ACCESS university

Top 10 Takeaways

ARIA for Developers I

1 Accessible Names

When a control does not have a name, many users will not be able to discover it. To provide an accessible name, give the content describing the element a unique id (i.e. <p id="“drumKitLabel”>DRUM KIT SELECT</p>). By adding a unique id attribute to an element we can programmatically associate it with the select element. Then, on the select element, add the aria-labelledby attribute with a value of “drumKitLabel”.

2 Labels and Descriptions

Without a label set in your code, there will be no information available to an assistive technology. Accessible descriptions provide tooltip-like information for your control. Associate the description of a field or control by putting the paragraph element's id value into the aria-describedby attribute on the input field.

3 Focus Navigation

By ensuring a control is keyboard-accessible, you will avoid one of the common web accessibility mistakes. Using a DIV element loses built in support for the keyboard to be able to toggle the button on and off. Add an onkeydown handler and support detection of the ENTER and SPACE keys to activate the button.

4 ARIA Listbox

When you build a custom composite control that has focusable parts, an attribute called aria- activedescendant can help with managing focus between items. Listboxes, trees, and grids are all examples of elements that can benefit from using the aria-activedescendant this way. Each item inside the container has a unique id attribute. Set tabindex to negative one. The composite element that is a div will have tabindex set to 0. The aria-activedescendant attribute will ensure that as focus changes visually in the UI, it stays in sync with focus that changes programmatically.

5 Visual Focus

UI changes which occur on hover cause issues for people who rely on the keyboard to navigate your content. Ensure that the element clearly indicates when it has keyboard focus. This can often be adjusted by ensuring that CSS styling applied to hover is also applied to both focus and active CSS pseudo-selectors. Avoid using outline:none on your elements. This can remove the highlight rectangle that the browser provides to display visual focus.

6 Accessible Menu

The TAB key should bring the user to focus on the first item in the menu. The next tab should bring the user off of the menu bar and in to the next interactive control. This behavior will ensure that users do not have to tab through endless lists of items to get to the next interactive part of the page. Similarly SHIFT+TAB must allow the user to navigate back up to the menu bar. When focus is on the menu bar, the left and right arrow keys must be able to be used to move between the next and previous menu bar items. Down, Space, and Enter should all be supported to expand a menu item that is within a menu bar. ESCAPE must be supported to close the menu and move the user back to the parent menu item.

7 ARIA Menubar

The container for an initial set of menu items will typically be set with role equal to menu bar. The container for an expanded set of items will be set with role equal menu. The menu control itself always contains various types of menu items. Most items in a menu will use role equal to menuitem. If a menu item can be expanded or collapsed it must support the attribute aria-expanded. This should be set true or false in accordance with the visual state of the UI. Menu items that can launch another menu must have aria-haspopup equal to true.

8 Accessible Dialog

Dialogs are a common cause of programmatic focus errors. Developers can use the tabindex attribute with a value of negative one on the div tag that contains the dialog's content to set focus (using the JavaScript focus method) on the content as a whole. Information in the dialog should be associated so that assistive technologies announce the purpose of the dialog to the user. The text contained within the dialog can be associated through using aria-labelledby to represent the dialog title. In cases where the user is presented with an alert, set the role to alertdialog. This role will enable functionality similar to the aria-live attribute being set to assertive.

9 Hiding Content and Verifying with NVDA

Ensure that when a dialog is launched, the user's screen reader is not able to read content that is behind the popup dialog in the DOM. Aria-hidden set to true must be separately applied to all content that needs to be hidden. The code inside of your dialog should have aria-hidden set to false.

10 Accessible Tabs

Page tab roles should not be applied to navigation links that are used to switch pages or change out all of the main content of a page – links should continue to be used for these navigation items. Tab roles should not be used for accordion elements (tabs that are presented vertically). When keyboard focus is on a tab you should support using the left and right arrow to move focus between the tabs. The tab key should only be used to move focus to the selected tab, it should not be used to move between tabs. The HOME and END keys should be used to move focus to the first and last tab respectively.