



Accessibility Guide for Interactive Web Maps

October 2024

Contents

Contents	2
Section 1: Summary and Key Takeaways.....	3
Summary	3
Key Takeaways	5
Section 2: Checklist	6
General Web Accessibility	6
Responsive Design and Reflow.....	6
Map Controls.....	6
Map Content	7
Map Symbology.....	8
Map Symbology Labels.....	9
Map Elements and Color (e.g., Buttons)	10
Forms and Labels.....	10
Context Changes.....	11
Testing the Visible Focus	11
Map Popups.....	12
Section 3: Solutions.....	13
General Solutions	13
Interactive Web Map Solutions.....	15
Embedded Web Map Solutions.....	17
Section 4: Accessible Examples	19
Interactive Web Maps (Static).....	19
Interactive Web Maps (Complex & Wayfinding).....	23
Section 5: Technology Recommendations	36
General Web Recommendations	36
ArcGIS Maps SDK for JavaScript	38
Leaflet Library.....	41
Mapbox GL Library	42
Section 6: Testing.....	43
Tools	43
Exercises and simulations.....	45
Section 7: Conclusion	47

Section 1: Summary and Key Takeaways

Summary

This document provides interactive web map designers and developers strategic and tactical guidance on accessibility best practices and standards for custom-built interactive web maps.

Interactive web maps allow a user to interact with a map using a mouse, keyboard, voice, or other mechanism, to select areas of interest, enter coordinates, or toggle layers on and off. The user can typically zoom in and out and pan around the map while data changes. Interactive web maps can be static or complex in nature.

The map's purpose and content determine whether it should be interactive, and if so, drives the requirements to make an interactive web map accessible. What works for one application may not work for another. When creating an interactive web map, ensure that all users can access its content.

By considering accessibility from the start, an interactive web map has a better opportunity to serve all audiences. Creating accessible information should not be an exception to the rule; it should be there when people need it, not by request.

The "Accessible Examples" section of this document contains examples of how accessible components differ between each map, and how they are based on the subject matter and functionality built into the interactive web mapping application.

State of Minnesota Statute and United States Federal Law

By statute, the [State of Minnesota's accessibility standard](#) references the Federal [Section 508](#) and World Wide Web Consortium's [Web Content Accessibility Guidelines \(WCAG\) 2.0, levels A and AA](#). Section 508 is driven by a federal law that requires electronic and information technology that is developed, procured, maintained, or used by the federal government to be accessible to people with disabilities. In July 1, 2024, the accessibility standard updated to incorporate [WCAG 2.1](#).

Key Takeaways

Throughout this document are comprehensive guidelines to making accessible interactive web maps. The following list includes some of the most important generalized guidelines for designers and developers to follow.

- **Integrate color contrast** ratios of at least 4.5:1 for normal text and 3:1 for large text (WCAG 2.1: 1.4.3 Contrast, Minimum).
- **Test without the mouse** and only using your keyboard with your interactive web map (WCAG 2.1: 2.1 Keyboard Accessible).
- **Avoid keyboard traps** such as a pop-up dialog that prevents the user from entering and interacting with its contents via a keyboard (WCAG 2.1: 2.1.2 No Keyboard Trap).
- **Use color and texture** to distinguish elements in your interactive web mapping application.
- **Organize the reading order** of elements in your interactive web mapping application into a flow that makes sense.
- **Format the text** content and length for readability.
- **Design for reflow** so the content is perceivable and operable when a browser is zoomed (WCAG 2.1: 1.4.10 Reflow and 1.4.12 Text Spacing).
- **Recognize technological constraints** that impact the delivery of information, such as a framework that doesn't automatically add <alt> text to basemap tiles. This can enable you to provide additional content, or let the user know of any constraints when accessing your interactive web map.

More detailed guidelines for the listed takeaways above can be found in the following "Checklist" section. The remaining sections will provide a more in-depth look into interactive web map accessibility and the tools available to meet accessibility standards.

Section 2: Checklist

General Web Accessibility

- ✓ Visit the [WCAG 2.1 quick reference guide](#).
 - In addition, visit [Web accessibility for developers](#). The supplemental checklist provides a level A (beginner web accessibility features), level AA (biggest and most common barriers for disabled users), and level AAA (the highest and most complex level of web accessibility) WCAG 2.1-specific checklist.
- ✓ Visit the [State of Minnesota Office of Accessibility Web and Apps page](#) for further information and resources on web/application accessibility. This page includes links to guidelines, training resources, checklists, and tools.
- ✓ A few more introductory resources are available here:
 - [A11y Project's Web Accessibility Checklist](#)
 - World Wide Web Consortium ([W3C](#)):
 - [Introduction to Web Accessibility](#)
 - [Tips and tutorials](#) for writing, designing, and developing for web accessibility.
 - [Accessibility Standards Overview](#)
- ✓ The map's essential information and purpose must be clearly explained in details of the map's content when a user visits the page. This is also a good location to share accessible limitations and contact information to obtain content in a differing accessible format.

Responsive Design and Reflow

- ✓ Orientation: Do not lock maps (or other content) to either portrait or landscape presentation. Remember some users will have devices with fixed orientation and others will want to change the orientation to improve visibility. Allow the user to choose. Do not force one particular orientation unless it is essential.
- ✓ Reflow: Set your browser to 1280 pixels wide and 1024 pixels high. Zoom the browser (not the map) to 400%. All functionality of your application should still be operable, and all content should still be accessible.
- ✓ Text spacing: Ensure that content is still available, legible, and functional if users increase the text spacing of the web map application.

Map Controls

- ✓ Navigation controls (+/-): If navigation controls such as zoom in (+) or zoom out (-) buttons are on your page, ensure they are keyboard accessible and fit the logical tab order of the page.

- ✓ Map navigation: Map navigation should be possible without the use of a mouse. For instance, the user can use keyboard navigation to zoom in, zoom out, and pan in the map. Consider also enabling a map click via keyboard, which can be accomplished via a crosshair graphic and fetching the map center for the location of the click event.
- ✓ Multipoint gestures: If map controls default to requiring multipoint (more than one finger) gestures, provide single-point (one finger) alternatives. An example of a multipoint gesture would be pinch zooming. Provide buttons for zoom in/out and/or single-finger gestures such as double tap as alternatives to pinch zooming.
- ✓ Path-based gestures: If map controls default to requiring path-based gestures, provide non-path-based alternatives if possible. A path-based gesture is when points between the start and end of the gesture matter. Freehand sketching a line or polygon is a path-based gesture. Ensure that freehand sketching is not the only option available for specific tasks, such as creating features.
- ✓ Motion actuation: Control of content in an application should not be solely dependent on device motion such as shaking or tilting the device or user motion such as gesturing towards the device. Content can be controlled by movement of users through space as registered by geolocation sensors such as GPS.
- ✓ Buttons: Like navigation controls, ensure buttons/widgets are keyboard accessible, fit the logical tab order of the page, and have sufficient color contrast between the button's text and background.
- ✓ Hover and focus: If a web application displays additional content when an element receives keyboard focus or mouse hover, ensure that these three criteria are met:
 - Content can be dismissed by pressing the Escape key.
 - Content is hoverable.
 - Content is visible until the user moves the hover, focuses on a different element, or dismisses the content.
- ✓ Key shortcuts: Character key shortcuts should only be active when the application has focus, require the use of the Ctrl key, and can be turned off.
- ✓ Pointer cancellation: Actions should only fire on key-up events unless a key-down event is reversed by the next key-up event or the action is reversible by other means. There should not be any unintended irreversible actions based solely on a pointer down-event.

Map Content

- ✓ Headings: If content is added to your map, ensure the logical order on the page goes from Heading Level 1 <h1> to Heading Level 2 <h2>, then Heading Level 3 <h3>, etc.
- ✓ Links: Links must be recognizable and differentiated from other text content in the page (e.g., underlined text, or using another method other than color alone). In addition, all links should be focusable or tabbable when navigating through the map application with a keyboard.

- ✓ Reading order: Ensure the map application's content and design has a logical structure. To check the reading order, tab through the application after loading it to see where the browser's focus is directed. The order should have the same flow as seen visually, particularly with forms and search dialogs.
- ✓ Plain language: If there are components of the map that describe the application in detail, it needs to be clear and concise, without any use of jargon or undefined acronyms.
- ✓ Color contrast: All components of the map application must meet the following contrast ratios defined by WCAG 2.1: 1.4.3 Contrast (Minimum):

Font type	WCAG 2.1 Level AA (Minimum)	WCAG 2.1 Level AAA
Normal text (< 14 -pt.font)	4.5:1	7:1
Large text (14-pt. font and larger)	3:1	4.5:1

- ✓ Paragraph line length: Paragraph line length must be considered; either too short or too long in length can be confusing to both sighted and non-sighted users. Ensure you keep text content concise but distinguishable. Consider using the CSS max-width property on containers.

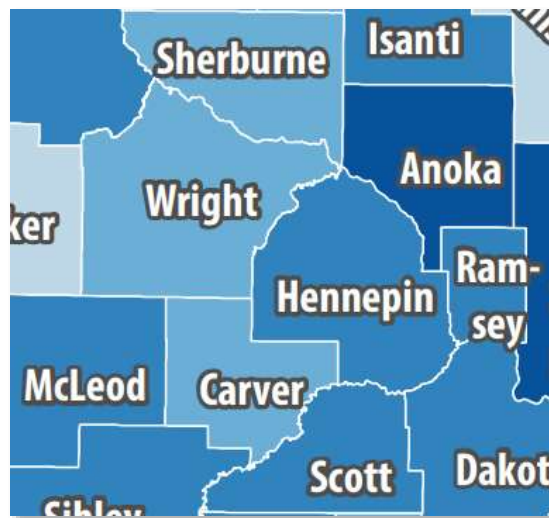
Map Symbology

- ✓ Consider using textures if contrast among map layers is difficult to achieve.
- ✓ Use a color contrast analyzer to evaluate the map symbology to verify that the points or features of interest have enough contrast compared to other maps contents. Keep in mind that individuals with low vision may use magnification, so test accordingly.
- ✓ Colors must be defined clearly in alternative text and with a legend/key.
- ✓ Consider using powerful colors for conveying important information.
- ✓ Select a color palette that does not have extreme color changes, as this can cause sensory overload for those with autism spectrum conditions and synesthesia disorders.
- ✓ If symbology is important to the map, a legend should either be included in the mapping application, or adjacent to the map (if the map is embedded on a webpage) so users can verify the information and symbology. It is strongly recommended the map legend be visible when the map loads, especially when

viewing on a desktop or large tablet device. The legend should be easy to access if hidden or disabled in the map view at any time, including views on mobile devices.

Map Symbology Labels

- ✓ Verify the color contrast of the label text with the underlying map symbology color.
- ✓ Analyzing map labels for contrast:
 - If color contrast cannot be met with one color, **try to combine two colors together using a halo** around the label's text. For example: Use white text with a black halo. The combined colors could meet ratios, where used alone they may not.



- ✓ The table below shows background colors with contrasting text, contrast ratios, and WCAG 2.1 AAA ratings for the ratios. Developers are encouraged to test failing combinations with color analysis tools to visualize the low contrast results.

Rating and Ratio for Black Text Over Blue Shade	Rating and Ratio for White Text Over Blue Shade	Demonstration Text Over Blue Shades
Pass (14:1)	Fail (1.5:1)	#000 over #BDD7E7
Pass (8.7:1)	Fail (2.4:1)	#000 over #6BAED6
Pass (5.1:1)	Fail (4.2:1)	#000 over #3182BD
Fail (2.7:1)	Pass (7.9:1)	#FFF over #08519C

Map Elements and Color (e.g., Buttons)

- ✓ Colors cannot be used alone to depict important information.
- ✓ Consider using textures if contrast among many map elements is difficult to achieve.
- ✓ Use a color contrast analyzer to evaluate colors. This helps verify that the items of interest have enough contrast compared to other elements in the mapping application. Keep in mind that individuals with low vision may use magnification, so test accordingly.
- ✓ Ensure user interface elements (buttons, form inputs, etc.) and graphical elements (charts, graphs, etc.) have a 3:1 contrast ratio where possible. If it is not possible to have a 3:1 contrast ratio, consider including borders, spacing, or halos to separate similar colors.

Forms and Labels

- ✓ Whenever possible, use the label element to associate text with form elements explicitly. The “for” attribute of the label must exactly match the id of the form control it is associated with.

```
<!--Label example-->  
<label for="mapSearch">Search in the map</label>  
<!--Search input-->  
<input type="text" id="mapSearch"></input>
```

- ✓ Visually position labels to the right of radio buttons and checkboxes, and to the left of, or directly above other form fields. Maintaining this practice increases predictability and understandability of your form for all users.
- ✓ For button elements, the label is set inside the element and can include markup. When using the **input** element to create buttons, the label is set in the **value** attribute of the element.

```
<!--Either button example below fits accessibility-->  
<button type="submit">Submit</button>  
<input type="submit" value="Submit">
```

- ✓ When programmatically identifying input purposes for form fields, use [common input purposes](#) only for their intended purpose. Good example:

```
<label for="fullName">Full Name:</label>  
<input name="fullName" id="fullName" type="text" autocomplete="name"/>
```

Bad example:

```
<label for="placeName">Place Name:</label>  
<input name="placeName" id="placeName" type="text" autocomplete="name"/>
```

Using `autocomplete="name"` for a Place Name input would result in assistive technologies (such as autofill) interpreting that a person’s full name is supposed to be entered.

Maps developers in particular will want to pay attention to the common input purpose terms for address elements, since address input and geocoding is often used in mapping applications.

- ✓ Ensure that controls, such as checkbox toggles for a table of contents (TOC) or legend/TOC, have labels that match or are contained within the text. For example, it would be confusing to have a programmatic/accessible label say "overlay1" when the text label says "watersheds."
- ✓ View more explicit guidelines and examples on the [w3.org website](https://www.w3.org).

Context Changes

- ✓ Look for items that require a change of context and ensure the content change is clear to sighted and non-sighted users. For example, a map popup that changes its content when an item is selected. Using the aria-live attribute ensures that when the content is updated inside the element, those changes are communicated to users of assistive technology. [View more examples for using aria-live](#).

```
<!--Context change example for a popup using the ARIA-live attribute-->  
<div id="popup" aria-live="assertive">  
<p>Map item description</p>  
</div>
```

- ✓ Ask assistive technology to announce the dynamic changes using roles, such as error or warning messages that appear on the screen. Alerts are particularly important for client-side validation notices to users.

```
<!--ARIA alert role-->  
<h2 role="alert">  
  The form was not be submitted because the "name" field is blank.  
</h2>
```

Testing the Visible Focus

- ✓ Tab through the map/page to verify all interactive web map elements can receive focus and focus is visually indicated to the user.
- ✓ The visual focus order should match the intended interaction order.
- ✓ Interact with all controls, links, and menus using only the keyboard.
- ✓ Do not "trap" keyboard focus within subsections, except for modal dialogs.
- ✓ Keyboard focus should stay inside a modal dialog until it is dismissed. Once the modal dialog is dismissed, the keyboard focus should be restored to the previously focused element, e.g. [Modal dialog example](#) (WCAG 2.1: 2.1.2 No Keyboard Trap).

- ✓ Off-screen content, such as responsive navigation, should not receive focus when it is not displayed. Do not hide content by positioning off-screen or behind other items. Use `display:none` in CSS to hide content from **all users**.

```
<!--Off screen content should use display:none to hide content from all users-->  
<div id="hiddenContentExample" style="display:none;">  
<!--Hidden content to all users-->  
</div>
```

- ✓ After interactions with controls, ensure that focus is set in a logical order or set to an element that contains the results created by the interaction. Example: When a search query is run and a data table is populated with the results, focus should be set to the results in that data table.

Map Popups

- ✓ Can a user access the popup content without using a mouse?
- ✓ Can a user tab through and access all the popup content when it is displayed on the screen?
- ✓ Does assistive technology know when a popup has opened and is displayed on the screen?
 - Use either `aria-live=assertive` **or** `role=alert` to interrupt anything assistive technology is currently reading or must read. This way the user knows a popup has been displayed in the map. Be careful not to add *both* `aria-live="assertive"` *and* `role="alert"`, as this will cause double speaking in VoiceOver and iOS. See use case examples on the [W3C website](#) (WCAG 2.1: 3.3.1 Error Identification). After reading the popup to the user, assistive technology will continue reading where it left off.

Section 3: Solutions

Address the key components from the law on accessibility and work your map around those components. For example, keep in mind keyboard-only navigation while developing the map application, so a user can navigate through the application using navigation keys. Do what you can to make the interactive web map accessible to the best of your ability.

General Solutions

1. [How to Meet WCAG 2.1](#), a quick reference using the four POUR principles of:

- **Perceivable:** information and user interface components must be presentable to users in ways they can perceive;
- **Operable:** user interface components and navigation must be operable;
- **Understandable:** information and the operation of user interface must be understandable; and
- **Robust:** content must be robust enough that it can be interpreted reliably by a wide variety of user agents, including assistive technologies.

2. Focus on the map's intent/purpose.

- Ask what the intent or purpose is, then focus on delivering key elements when designing your interactive web map.
- For example, if you are creating an interactive web map to display Minnesota state parks, you could provide a visual of where the state parks are located around the state in a map and provide a list of Minnesota state parks.



- Focus on the map delivery, and don't add in every component or map control available to you. Just because you *can* add in components, controls, and layers, doesn't mean you *should*. Provide only the basic elements needed for the map and its message.
3. Start with foundational cartography principles, such as [Ordnance Survey's Cartographic design principles](#):
- Understand user requirements.
 - Consider the display format.

- Create a clear visual hierarchy.
 - Focus on simplicity.
 - Ensure legibility.
 - Consider consistency.
 - Ensure accessibility.
 - Maintain good composition.
4. Use color and texture (WCAG 2.1: 1.3.1 Info and Relationships; WCAG 2.1: 1.4.1 Use of Color).
 - Provide good color contrast through navigation elements, buttons, links, text, and the map symbology (WCAG 2.1: 1.4.1 Use of Color; WCAG 2.1: 1.4.3 Contrast, Minimum; WCAG 2.1: 1.4.6 Contrast, Enhanced).
 - Don't rely on color alone for functionality or conveying information, as users may be unable to distinguish them, or may override page colors. Color cannot be the only way information is conveyed (WCAG 2.1: 1.3.1 Info and Relationships; and WCAG 2.1: 1.4.1 Use of Color).
 5. Do not assume your user is a power user.
 - Focus on plain language and a basic level of understanding to provide a better user experience and more accessible format to all users.
 6. Use the best available tools or technology to get your message across. This may not necessarily be the latest version available.
 - For an example of an improvement when moving to the latest version, upgrading from the Esri JavaScript API 3.x series to the 4.x series can add out-of-the-box assistive technology support.
 - For an example of a difficulty when moving to the latest version, upgrading the Esri Maps SDK for JavaScript from 4.27 to 4.28 made focusing the title of a popup more complicated.
 7. Recognize technological constraints.
 - For example, assistive technology reads the document object model (DOM), making it important for developers to use semantic HTML elements and other code such as JavaScript and/or ARIA to describe what is taking place on the page/map as much as possible.
 - Don't assume libraries will be well designed for accessibility. Test, test, and test again! While libraries such as the Esri Maps SDK for JavaScript or Angular Materials can be very helpful, they can also cause accessibility problems that you may have to work around. For example, the Angular Materials component for a Select dropdown [does not perform well](#) for accessibility when the "multiple" property is set. When the items in the list are focused, there is no notice of whether items are selected or not.
 8. Design for layouts to reflow:
 - Avoid fixed height and width by using relative CSS measurements such as percentages and view height/width.

- Map containers may not respond well to height as a percentage; the CSS `calc()` function combined with view height can be a good solution if your page does not scroll vertically. For example, if your page has a 90 pixel header, set the map container height as: `height: calc(100vh - 90px)`.
- For narrative content, do not let your container stretch too wide. CSS `max-width` can be helpful.
- A combination of CSS grid (or other layout libraries) and `@media` queries can be used to rearrange layout elements for different screen sizes and orientations.

Interactive Web Map Solutions

1. Consider whether the functional requirements of the application can be met without an interactive web map and, if so, consider incorporating the ability to use the application with or without the map present.
 - For example: See the [Order Help Me Grow Materials](#) accessible example.
2. Add a splash screen with text when the map loads alerting the user of potential constraints, and who to contact for alternative products (WCAG 2.1: 2.4 Navigable).
3. Consider the reading order of the map and how users will navigate through the map (WCAG 2.1: 3.2.3 Consistent Navigation):
 - Some users will leverage keyboard-only tools, such as the arrow keys, tab, spacebar, and/or enter keys (WCAG 2.1: 2.1.1 Keyboard), to navigate the application.
 - A `tabindex` of 0 allows assistive technology to access `<div>` elements. Try to avoid any `tabindex` above 0, as it can interfere with the order of elements on your page, or map.
 - The order of elements should be in the same order as they appear on the screen as they are in your DOM/HTML. (WCAG 2.1: 1.3.2 Meaningful Sequence)
 - Consider element sizing (e.g. buttons, lists, etc.). The recommended minimum font size is 12-pt for readability.
4. Consider the text layout in the navigation elements, buttons, and body text (WCAG 2.1: 1.3.2 Meaningful Sequence) including:
 - Use true text (HTML text), where possible. True text enlarges better, loads faster, and is easier to translate. For visual styling, use CSS.
 - Try to stay away from using ALL CAPS. All caps can be difficult to read and can be read incorrectly or misinterpreted by assistive technology (WCAG 2.1: 1.4.8 Visual Presentation; WCAG 2.1: 3.1 Readable).
 - Use adequate font size, as sizing can vary based on the font chosen and the device. Twelve-point (12 pt.) font, or 1em, is the recommended minimum font size (WCAG 2.1: 1.4.4 Resize text; WCAG 2.1: 1.4.8 Visual Presentation).
 - Test text layout and contrast under magnification to ensure that it is readable (WCAG 2.1: 3.1 Readable), including icons and/or legends depicted in an image format.
5. Consider the text layout, including:

- Use clear semantics where content should have good semantic structure (WCAG 2.1: 1.3.1 Info and Relationships).

```

<!--Example where the role is specified so assistive technology
knows the function of the content-->
<li tabindex="0" class="checkbox" role="checkbox" checked aria-
checked="true">
Correct example of a checkbox as a list item
</li>

```

- Ensuring a link’s text makes sense without supporting text. Avoid using “click here,” “more,” or “continue”, and instead use phrases related to the link, such as “Learn more about map accessibility” (WCAG 2.1: 2.4.4 Link Purpose, In Context; WCAG 2.1: 2.4.9 Link Purpose, Link Only).
 - Remember line length; don’t make lines of copy too long or too short (WCAG 2.1: 1.4.8 Visual Presentation).
 - Make links recognizable by differentiating them in the body of the page with underlines and/or a method besides color (WCAG 2.1: 1.4.1 Use of Color; WCAG 2.1: 1.4.8 Visual Presentation; WCAG 2.1: 2.4 Navigable).
 - A visible focus should be obvious for all focusable elements on the page, including buttons. For example, once focused, a link could have a dotted border, background, or a non-color designator (WCAG 2.1: 1.4.1 Use of Color; WCAG 2.1: 2.4.7 Focus Visible).
 - Design accessible form controls by ensuring descriptive labels and instructions are included. Pay close attention to form validation errors and recovery mechanisms (WCAG 2.1: 3.3.2 Labels or Instructions; WCAG 2.1: 3.3.1 Error Identification; WCAG 2.1: 3.3.3 Error Suggestion; WCAG 2.1: 3.3.5 Help).
6. Design a “Skip to” content link, or bypass block, for keyboard users to skip navigation, the map, and the map navigation controls or to skip to the beginning and end of lengthy lists. The “Skip to” content link should navigate keyboard users to the content that makes up your map and gives users the ability to get to the purpose without having to navigate through all of the map (WCAG 2.1: 2.4.1).
- While a “Skip to table” / “Skip to data” option is handy, if you provide a table, it is useful to all audiences, and can be a supplemental piece of your interactive web map application, even when the map is the main focus.
 - You can bypass navigation and the map using “Skip to content.” You could also include a “Skip to table,” “Skip to data,” etc. where a data table is provided with the map.
 - If selected, the “Skip to” link takes the user to the next focusable element in the page.
7. Avoid blinking, flashing, or strobing content or colors as they can cause seizures (WCAG 2.1: 2.3 Seizures) and be difficult for those on the Autistic spectrum (WCAG 2.1: 2.2.2 Pause, Stop, Hide). Be wary of libraries and functions, especially when working with WebGL and/or 3D, that can cause jerking or rapid movements. If such movements are necessary, do not enable them by default and try to slow

down movements so users can adjust to the changes. Examples: [Uber traffic patterns, video](#) and [Esri JS v4 Web Scene slide tour](#).

8. [Accessible Rich Internet Applications](#) (ARIA) [live regions](#) provide a way to programmatically expose dynamic content changes in a way that can be announced by assistive technology. However, use ARIA sparingly. If you can perform the accessible component with HTML, do that first, and ARIA second.
 - aria-live region – used to alert the assistive technology user to a change in content, for example a popup window. In the example below, using the assertive option of the aria-live attribute will immediately read the updated contents of the region to the user whenever the region’s content changes.

```
<!--ARIA: aria-live region example-->
<div role="region" id="info" aria-live="assertive">
  <p id="description">Select to view its description</p>
</div>
```
 - aria-describedby attribute - used to indicate the IDs of the elements that describe the object. It is used to establish a relationship between widgets or groups and text that described them. This is very similar to [aria-labelledby](#): a label describes the essence of an object, while a description provides more information that the user might need.
 - [View more practical ARIA examples](#)
9. Try listening to your maps using a screen reader. See [this slide book created by Jason Ewert](#) to learn more about how to announce map operations and content via a screen reader.
10. Apply static map techniques, where possible.

Embedded Web Map Solutions

1. The embedded web map, dashboard, etc. solution must meet the same accessibility standards defined in the [Interactive Web Map Solutions section](#).
 - If the solution is not accessible it is recommended to provide alternative options for all users (e.g., a table, a static image with alt text, etc.).
 - To achieve usability across all audiences, focus on the solution’s purpose and provide all users with opportunities to get at the same content the solution aims to achieve.
2. Accompany the map with alternative options, such as a table, where possible.
3. If a known solution is not accessible, you can embed the solution with a tabindex value of -1 so screen readers and users that tab cannot access the product (e.g., Esri Story Maps, Esri Dashboard, etc.).
 - **Note:** If you do implement this as a solution you **must** provide alternatives to the content you are not showing to screen readers and users who leverage tab.

```
<!-- Map hidden from screen readers and users who tab -->
```

```
<!-- Note: You MUST provide alternatives to access the content you are not showing -->
<div>
  <div id="embedded-map" tabindex="-1"></div>
  <table id="embedded-map-table"></table>
</div>
```

4. Inline frames, or iframes, allow full sized applications or web sites to be within a sub-window of a parent web page. The iframe may be used to embed a map or map solution within a webpage. There are no distinct accessibility issues with inline frames.
- Some users enlarge fonts and page elements, so you should **not disable scrolling** for iframes by using `scrolling="no"`.
 - You can provide a title to list out the purpose of the content in an iframe, such as “embedded web map of Minnesota’s state parks”, like so:

```
<!-- Inline frame (iframe) example -->
<iframe src="embedded-webmap-url.html" title="Embedded web map of Minnesota's
state parks"></div>
```

Section 4: Accessible Examples

Interactive Web Maps (Static)

Order Help Me Grow Materials – Minnesota Department of Education

Introduction

Allow educators to obtain contact information to receive Help Me Grow outreach materials to share with parents. The Help Me Grow Program helps with young children's development, learning, and growing by developing milestones and encouraging healthy development.

[Access the interactive web mapping application.](#)

Accessibility components

1. Contains a "keyboard-only" option allowing tabbing to the Address search field or County dropdown. The "Enter" key submits the search queries.
2. A popup appears upon selection and receives focus that contains the contact information for that County.
3. The visual focus order matches the intended interaction order.

More information



Minnesota Public Health Data Access Portal – Minnesota Department of Health

Introduction

A web portal featuring interactive web maps dedicated to visualizing and making public health data available to public health professionals and the public. The Minnesota Public Health Data Access Portal was created with all devices and audiences in mind. The data are accompanied by both a visual and tabular format so all users have access in a meaningful way. Accessibility testing has not been conducted for mobile devices at this time.

[Access the portal.](#)

Accessibility components

The portal includes some of the following functionality:

1. "Skip to page content" links are provided at the top of the page allowing keyboard users to skip past navigation elements and directly to content.
2. Maps and other data visualizations are accompanied by data tables that are accessible to all users.
3. Each table has screen-reader only instructions that tell users how to drill down into the data via keyboard operations.
4. Maps are accompanied by a full data table and are also created with keyboard and assistive technology users in mind.
5. All map components, elements, and buttons are tabbable using the 'Tab' and 'Enter' or 'Spacebar' keys.
6. Consider keyboard only users. When a user accesses the application using their mouse or keyboard all map components, elements, and buttons show focus (:focus) throughout the application.

More information

There are many different topics covered in the portal. One example is the [Asthma in Counties web map](#) and accompanying table:

Asthma in Counties

ED visits Hospitalizations Table Notes

Asthma ED visit rates for 2019-2021 by county

Select a county from the menu below to view details for that area

[Download data \(CSV\)](#)

County (select one or more)

(All)

ED visits

Age-adjusted rate per 10,000 people

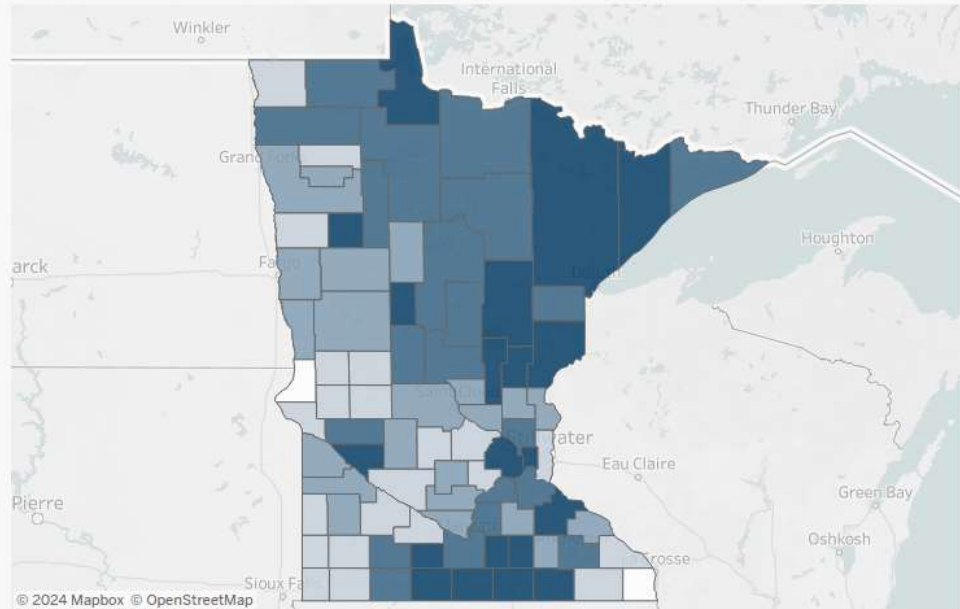
■ 29.6-47.6

■ 22.9-28.9

■ 19.1-22.8

■ 8.1-18.5

□ *Data suppressed



Minnesota Statewide

In **Minnesota** from 2019-2021, the statewide age-adjusted rate was **29.8** cases per 10,000.

Select a **County** from the menu above to view a specific county.

ED visits

Hospitalizations

Table

Notes

Asthma hospitalization rates for 2019-2021 by county

Click [Download data](#) to the right for a plain text version of the table

[Download data \(CSV\)](#)

County (select one or more)

(All) ▾

Outcome	County	Age-adjusted rate per 10,000	Count of cases	Note
ED visits	Aitkin	36.2	126	Null
	Anoka	22.8	2,303	Null
	Becker	22.6	204	Null
	Beltrami	25.3	336	Null
	Benton	19.1	226	Null
	Big Stone	9.5	12	Unstable rate due to small underlying ..
	Blue Earth	28.8	572	Null
	Brown	14.9	96	Null
	Carlton	23.3	232	Null
	Carver	11.3	337	Null
	Cass	25.5	186	Null
	Chippewa	40.3	129	Null



Interactive Web Maps (Complex & Wayfinding)

Environmental Review Projects – Minnesota Environmental Quality Board

Introduction

The Environmental Review Projects Interactive Map provides information about in-progress environmental review projects and past environmental review projects in your community.

[Access the interactive web mapping application.](#)

Accessibility Components

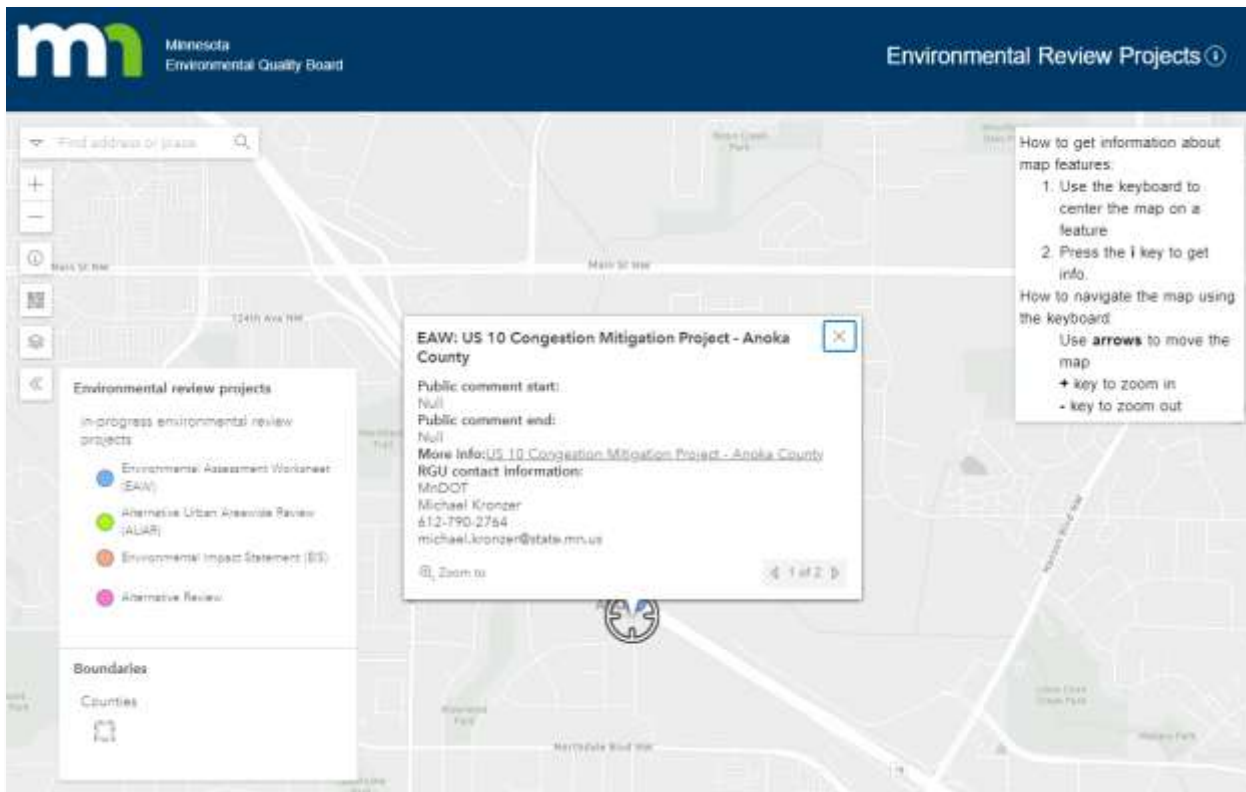
The application includes some of the following functionality:

1. All map components, elements, and buttons are tabbable using the 'Tab' and 'Enter' or 'Spacebar' keys.
2. A map tour can be launched on startup to describe the functionality of the map.
3. The opening splash screen is an intentional keyboard trap that loops tabbing back to the top to prevent users from tabbing to otherwise inoperable elements while the splash screen is open.
4. The application does not have any unintentional keyboard traps.
5. The map application includes the ability to click the map using only the keyboard. The functionality works as follows:
 - Tab to the "Information Tool".
 - The button has a hidden `` element that contains the text "Information Tool. Use the keyboard to center the map on a feature. Press the i key to get info." This is read by screen readers when the button receives focus.
 - Press enter when focused on the Information Tool button. The focus is moved to the map application and a crosshairs graphic is placed at the center of the map. Instructions open in the top-right corner of the map.
 - The map application uses the ArcGIS Maps SDK for JavaScript, which has some useful keyboard functionality out-of-the-box including:
 - Arrow keys to pan the map.
 - Plus and minus keys to zoom in and out.
 - Navigate to an area of interest. Press the 'i' key. This will open a popup showing attributes for nearby features. This functionality was added using custom code.

- When the popup opens, the focus is moved to the popup's close button. This indicates to the user that an action occurred and gives the user quick access to the popup's contents and the button to close the popup.
- When the popup is closed, the focus returns to the map application.

More information

Screenshot of the application with the Information Tool activated, the map application navigated to a feature, the popup for the feature open, and the close popup button focused:

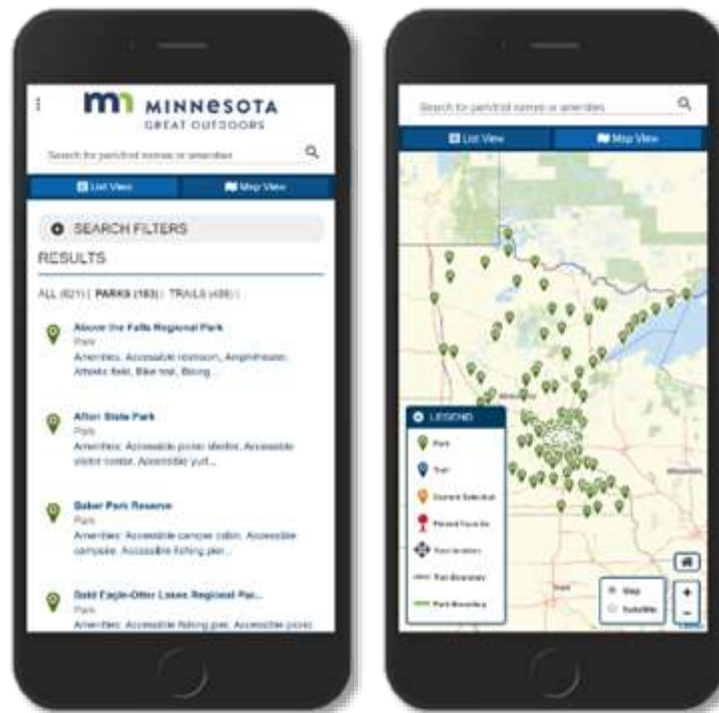


Introduction

An interactive web mapping application showcasing all state and regional parks and trails that are eligible to receive Clean Water Land and Legacy funding. Users can obtain maps, full details, and the most current information about any location.

The application was created to showcase all state and regional parks and trails that are eligible to receive Clean Water Land and Legacy funding.

[Access the interactive web mapping application.](#)



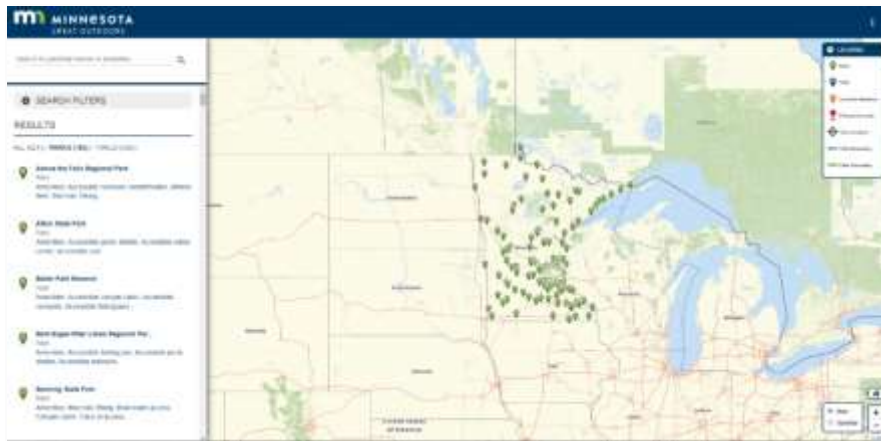
Accessibility components

The application includes some of the following functionality:

1. Simplicity is one of the key guiding principles of the application. Keeping the structure simple makes it easier to sustain and maintain accessibility.
2. A list complements the map and could be used as a standalone alternative, showcasing the data. Additionally:
 - There is a list-only option on mobile devices, and
 - The list comes first in the document structure when a user is navigating via the keyboard.

3. [Material design accessibility](#) was incorporated improving accessibility and usability for all users and includes some of the following components:
 - Alternative text, <alt>, tags that are smart;
 - Screen reader classes that can be added;
 - Easy to read buttons; and
 - Color contrast ratios throughout the interactive web mapping application.
4. Pagination allows users to get through the list more effectively and find what they are looking for.
5. The search was given a high order in the focus priority, making it more readily available when a user is navigating via the keyboard.
6. The logical tab order and focus placing the map <div> at the bottom of the tab order allows users to navigate through content first and takes away the ability to tab through pins on the map at the beginning of their experience.

More information



Mobile Maps – Minnesota Department of Natural Resources

Introduction

An interactive web mapping application containing geospatial PDF maps with digital coordinates. Users can download a geospatial PDF map for an area of interest and then load it to the [Avenza Maps](#) application (or similar) on their smartphone to view their location on the map. The Avenza Maps app will be able to show the device's location on the geospatial PDF map even when airplane mode is active or cellular service is unavailable.

The application was created to provide a list of geospatial PDFs to the public and Department of Natural Resources staff for use when in areas of low connectivity with all devices and audiences in mind.

[Access the interactive web mapping application.](#)

Accessibility components

The application includes some of the following functionality:

1. All map components, elements, and buttons are tabbable using the 'Tab' and 'Enter' or 'Spacebar' keys.
2. The opening splash screen is an intentional keyboard trap that loops tabbing back to the top to prevent users from tabbing to otherwise inoperable elements while the splash screen is open.
3. The application does not have any unintentional keyboard traps.
4. The list of geospatial PDF maps is the focus of the application. There are multiple ways to filter the list and find the map the user is looking for. Whenever the list is updated, a description is updated at the top of the list describing how many maps were found and what filters are used. This description has a div setting of role="status", which tells a screen reader to read the contents whenever the contents change.
5. The Map Topics dropdown element required a workaround due to a [known bug with the Angular Materials component](#) that was used. Problems include:
 - Items in the dropdown list are actionable via the checkbox, but they are not part of the tab order.
 - Items in the dropdown list can be reached via arrow keys, but the only announcements a screen reader will say is list position (e.g. "recreation basemap 2 of 10") and "selected" when pressing ENTER on an item that is not selected.

As a result, when a user arrows through the list, they do not know what is or is not selected. When the user presses ENTER on an item that is selected, there is no notice that anything happened. To work around these issues, the following approaches were used:

- Visually hidden text was added so when the dropdown is focused, screen readers will announce: "List items are actionable. Use arrow keys to access list items. Selecting the item with enter or

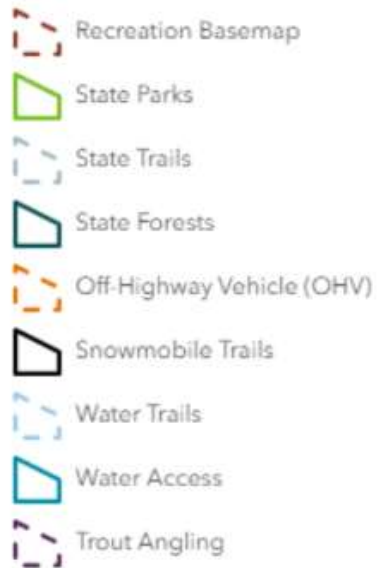
spacebar will toggle if the map topic is used to filter the map list. Multiple map topics can be selected at once.”

- The status description for the map list that has the count of maps and the filters used will be read by a screen reader whenever the description changes, and changing the selected Map Topics triggers a change to the description, so the status description serves as a notification of a change to list item checkboxes.
6. The list of maps can be long (478 total as of August 2024). For keyboard-only users that want to filter maps via a map location set by a map click, tabbing through all of the buttons would be tedious. To alleviate this, there are buttons that are hidden until they are focused at the top and bottom of the list that allow the user to jump from one to the other (“Skip to map list end” and “Skip to map list start”).
 7. The map application includes the ability to click the map using only the keyboard. The functionality works as follows:
 - Tab to the next element following the map application, titled “Activate Keyboard Mode” (labeled “KB”). Be sure to use the “skip to map list end” button on the way!
 - Press ‘Enter’ on the Activate Keyboard Mode button. A dialog opens describing how to operate the map in Keyboard Mode. Enable Keyboard Mode by selecting OK on the dialog.
 - The map application will be focused and a crosshair will appear over the center of the map.
 - The map application uses the ArcGIS Maps SDK for JavaScript, which has some useful keyboard functionality out-of-the-box including:
 - Arrow keys to pan the map.
 - Plus and minus keys to zoom in and out.
 - Navigate to an area of interest. Press the ‘Enter’ or ‘Spacebar’ on your keyboard. This will set a location on the map, filter the list of geospatial PDF maps to only those that cover that location, and show the footprints of those maps in the map application. This functionality was added using custom code.
 - The aria-live description for the map list that has the count of maps and the filters used will be read by a screen reader if in use.
 - Use ‘Shift’ + ‘Tab’ to return to the list of maps where you can zoom to the map footprint or download the map.
 8. If a location is set on the map, the map will show the geospatial PDF map footprints that cover that location. Setting the location can be completed via the common approach of a mouse click on the map. A location can also be set in ways more accessible to keyboard only and screen reader users such as:
 - Using “Keyboard Mode” as described above.

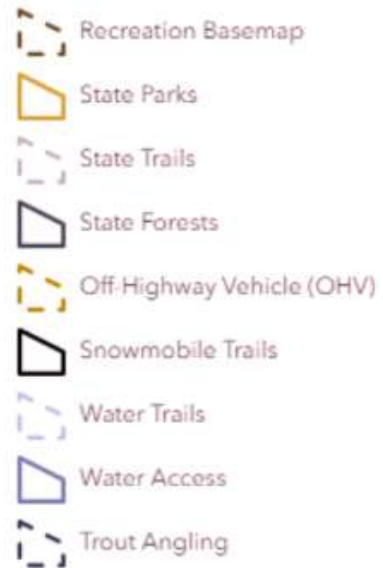
- Using an address or place search. This search option is in the Location tab. When a search result is chosen, the focus is changed to the Maps tab and the aria-live description for map count and filters will be read by a screen reader if in use.
 - Searching by Public Land Survey Township or Section. This search option is in the Location tab. When a search result is chosen, the focus is changed to the Maps tab and the aria-live description for map count and filters will be read by a screen reader if in use. Searching for Township has been simplified from separate Township, Range, and Range Direction inputs to one autocomplete input. When a Township is chosen, the autocomplete for Section is filtered to only Sections within that Township.
9. The application is designed with layouts that are responsive to changes in screen size. When viewed on a mobile device, the Maps tab is split into tabs for the Map List and the Map Areas. All text elements flow to prevent overruns. Legend items in the map use vector symbols and text, which responds effectively to different container sizes.
 10. The responsive design also serves to meet WCAG 2.1: 1.4.10 for reflow. If the browser is set to 1280 pixels wide and 1024 pixels high and the zoom level is set to 400%, the majority of the application is still functional. The exception on functionality is the widget buttons in the ArcGIS Maps SDK for JavaScript. While the buttons do resize in response to a mobile screen size, they do not resize on browser zoom. Browser zoom is notoriously hard to detect, especially with retina displays. The developers of the Mobile Maps application are still researching options to improve this.
 11. Geospatial PDF map topics are distinguished by outline color and style. Differing colors with styles pass distinction for: Deuteranopia, Protanopia, and Tritanopia color blindness.

Legend with different vision types

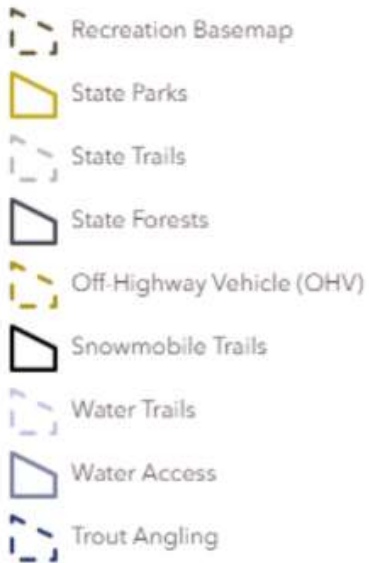
Normal Vision



Deuteranopia Vision



Protanopia Vision



Tritanopia Vision



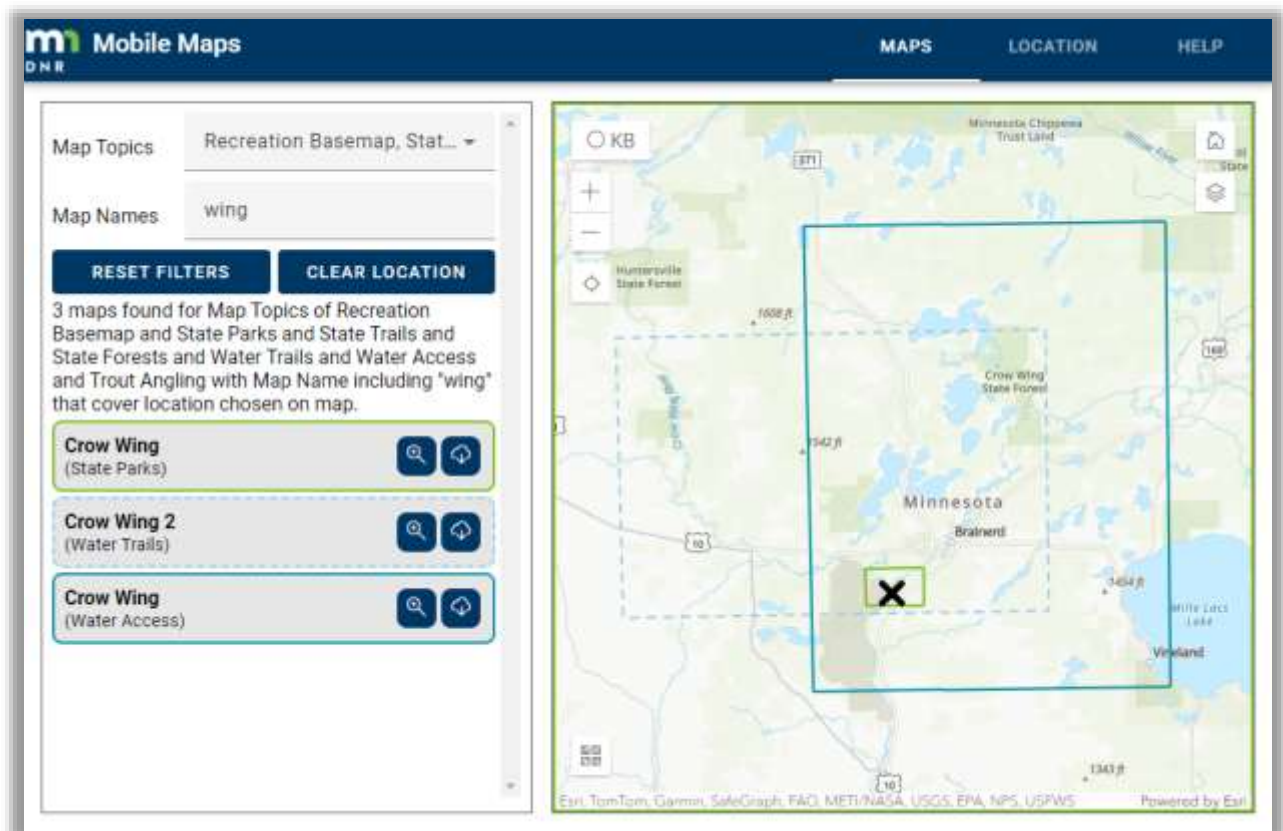
12. To visually align with the geospatial PDF map topic symbology, outlines for map list tiles are distinguished by a similar outline color and style.

3 maps found for Map Topics of Recreation Basemap and State Parks and State Trails and State Forests and Water Trails and Water Access and Trout Angling with Map Name including "wing" that cover location chosen on map.

- Crow Wing**
(State Parks)
- Crow Wing 2**
(Water Trails)
- Crow Wing**
(Water Access)

More information

Screenshot of the Mobile Maps application.



Watershed Health Assessment Framework: Lakes – Minnesota Department of Natural Resources

Introduction

The Watershed Health Assessment Framework: Lakes app helps natural resource professionals and the general public to learn about lake health, review individual lakes for different health metrics, and compare lakes within similar geographic areas.

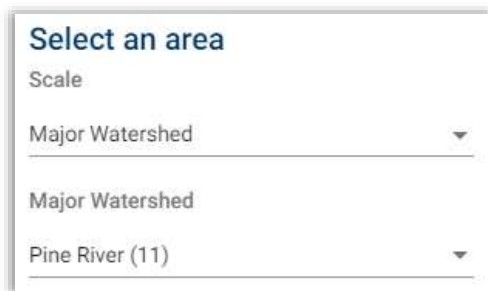
The application was developed using [Angular 16](#) and uses the [ArcGIS API for JavaScript version 4.23](#). Non GIS user interface components were built using a combination of [Ionic](#) and [Angular Material Design](#).

[Access the interactive web mapping application.](#)

Accessibility components

The application includes some of the following functionality:

1. All map components, elements, and buttons are tabbable using the 'Tab' and 'Enter' or 'Spacebar' keys.
2. Functionality to find areas of interest (watersheds and lakes) was put at the top of the semantic HTML order, making it more readily available when a user is navigating via the keyboard.



Select an area

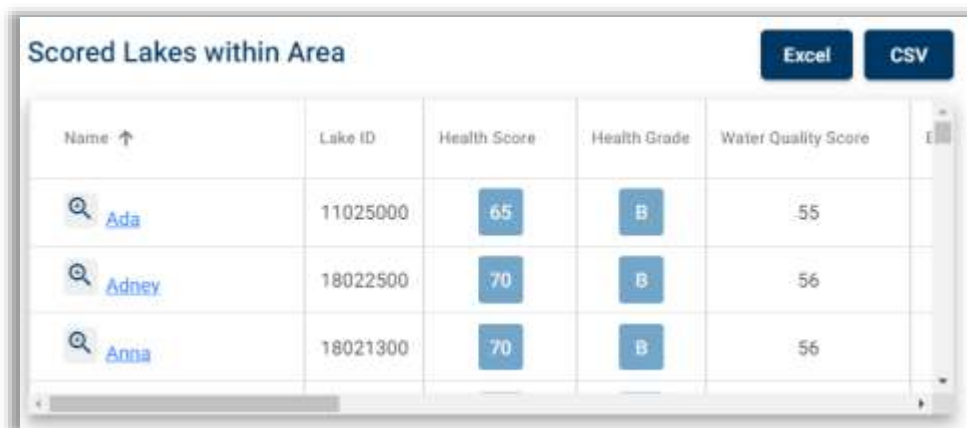
Scale



Major Watershed

Major Watershed

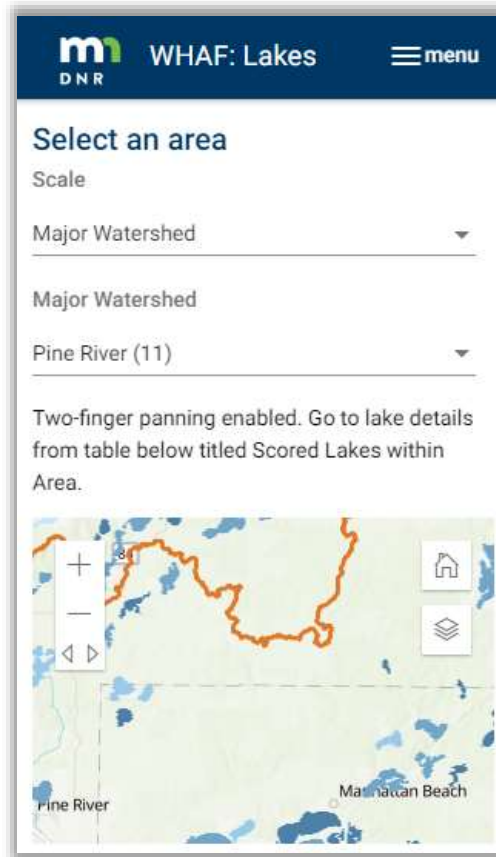
Pine River (11)

3. All data about lake health is made available in tables or narratives.



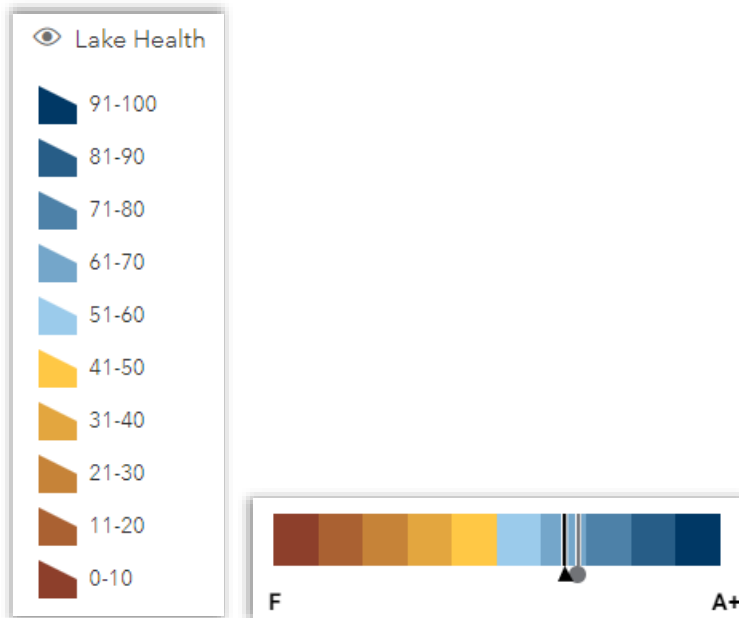
Name ↑	Lake ID	Health Score	Health Grade	Water Quality Score	
 Ada	11025000	65	B	55	
 Adney	18022500	70	B	56	
 Anna	18021300	70	B	56	

4. Responsive design is employed to configure the layout based on screen size. This helps with both mobile device use and with web browser zooming.
5. When on a mobile device, the map becomes part of vertically scrollable content. Two-finger panning is enabled to allow the user to use swipe gestures to scroll past the map.

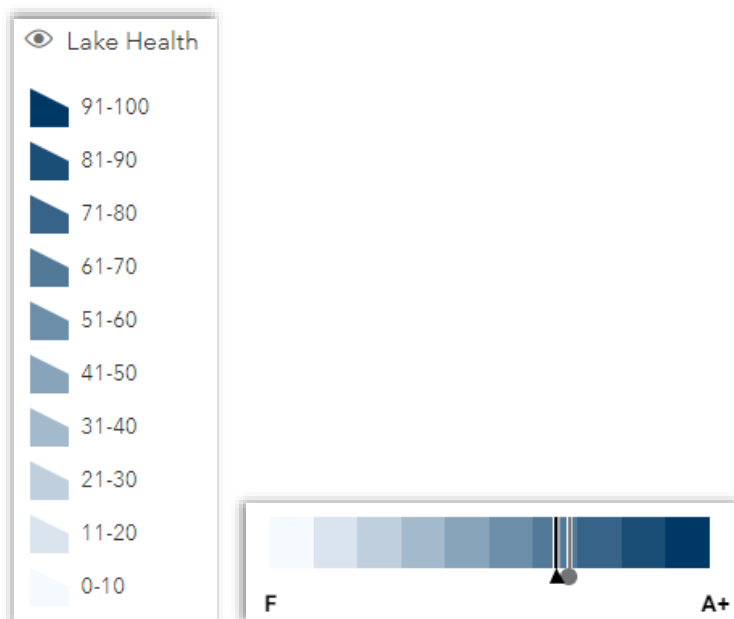


Were two-finger panning not enabled, the swipe gesture could result in a map pan event and the mobile device user may not be able to scroll past the map. This unfortunately then conflicts with [WCAG 2.1: 2.5.1 for Pointer Gestures](#) because the user is unable to pan the map with a single-point gesture. There are plans to add a toggle to turn off two-finger panning for mobile screen sizes.

6. To illustrate lake health, the data visualizations in the app use a divergent color ramp based on the [Minnesota Brand Color Palette](#). The divergent color ramp goes from Accent Orange to Extended Accent Gold for low health scores and from Extended Accent Sky Blue to Minnesota Blue for high health scores.



While this ramp was carefully considered (use of green for high health scores was avoided), it still may not work well for some users. To help there, an option is provided in the menu to “Change to Monochrome Ramp”. This updates all parts of the application that show the divergent ramp (map, tables, and data visualizations) to show a monochrome ramp from light blue to dark blue instead.



7. The application has two pages (lake lists by area and lake details), but essentially behaves as a single page app. In the “Scored Lakes within Area” table on the lake list page, the element that navigates the user to an individual lake detail page could have been a button that changes what is displayed or a hyperlink to navigate to a “new” page for the lake details. The hyperlink was ultimately chosen because the page refresh made resetting the keyboard focus far more straightforward.

Section 5: Technology Recommendations

General Web Recommendations

It is strongly recommended developers create the element you wish to use (e.g. a button as a button, rather than a div as a button), as functionality and accessibility are built in, and require no JavaScript or ARIA and is robust for the future.

```
<!--HTML for button that meets accessibility-->
<button id="searchBtn">Search</button>
```

However, if you must create a <div> element acting as a button, there are a few elements the developer must add in to ensure the element is keyboard accessible, including the: 'role,' 'tab index,' and functionality when a user clicks or accesses via the enter or space keys. The developer must ensure the <div> element can be accessed by a keyboard using the 'tabindex' attribute.

```
<!--HTML for a div mimicking a button that meets accessibility-->
<div role="button" tabindex="0" id="searchBtn">Search</div>

<script>
  /* JavaScript */
  document.getElementById("searchBtn")
    .addEventListener("keyup", function (event) {
      event.preventDefault();
      if ((event.keyCode === 13) || (event.keyCode === 32)) {
        //Add functionality when a user accesses via the Enter/Space keys
        //Can be set to have the same functionality as click events
      }
    });
</script>
```

Generally avoid:

1. CSS outline

- **Setting a CSS outline of 0 or none should not be implemented** across the interactive web mapping application, as assistive technologies will be unable to navigate through your site.

```
<!--CSS that should NOT be implemented-->
a {
  outline: 0; /* Do not implement */
  outline: none; /* Do not implement */
}
```

2. aria-hidden

- Only use if hiding content improves the experience for assistive technology users by removing redundant or extraneous content.
- For example: A menu item that has text and a font awesome icon could use aria-hidden on the font awesome icon as it is redundant information to those using assistive technologies.

```
<!--ARIA hidden property-->  
<i class="fas fa-camera-retro" aria-hidden></i>
```

3. tabindex greater than 0

- Setting DOM elements to have tabindex values greater than 0 will change the order of the DOM elements in your mapping application. This can confuse not only those with assistive technologies, but sighted mouse-users, and the developers coding the application. Instead of changing the value of tabindex to a positive integer, change the order of the elements in the application to reflect the order seen/heard in the application.

4. tabindex = -1

- Setting DOM elements to have a tabindex of -1 hides the content from assistive technology users. While not recommended, it can be implemented for visual elements in the page. However, this is not common, and most applications will not have a need for implementation of tabindex = -1.
- For example: If displaying a div that contains a Google street view image, this content may not be pertinent to an assistive technology user in which case implementing a tabindex of -1 would be a good implementation.

ArcGIS Maps SDK for JavaScript

Key Accessibility Features

The [ArcGIS Maps SDK for JavaScript](#) includes the following accessibility features.

[Map](#) (aka MapView) navigation can be performed with keyboard shortcuts. **However, these keyboard commands conflict with some JAWS and NVDA keyboard commands.**

Action	Behavior
Arrow keys	Nudge the map to the left, right, up, or down
N	Adjust the view to point north
A	Rotate the view counterclockwise
D	Rotate the view clockwise
+	Incrementally zoom in
-	Incrementally zoom out

[SceneView](#) (for 3D maps) navigation can be performed with keyboard shortcuts (not tested with assistive technology).

Action	Behavior
Arrow keys	Nudge the scene to the left, right, up, or down
P	Move the camera to look perpendicular
N	Adjust the scene to point north
J	Move down closer to the view
U	Move up, higher from the view

[Popup widgets](#) in the SDK have a “shouldFocus” option within the “open” method that should be used:

```
/* use the shouldFocus option to shift focus to the popup when opened */
view.popup.open({
  location: event.mapPoint,
  shouldFocus: true
})
```

Also note:

- Tab order of widgets shown in the map application are reflected in the user experience.
- Page elements are keyboard accessible via the arrow, tab, spacebar and enter keys.

- The API allows the developer to include accessibility functionality in custom widgets/buttons more readily.

Resources

Esri has a [web page for Accessibility Resources](#) that links to multiple resources including articles, blogs, technical documentation, and videos.

There is a guide to the [Calcite Design System Accessibility for the ArcGIS Maps SDK for JavaScript](#). It discusses:

- Designing for individuals
- Color
- Animation
- Content
- Forms and labels
- Checklist of potential accessibility issues

The [ArcGIS Maps SDK for JavaScript Accessibility](#) guide discusses:

- Keyboard navigation
- Reduced motion
- Color
- Language

The [ArcGIS Web AppBuilder Accessibility](#) guide discusses:

- Accessibility support
- Screen readers
- Themes
- Widgets

[Release notes for the SDK](#) are also important to review.

Blog Posts and Presentations

At the 2024 Esri User Conference in San Diego, the Esri Accessibility team presented several sessions focused on accessibility in GIS. The [resources from these sessions](#) are available online.

Kitty Hurley and Jessica McCall's "[Build accessible web apps with ArcGIS Maps SDK for JavaScript and Calcite Design System](#)" focuses on the ArcGIS Maps SDK for JavaScript. The post discusses:

- Accessibility in general
- Color contrast
- Customizing graphics by basemap theme
- Navigating through content using focus
- Animations

Anne Fitz and Kitty Hurley's "[Supporting Reduced Motion: Enhancing Accessibility in Web Apps](#)" focuses on the ArcGIS Maps SDK for JavaScript. The post discusses:

- Importance of supporting reduced motion preference
- Reduced motion support in Esri online products

Kitty Hurley and Kelly Hutchins' "[Build Accessibility into your Web Maps with ArcGIS Maps SDK for JavaScript](#)" discusses:

- Live regions
- Map application descriptions

Jessica McCall's "[Designing and testing for accessibility in GIS and mapping](#)" covers accessibility in maps more generally, discussing:

- Designing for accessibility
- Testing for accessibility

Jessica McCall and Krista McPherson's "[Accessibility essentials for GIS and mapping](#)" covers ArcGIS Instant Apps. The post discusses:

- Basemap selection
- Colors
- Labels and areas of interest
- Alternative text and map descriptions
- Providing different ways to access data

Leaflet Library

Key Accessibility Features

The [Leaflet Library](#) is an open-source JavaScript library for mobile-friendly maps.

Map navigation:

Interaction	Keyboard Procedure
Pan map up/down, right/left	Arrow keys
Zoom in/out	+/- or tab to controls
Focus on marker	Tab to marker
Open popup	Select 'enter' on marker focus

Also note:

- [Map navigation](#) can also be modified using 'zoomDelta' options to customize the zoom level change (for both clicking and using the +/- navigation buttons).
- Includes the ability to add '<alt>' tags to Leaflet markers:
 - `L.marker(latlng, {alt:"text for alt attribute"})`
- Tile images use `role=presentation` and the library [sets alt text on map tiles](#) to an empty string so assistive technology doesn't read their URLs.
- [Map markers](#) use `tabindex=0` and popup content containers have an anchor tag to close. This allows the user to access the marker via keyboard or mouse.
- When using [image overlay](#), the alt text can be added in to display a single image over specific bounds of the map.
- Zoom controls use `role=button` and `aria-label="Zoom In/Out"` on the zoom buttons
- Note the long thread and resolution detailed in this [GitHub Issue](#) about accessibility improvements, first identified via iPad VoiceOver.

Resources

There is a [Leaflet tutorial on accessible maps](#) that discusses useful defaults, labeling markers, testing and plugins.

Mapbox GL Library

Key Accessibility Features

The [Mapbox GL library](#) is a client-side JavaScript library for building web maps with Mapbox's technology.

Map Navigation:

Interaction	Keyboard Procedure
Pan map up/down, right/left	Arrow keys
Zoom in/out	+/- or tab to controls
Tilt	Shift + up/down arrow keys
Rotate	Shift + right/left arrow keys
Focus on marker	*requires plugin" Tab into marker
Open popup	Select 'enter' on marker focus

Note:

- Mapbox includes no explicit references to map accessibility, except through a lightweight plugin (was in [development at one point](#))
- The canvas contains an aria-label="map"
- The zoom controls use buttons and aria-label="zoom in/out"
- The library includes the ability to tab into markers with plugin that uses tabindex, aria-label, title

Section 6: Testing

Testing is the only way to ensure the experience is accessible. It is key to perform a functional test using the WCAG 2.1 Success Criteria to verify how well web content functions compare to the criteria and is less subjective than usability testing.

Overview of WCAG 2.1, and its POUR principles:

Principle	Success Criteria	Level A	Level AA	Level AAA
1. Perceivable	1.1 Text alternatives, 1.2 Time-based media, 1.3 Adaptable, 1.4 Distinguishable	1.1.1, 1.2.1 – 1.2.3, 1.3.1 – 1.3.3, 1.4.1 – 1.4.5	1.2.4 – 1.2.5, 1.3.4 – 1.3.5, 1.4.3 – 1.4.5, 1.4.10 – 1.4.13	1.2.6 – 1.2.9, 1.3.6 1.4.6 – 1.4.9
2. Operable	2.1 Keyboard accessible, 2.2 Enough time, 2.3. Seizures, 2.4 Navigable, 2.5 Input modalities	2.1.1 – 2.1.2, 2.1.4, 2.2.1 – 2.2.2, 2.3.1 2.4.1 – 2.4.4, 2.5.1 – 2.5.4	2.4.5 – 2.4.7	2.1.3, 2.2.3 – 2.2.6, 2.3.2 – 2.3.3, 2.4.8 – 2.4.10, 2.5.5 – 2.5.6
3. Understandable	3.1 Readable, 3.2 Predictable, 3.3 Input assistance	3.1.1, 3.2.1 – 3.2.2, 3.3.1 – 3.3.2	3.1.2, 3.2.3 – 3.2.4, 3.3.3 – 3.3.4	3.1.3 – 3.1.6, 3.2.5, 3.3.5 – 3.3.6
4. Robust	4.1 Compatible	4.1.1. – 4.1.2	4.1.3	

Tools

1. Hardware

- Perform keyboard testing, and/or unplug your mouse while interacting with the interactive web mapping application.

2. Software

- [Colour Contrast Analyser](#): The Colour Contrast Analyser (CCA) helps you to determine the legibility of text and the contrast of visual elements, such as graphical controls and visual indicators. It features:
 - WCAG 2.1 compliance indicators
 - Several ways to set colors: raw text entry (accepts valid CSS color formats), RGB sliders, color picker (Windows and Mac OS only)
 - Support for alpha transparency on foreground colors
 - Color blindness simulator

3. Websites

- [WebAIM Color Contrast Checker](#): This website is a companion to the CCA software. Its most useful feature is a set of lighten and darken links that can be used to make subtle changes to existing colors until a “pass” rating is attained. This tool:
 - Allows the user to enter or select foreground and background colors
 - Provides functionality to lighten or darken colors to find acceptable contrast levels
 - Displays the contrast ratio based on the WCAG 2.0 luminosity formula
 - Indicates if the colors pass the WCAG 2.0 level AA and level AAA requirements for normal and large text sizes
 - Shows a sample of the colors specified for quick viewing

4. Browser extensions

- [aXe](#): An open-source rules library for accessibility testing. It was developed to empower developers to take automated accessibility testing into their own hands and to avoid common pitfalls of other automated accessibility tools. It does not require any outside server calls, and it can be customized to include custom rules and to integrate with all modern browsers and testing frameworks.
- [Web Accessibility Evaluation Tool \(WAVE\)](#): A tool to help web developers evaluate their web content and make it more accessible. Using the tool, enter a web address and the web page is presented with embedded icons and indicators giving information about its accessibility. Click on an icon to see a brief overview of what each icon means and view its documentation or access the documentation panel. People who are not web accessibility experts can also benefit from WAVE.
- [Text Color Contrast Analyzer Tool](#): This tool allows the user to identify text color contrast issues on a web page, including text on top of complex backgrounds or images. Parameters in the tool allow the text to be assessed using WCAG 2.0 color contrast requirements for either AA or AAA conformance.

5. Assistive technology: Assistive technology testing requires training either via a course or with an expert assistive technology user. Testing, as with any other tool, is different than using the tool.

- [JAWS](#): An assistive technology that allows blind and low vision users to read the screen either with a text-to-speech output or by a refreshable Braille display. JAWS has a \$95 per year subscription cost as of August 2024.
- [Non-Visual Desktop Access \(NVDA\)](#): A free and open-source assistive technology for the Microsoft Windows operating system. It provides feedback via synthetic speech and Braille enabling blind and low vision users to access computers running Windows.
- [VoiceOver](#): Assistive technology built into Apple Inc.’s Mac OS and related operating systems.

Exercises and simulations

“Testing all the things” is not realistic. It is also critical to test with users that use assistive technology and with the browser zoom/screen magnification on an everyday basis. However, there are certain exercises and functions **you** can test, for example:

1. **Perform keyboard testing**, or unplug your mouse!
 - Move keyboard focus using the ‘tab’ / ‘shift’ + ‘tab’ keys.
 - Do you see the visual indication of focus? If unable to locate the focus while testing in-browser:
 - Open browser developer tools (F12 in Chrome and Firefox)
 - In the developer tools, open the Console tab
 - In the web page, tab to an element
 - In the Console, type `document.activeElement`
 - The console should then print the element's HTML
 - If you hover over the HTML in the console, it should show an indicator on the page for the location of the element
 - Do you see the expected behavior while using the ‘Tab’ key in the application?
 - To access links, use the ‘enter’ key.
 - To access buttons, use the ‘enter’ + ‘space’ keys.
 - To access menus and form controls, use the arrow keys.
 - Is any content hidden by a ‘mouse-over’ event listener?
 - For additional support, see [WCAG 2.1 Principle 2, Operable](#), for a list of success criteria to perform keyboard testing.
2. **Turn off or disable CSS** on your mapping application to visualize the page without styling.
3. **Turn off or disable images** in your application to reveal alt text in blank image boxes.
4. **Verify the color patterns** you have implemented meet the [WCAG 2.1 Success Criterion 1.4, Distinguishable](#), for a list of success criteria to achieve color validation.
5. View your page in “high contrast” mode.
6. Verify field labels are coded to the proper form input when brought into focus.
7. Check the DOM’s heading order of <h1>, <h2>, etc. tags.
8. Verify items that require a change of context. For example, a dropdown menu.
9. Check the HTML’s navigation order.
10. Check the skip navigation and landmarks.

11. Check that lists are properly coded as a list. For example, a navigation menu with a dropdown should be a list.
12. Check link text.
13. Check multimedia components. For example, videos, featured stories, etc.
14. **Run an automated test.** No automated test tool can prove conformance with WCAG Success Criterion. Automated tests are a good starting point but cannot detect all accessibility issues. The test should run on each page state and are rendered in the browser's DOM. Use the browser tools noted above.
15. Share with a colleague. Once completed, contact your organization's accessibility point of contact to have them do a final check and provide their feedback.

Section 7: Conclusion

If you have read this entire document, then, congratulations! You should have a good idea on how to make web maps accessible for people with disabilities. A few important points are worth repeating:

- Accessibility testing tools can be an effective way of verifying accessibility for users living with disabilities; however, they are not a catchall for all situations and users. The best accessibility software covers approximately 25% of WCAG 2.1, catching the “big” issues.
- Manual testing, such as with the keyboard and using assistive technology such as screen readers, to run more intensive tests, are critical tools in discovering and isolating issues that automated tests miss.
- A final step, end-user testing in which individuals with disabilities run through your product with their assistive technology tools can help unearth both accessibility and usability issues.

Additionally, other factors can affect testing, such as: technology can be difficult to learn for a user, a user’s investment may differ if they have a disability, and/or a user’s disability type can play a part in their experience.

Lastly, it is critical to focus on all types of disabilities including, but not limited to, visual, cognitive, hearing and mobility disabilities.