAIM contrast checker.
- For visualizations with various combinations of adjacent colors and non-gradient color scales, create 3:1 contrast by adding a hover/focus feature on the color legend:
    - On hover / focus of a legend item, dull the color of all other viz items corresponding to the non-hovered legend items to a 3:1 contrast of the hovered item color
    - Add instructional text to the legend: "Hover over legend items to isolate values"
    - Ensure the legend is keyboard navigable
    - See more on hover / focus detail requirements in the style guide 5. Update legends ticket. Note: adding checkboxes to the legend is not an accessibility requirement
- For visualizations with various combinations of adjacent colors and gradient color scales, do one of the following:
    - Add a highlight selector that highlights the datum selected and greys all other items, or creates a color contrast with the remaining items on the screen, zoom to the item optional
    - Add a legend, like that in FGH > DAH Comparisons, that allows the user to reset the color scale for gradient colors that are close in value
    - Add a legend, like that in FGH > DAH Comparisons, that allows the user to click or hover on an item and see where on the axis the datum falls
- Target passing at a AA level: (for text WCAG AA 1.4.3 and for non-text AA 1.4.11)
    - Small text, minimum contrast ratio of 4.5:1
    - Large text (18px bold or 24px), minimum contrast ratio of 3:1
    - For graphs: lines should have 3:1 contrast against the background, if there is little overlap of lines, they do not need to contrast with each other or the graduated lines
    - For graphs: avoid thin lines

Colorblind safe colors

- Sources for colorblind safe color palettes.
    - Paul Tol's color pallets for qualitative, diverging, and sequential data.
    - Martin Krsywinski's 12, 15, and 24 color colorblind safe pallets.
    - Color Universal Design's colorblind safe pallet (see figure 16).
- You can also create and test color palettes on Viz Palette or Chroma.
- To check your color palettes, use a browser extension like the Web Disability Simulator to view how different colorblind users will actually see your tool.
- It is also helpful to use a tool like Coolers to select a color palette and then view it through the Web Disability Simulator filters to get an idea if it's acceptable before using it in your data viz.

7.Focus ring (WCAG AA 2.4.7)*

WG TODO: Viz team & C&D to determine if balck white and green are accessible colors, C&D to determine focus states from there to see if C&D for a consistent focus indicator

- The focus ring is what enables keyboard users to know where they are on the page. It is a visual indicator of their cursor's location.
- The focus ring appears on interactive elements.
- If you are adding a custom header link (i.e., a slot element) to the **vizhub template** from the application side, the application will need to have focus-visible set up. See Focus Visible Polyfill below.
- If you can, **style :hover and :focus identically** so that mouse users and keyboard users have a similar experience of visual indicators for interactive elements. If you cannot, use the Focus Visible Polyfill to style keyboard interactions separately from mouse interactions.
- If using a colored outline, ensure the contrast between the focus ring and the item focused is 3:1 (WCAG A 1.4.11)
- Indicate the user's current location in the navigation bar/buttonset through styling. Be sure to use more than just color; add a border, background fill, or underline. Follow the example provided in the Style Guide Epic, STP-1150.

8.Focus Visible Polyfill (WCAG AA 2.4.7)*

WG TODO: Viz team to test :focus:visible class (now supported in all major browsers) and the removal of the package below. Next would be to update the guidelines below.

- Use the **Focus Visible Polyfill** if you need to style :hover and :focus separately.
- Install the focus-visible package in your repo. This package contains a polyfill that allows you to use native browser styles or your own styles to add a visual keyboard :focus indicator **separate** from mouse focus. Basically, you can style the outline (or box shadow) for keyboard users but that style won't apply for mouse users. You will make the tool keyboard accessible without affecting the styling of interactive elements for mouse users.
- For React projects, you will need to use slightly different import technique than described on the npm page. In your root index file, import focus-visible before any other libraries:

Import focus-visible

```
import 'focus-visible';
import React from 'react';
import ReactDOM from 'react-dom';
```

- In the main css file for your app (e.g., App.scss), add focus-visible styles. These particular styles happen to be for the COVID-19 viz tool, so please change them as appropriate for your viz tool.
- You'll probably need to do a little finagling to get the styles to work. (e.g., You may not always need to include .js-focus-visible. Sometimes, you can just target .focus-visible and/or *[data-focus-visible-added].)

Add focus-visible styles

```
/************* Focus styling for keyboard users *************/
  // Hide the focus indicator if the element receives focus via the mouse,
  // but it will still appear on keyboard focus.
  .js-focus-visible :focus:not(.focus-visible) {
    outline: none;
  }

  // Hides focus indicator if element receives focus via the mouse.
  // This is for any frameworks/libraries that overwrite the .focus-visible class name.
  // For example, if you are using the Classnames package, you might need to target
  // data-focus-visible-added instead of .focus-visible.
  // You look for the data attribute 'data-focus-visible-added' instead.
  .js-focus-visible :focus:not([data-focus-visible-added]) {
    outline: none;
  }

  // Define a focus indicator for keyboard focus.
  .js-focus-visible .focus-visible,
  .js-focus-visible *[data-focus-visible-added] {
    outline: 1px solid #72bb99;
  }

  // Here's an example of overriding the focus-visible styling in Ant Design.
  // Styles the location dropdown of the COVID-19 viz tool.
  .js-focus-visible .ant-select-focused[data-focus-visible-added],
  .js-focus-visible .ant-select-focused[data-focus-visible-added] .ant-select-selector,
  .js-focus-visible .ant-dropdown-trigger[data-focus-visible-added] {
    outline: 0;
    border: 1px solid #255024;
  }

  // Here's an example of styling the focus-visible class in Material UI (MUI).
  // Styles the MUI chart tabs on focus-visible.
  // As outline doesn't show entirely because overflow is hidden,
  // turn off outline and use other styles instead.
  .js-focus-visible .Mui-focusVisible {
    outline: 0;
    color: #3E853C;
    background-color: #f5f5f5;
  }
  /*************************/
```

Troubleshooting

1.Styling

- You can try targeting element states by selecting for specific aria attributes.

Selecting for an aria attribute

```
.toggle[aria-pressed="true'] {
  /* styles for mouse and keyboard press */
}
```

2.Focus Visible Polyfill

Shadow DOM

- If your viz tool uses a shadow DOM, you'll need to use this workaround for the focus visible polyfill.
- The polyfill ignores the shadow DOM by default. Learn more about the shadow DOM workaround here.

Shadow DOM workaround

```
// This example is from the vizhub template:
// https://stash.ihme.washington.edu/projects/VIZ/repos/vizhub/browse/src/index.jsx#165
// In the root index file, if the shadowRoot has been
// rendered, apply the polyfill to this shadowRoot.
...
render() {
    const { props, shadowRoot } = this;

        // Workaround for focus visible polyfill
    if (window.applyFocusVisiblePolyfill != null) {
      window.applyFocusVisiblePolyfill(shadowRoot);
    }

    render(
      <ShadowDomContext.Provider value={shadowRoot}>
        <QueryClientProvider client={queryClient}>
          <Template {...props} />
        </QueryClientProvider>
      </ShadowDomContext.Provider>,
      shadowRoot,
    );
}
```

Focus visible isn't applying correctly

Here are a few things to try if you're having difficulty getting focus-visible to apply correctly. These can be especially helpful when using a UI library like Ant Design or Material UI.

- For any element that doesn't receive (or doesn't seem to be receiving focus), add a `tabIndex={0}` prop.
- Sometimes you can't access Ant Design's focus-visible class so you can't apply the focus-visible polyfill. Instead, you can try to create a subtle visual indictor to use on Ant Design's 'xxx-active' class. This styling WILL apply to both keyboard focus and mouse hover, so you do need to ensure it is visible, but isn't overwhelming. So, maybe don't use outline, use background-color, or underline, or box-shadow styles instead.
- For the IHME logo, ensure `display-block` is not applied.
- If you're having trouble getting :focus styles applied to an element, you can try to apply :focus-within to its parent instead.
- If you're using Ant Design, be sure you are targeting the correct :global {} section of the css. Sometimes :global is nested within a specific class name for targeting, so be sure you are where you want to be!

- If your Ant Design dropdown doesn't seem to be accepting focus try wrapping it's clickable content in an <a>, like below, or use a <Button> component for less linting errors:

Antd Dropdown

```
<a
  className={classNames.actionMenuIcon}
  aria-label="Download results, more information, contact, and share menu"
  onClick={(e) => e.preventDefault()}
  href="#0"
  tabIndex={0}
>
```

- To get **Ant Design ToolTips (info hovers)** to open on hover and focus, you need to: pass an array of triggers ['focus' and 'hover'] to the trigger prop, and be sure the icon component is given focus through tabIndex. (WCAG AA 1.4.13)

Antd Tooltips/Info Hovers

```
<Tooltip
   title={TOOLTIP_CONTENT}
   placement="bottomRight"
   arrowPointAtCenter
   trigger={['hover', 'focus']}
>
   <InfoCircleOutlined
       className={styles.infoIcon}
       onMouseEnter={() => trackInfoHover(ANALYTICS_LABEL)}
       onClick={() => trackInfoHover(ANALYTICS_LABEL)}
       tabIndex={0}
   />
</Tooltip>
```

- To get **Ant Design's Popover** to open on focus, you need to: add the getPopupContainer prop with the appropriate trigger node (more about nodes), and kind of do a bit of an ugly hack on the title prop to allow for keyboard access.

Antd Popover

```
<Popover
     arrowPointAtCenter
     content={<ChartSettingsContent {...props} />}
     title={
       <>
         {/* Setting tabIndex allows keyboard access to chart settings. */}
         <span tabIndex={0}>Chart settings</span>
       </>
     }
     placement="bottomRight"
     onVisibleChange={handlePopoverVisibleChange}
     trigger="click"
     // Allows keyboard access to the popover.
         // Depending on the DOM, you may need to try different trigger nodes, like .parentNode
     getPopupContainer={(triggerNode) => triggerNode.nextSibling}
   >
     <Tooltip placement="left" title="Chart settings" visible={hovered && !popoverVisible}>
       <div
         className={styles.chartSettingsButtonWrapper}
         ref={buttonRef}
         onMouseOver={handleMouseOver}
         onFocus={handleMouseOver}
         onMouseLeave={handleMouseLeave}
       >
         <Button
           className={`${styles.buttonWithoutAnimation} ${styles.settingsButton}`}
           size="large"
           aria-label="Chart settings"
           icon={<FontAwesomeIcon icon={faSlidersH} />}
           onClick={() => trackChartSettings(`${analyticsLabel} Clicked chart settings menu`)}
         />
       </div>
     </Tooltip>
   </Popover>
```

- To get an **Ant Design Dropdown with a Tooltip inside of it** to show the Tooltip on hover/focus and open on enter/click, you need to: surround the icon that activates the Dropdown and Tooltip with an Ant Design Button component, and pass an array of ['hover', 'focus'] to trigger on the Tooltip.

Antd Dropdown with Tooltip

```
<div ref={containerRef} className={classNames.geoLocation}>
     <Dropdown
       trigger="click"
       overlay={menu}
       getPopupContainer={getPopupContainer}
       onVisibleChange={handleVisibleChange}
     >
       <Tooltip
         getPopupContainer={getPopupContainer}
         placement="bottom"
         title="Use geolocation to find available locations"
         // Pass an array of hover and focus to the trigger prop.
                 trigger={['hover', 'focus']}
         {...(dropdownVisible ? { visible: false } : {})}
       >
         // Surround the icon with a Button element, this will allow the focus-visible
                 // class to be attached to the element and ensure keyboard functionality.
                 // (You may need to rework the button styling in your css file as well.)
                 // If you need to position the icon, wrap it in the Button (as shown below) instead
                 // of using the Button's icon prop.
                 <Button
           className={classNames.geolocationButton}
           size="large"
           aria-label="Use geolocation to find available locations"
         >
           <FontAwesomeIcon icon={faLocationArrow} />
         </Button>
       </Tooltip>
     </Dropdown>
   </div>
```

- To ensure **Ant Design's Radio (buttonsets)** are treated separately by the browser, you need to pass a name prop to the Radio.Group (RadioGroup). This will ensure that keyboard and screen readers see each buttonset as unique and separate from other buttonsets on the page. Without this name prop, the browser will see all of the buttonsets in the viz as part of a single large buttonset.

Antd Radio/Buttonset

```
// In the Buttonset UI component, include the name prop for each Radio.Group (RadioGroup in this case).
import { Radio } from 'antd';
...
const { Button: RadioButton, Group: RadioGroup } = Radio;
...
export default function Buttonset(props) {
  const {
    autoSize,
    buttonStyle,
    label,
    labelKey,
    name, // name prop will ensure each button set is recognized as unique by the browser
    onChange,
    options,
    style,
    valueKey,
    ...restProps
  } = props;

return (
    <Container label={label} style={pick(style, ['container', 'label'])}>
      <RadioGroup
        {...restProps}
        buttonStyle={buttonStyle}
        className={classnames.buttonset}
        name={name} // name prop will ensure each button set is recognized as unique by the browser
        style={style.buttonset}
        onChange={handleChange}
      >
        {buttonsetOptions}
      </RadioGroup>
    </Container>
  );
}

// Where each Buttonset is called, pass an appropriate and unique name prop
export default function MetricControl() {
...
  return (
    <Buttonset
      autoSize
      label="Metric"
      labelKey="name"
      name="metricControl" // pass a unique name prop
      options={options}
      value={value}
      valueKey="metric_id"
      onChange={handleChange}
    />
  );
}
```

Optional/In development

- Evan Laurie started development on using patterns along with colors to make our visualizations more accessible for colorblind users. Here's a link to his patterns branch in WHO Rehabilitation. (WCAG A 1.4.1 and Color and Pattern Rule)
- We are looking at adding a control to toggle on/off colorblind safe colors/patterns. This might be part of the vizhub template or it might be part of individual applications.