Basic Accessibility Guidelines

Introduction

This document will describe the ways in which Accessible Rich Internet Applications (ARIA) labels and mobile accessibility media triggers should be used to ensure that your webpage has proper accessibility for those with disabilities and mobile users. This does not in any way describe all the accessibility features that a web page should have, but simply gives an overview of ARIA labels and media triggers.

Accessible Rich Internet Applications (ARIA) labels

ARIA labels are a simple yet effective way to make a webpage more navigable for people with visual impairments. The implementation of ARIA label is relatively straight forward, we simply have to use the “aria-label=” inside the html like so:

<button aria-label=”homepage” onclick=”homepage()”> . . . </button>

And that is it. However, there are other similar labels you can use to help out even further, such as “aria-labelledby=” and the “aria-describedby=”. These both have slightly different semantic functionality.

- The labelledby label is used to link two elements together inorder to provide additional context about the current html tag.
- The describedby label, similar to the labellby label, descripes the current element even further.

Conventions and common practices:

- Aria-labels should not be a replacement for semantic html (link provided for context). Aria labels are meant to be used in conjunction with semantic html.
- Testing Aria labels are with a screen reader is highly recommended, otherwise you will not know how you page will actually work in regards to ARIA-labels.
- Though Aria labels can be used for any HTML element, ARIA labels are mainly used for buttons and labels

Mobile Accessibility Media Triggers

If left unaddressed, webpages will likely display incorrectly on mobile. To address this, we can use media triggers. The most easy to implement media trigger I was able to find was the following CSS class: “@media only screen and (max-width: 900px)”.

Example Usage (CSS):

```
@media only screen and (max-width: 900px) {
        .flex-container {
            flex-direction: column;
            padding: 4px;
        }
}
.flex-container {
        display: flex;
        flex-direction: row;
        padding: 10px;
}
```

In this code snippet, we can see that we have a “default” class, “flex-container” which is simple flexbox use for webpage layout. When the webpage detects that the user is viewing the page in portrait mode. The page will automatically update to instead use the class found inside the brackets after @media only screen and (max-width: 900px). In this case, direction of the flex box (for row to column) and the padding around the flexbox (from 10px to 4px).

Example usage #2 (CSS), Hiding Content:

```
@media only screen and (max-width: 900px) {
  .mobile-hide{
    display: none;
  }
```

In this case, we just want to completely hide the content when we are viewing this from a mobile device. Since the class doesn’t have a default case defined, it will not affect the content on the page (the content with the .mobile-hide class). Therefore, this class will only affect the page when we are viewing the page from a mobile device (in portrait mod). In the HTML you can just set up you class like this:

```
<th class="one">Etc. Data</th>
```

Contract Ratio Best Practices