ADA Web Accessibility In React

By John Dahl

Experience

Experience

Timeline

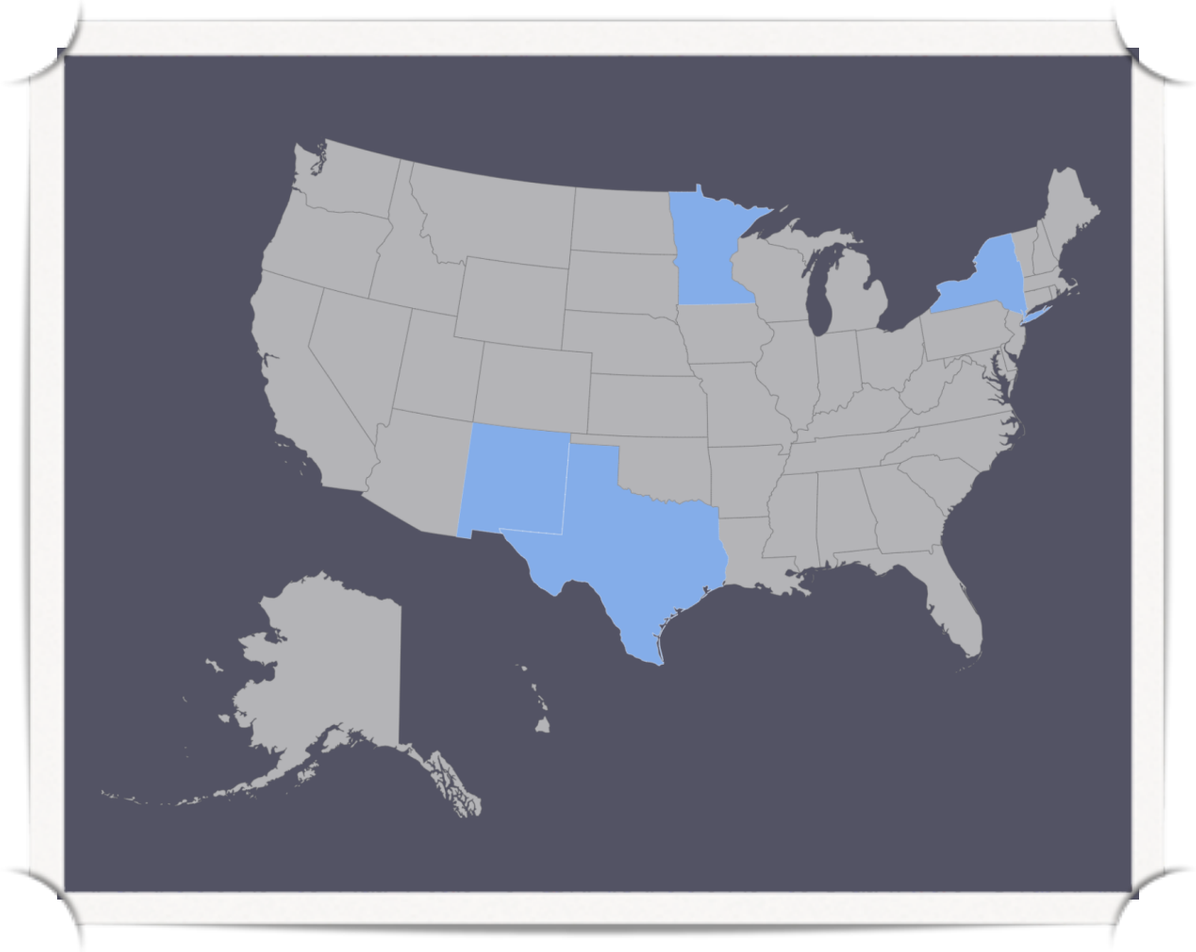
University of Minnesota - 2003-2007

Minnesota - 2007-2009

New York - 2009-2013

New Mexico - 2013-2016

Texas - 2016-Present



New York

Notable Clients/Roles

MTV Networks - CSS Architect

Untuckit - Ruby Developer

Shellac NYC - PHP Developer

New Mexico

Inixta Technologies

Construction Software Platform

- iPad App (Objective C)
- Windows App (Visual Basic)
- Online App (PHP, MySQL)

Notable Clients in Texas

- Texas A&M
- TCU
- Lewisville School District
- Keller ISD

Texas

Neiman Marcus

UX/UI Designer

2016-2017

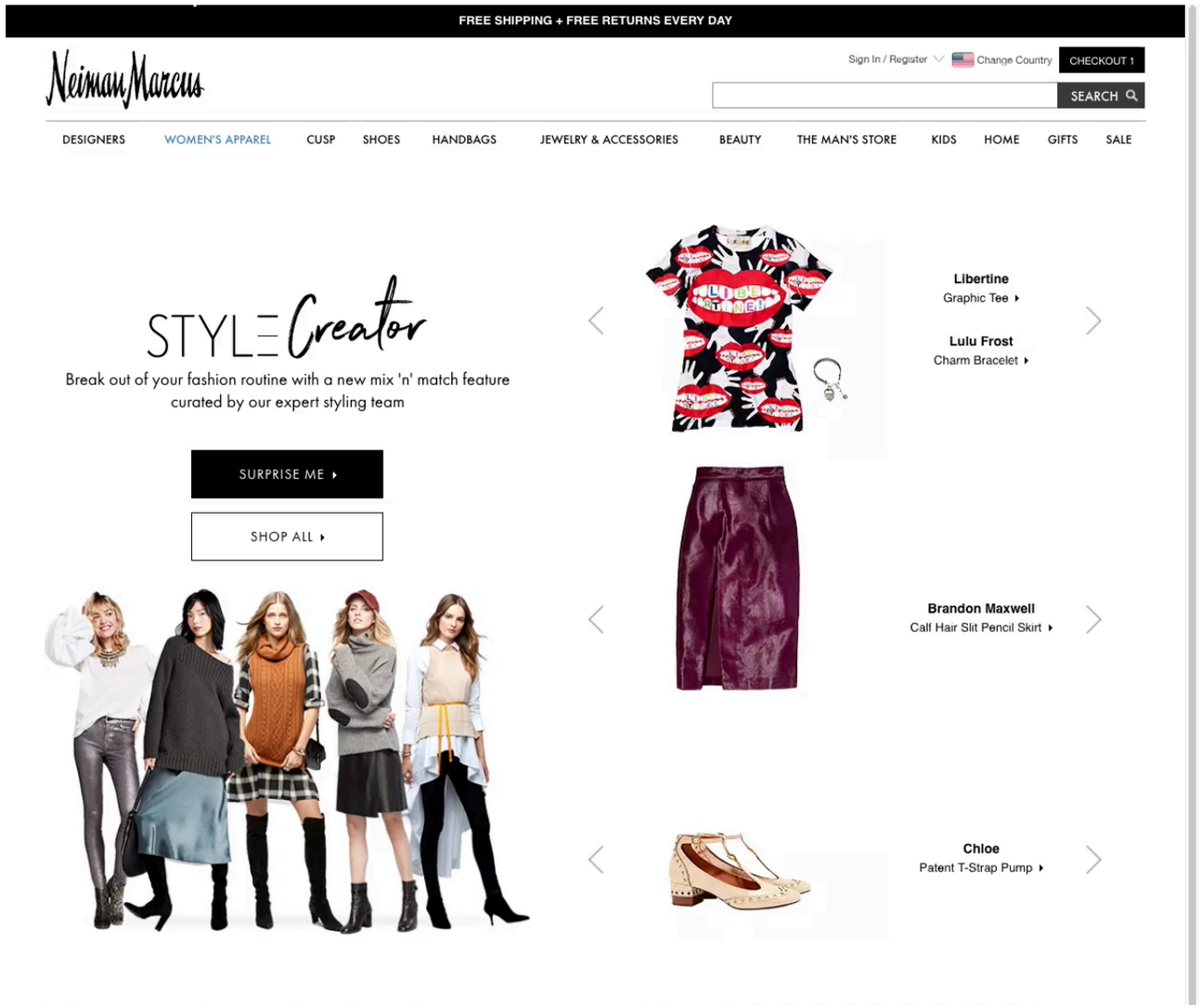
- Build prototypes for Creative (Javascript)
- Design User Experience Workflows
- Format NM Footer and InCircle Website (CSS)

Software Development Engineer -

Cloud Services, UI, ADA

2017-Present

- React.js
- Node.js
- Jenkins
- Mocha, Chai, Enzyme



Web Accessibility

Web Accessibility

Web accessibility means making the web available to everyone.

There are over one billion people with disabilities, which is about 15-20% of the population.

Making the web accessible is similar to:

- Having a wheelchair ramp to your business.
- The crosswalk sound notifications announcing the time to cross the street.
- Color contrast on highway signs so you can read them from a distance.

The same concepts apply to the web.

Sources:
https://www.w3.org/WAI/videos/standards-and-benefits.html
https://images.myparkingsign.com/img/dp/lg/access-ramp-directional-sign.png

Types of Disabilities

There are multiple types of users with disabilities that we have to think about when making our websites and apps accessible.

Blindness

Use text to speech (screen readers) to use websites

Low Vision/Visually Impaired/Low Contrast

Zoom in on your website to view

Hard time differentiating colors

Cognitive Disabilities

Need text and content to be simple to understand

Motor

No arm function users use a mouth stick to type

Hard of Hearing

Need captions for audio and video

Reduced Dexterity

Need buttons and links bigger

Other Factors

There are other types of factors that we also have to think about when making our websites and apps accessible.

Short Term Disabilities

Eye Surgery

Broken arm

Carpal tunnel syndrome

Tennis elbow

Trigger finger

Outside Circumstances

Mouse/Keypad Stops Working

Captions for video or audio due to a quiet environment

Holding a baby and only have one arm available

Many use just keyboard for preference or circumstance

Our Responsibility

As developers, we are responsible to make the web available to everyone.

Main Concepts

1. Format your code for screen readers
2. Make sure your site is scalable
3. Check your color contrast
4. Have clear and concise content
5. Make your site entirely keyboard accessible
6. Add captions for audio and video
7. Make buttons and links larger

Benefits

Besides being a good person and making your site accessible to every, there are additional benefits.

Main Benefits

- Better overall usability and user experience
- SEO rank increase
- Avoid American Disabilities Act (ADA) lawsuits

Case Studies

SEO Rank Increase

CNET - 30% increase in traffic from Google after CNET started providing transcripts.

ADA Lawsuits

Target Corporation - settlement for damages of $6 million USD and attorney's fees and costs over $3.7 million after lawsuit by US National Federation of the Blind (NFB).

Source: https://www.w3.org/WAI/bcase/resources.html

How do we do this?

Optimize for Screen Readers

Screen Reader Options

JAWS: $1,300 screen reading software

NVDA: Free screen reading software

VoiceOver: Built into OSX

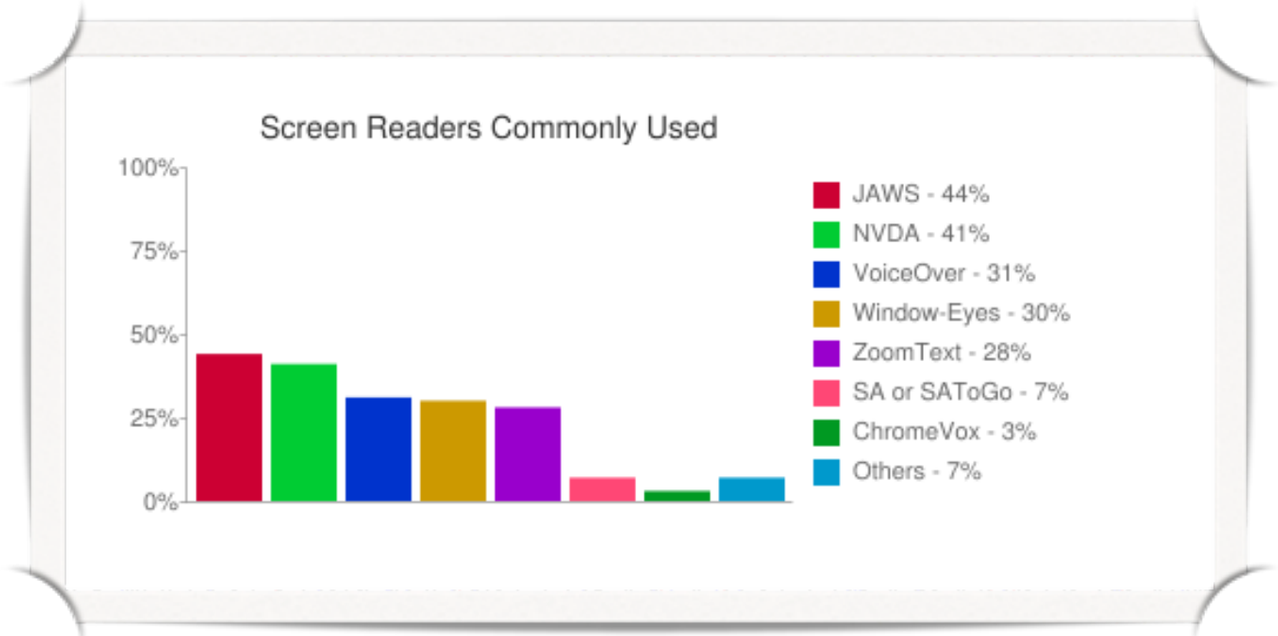
TalkBack: Built into Android Devices

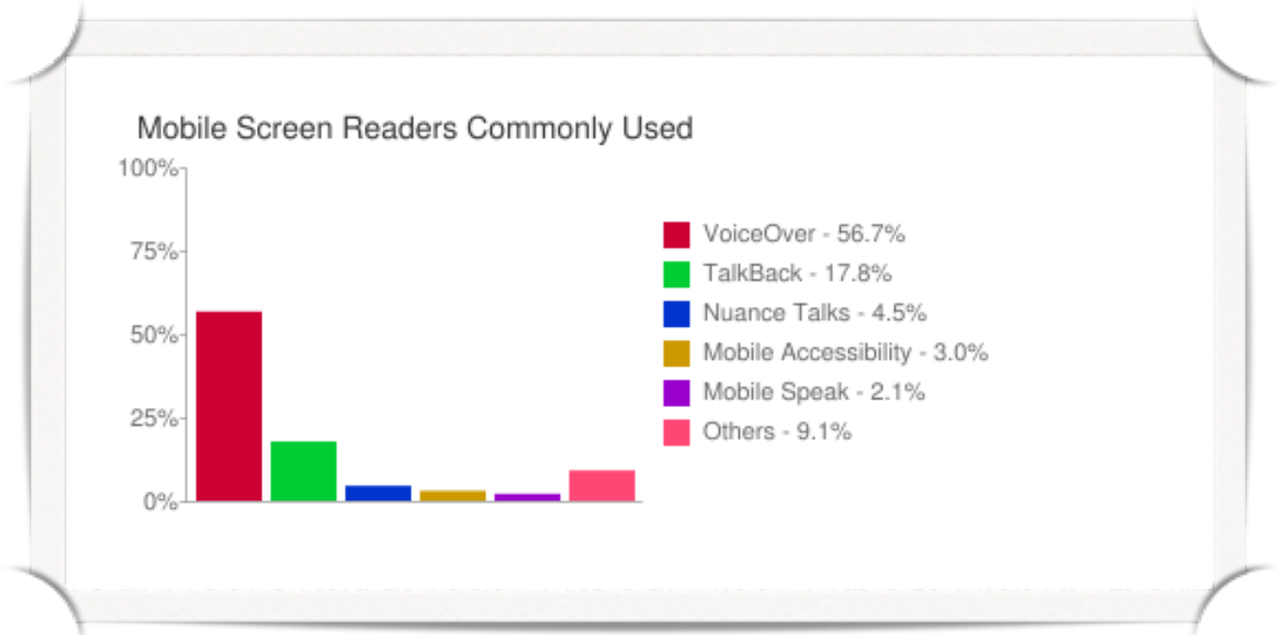
Most popular desktop combinations:

- PC + JAWS + Windows + Internet Explorer
- PC + NVDA + Windows + Firefox
- MAC + VoiceOver + OSX + Safari

Most popular mobile combinations:

- VoiceOver + iOS
- TalkBack + Android





Source: https://webaim.org/projects/screenreadersurvey6/

Keyboard/ Screen Reader Actions

The keyboard and screen reader users use buttons on the keyboard to navigate and fire action events on a webpage.

Two Types Elements:

- Interactive Elements
  - Links
  - Buttons
  - Inputs
- Non-Interactive Elements
  - Headings
  - Static Text
  - Images

Shared Actions

Down the DOM tree: **TAB**

Up the DOM tree: **SHIFT + TAB**

Simulating a click: **ENTER or SPACE BAR**

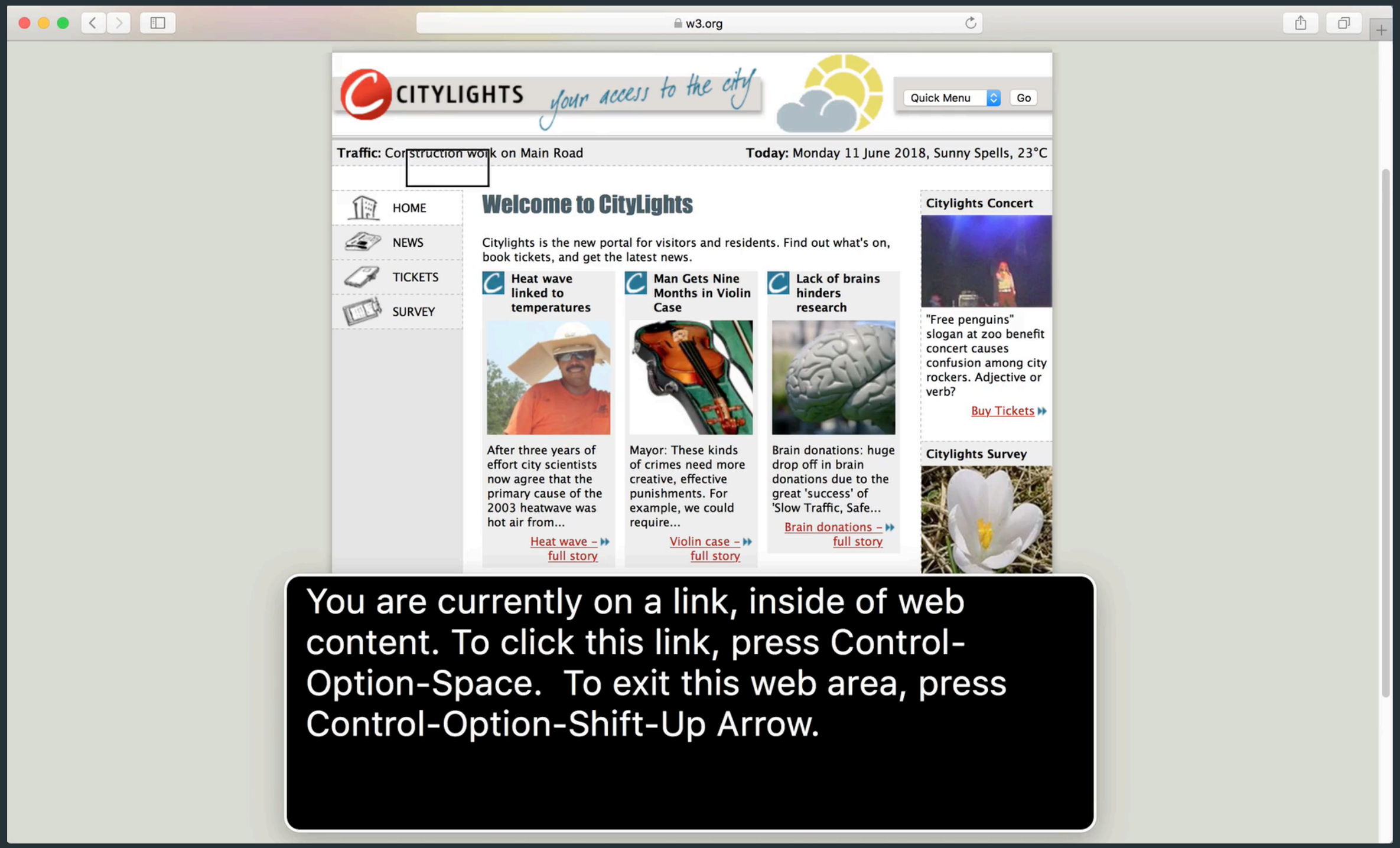
Close a popup/modal: **ESC**

Screen Reader Only Actions

Quick Navigation: **LEFT ARROW + RIGHT ARROW**

Source: https://pixabay.com/en/keyboard-computer-technology-laptop-1176257/

VoiceOver Demo



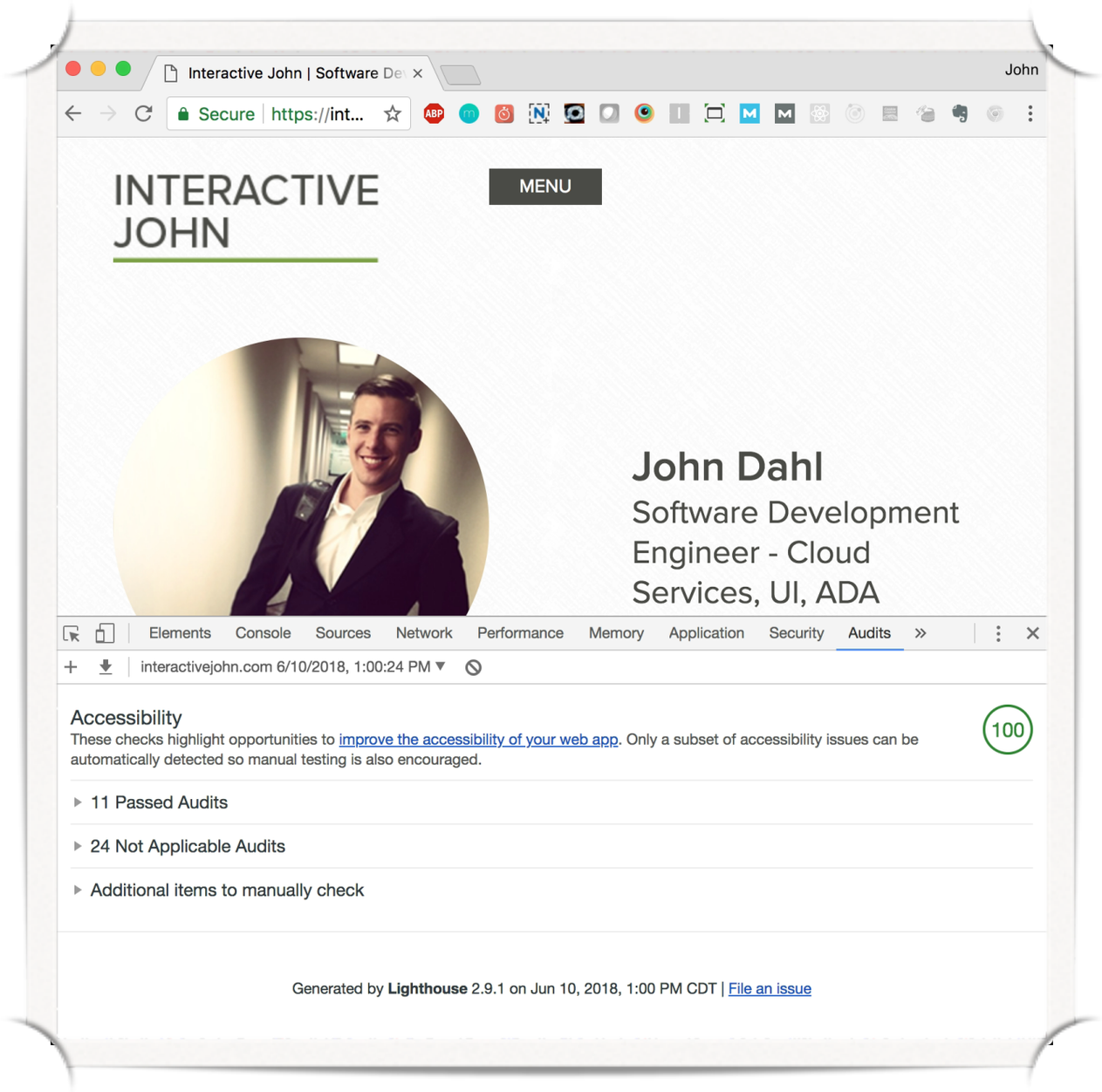
How do we test for this?

Accessibility Audit Tools

There are 109 audit tools listed on the W3 website, but none of them test the actual screen reader. https://www.w3.org/WAI/ER/tools/

These tools just test semantics in your code to make sure that all the code is compliant. It does not test functionality of the accessibility.

Google Lighthouse's Accessibility Audit is the one I generally use and recommend.



More Info: https://developers.google.com/web/tools/lighthouse/

ARIA


Aria
RESORT & CASINO
LAS VEGAS

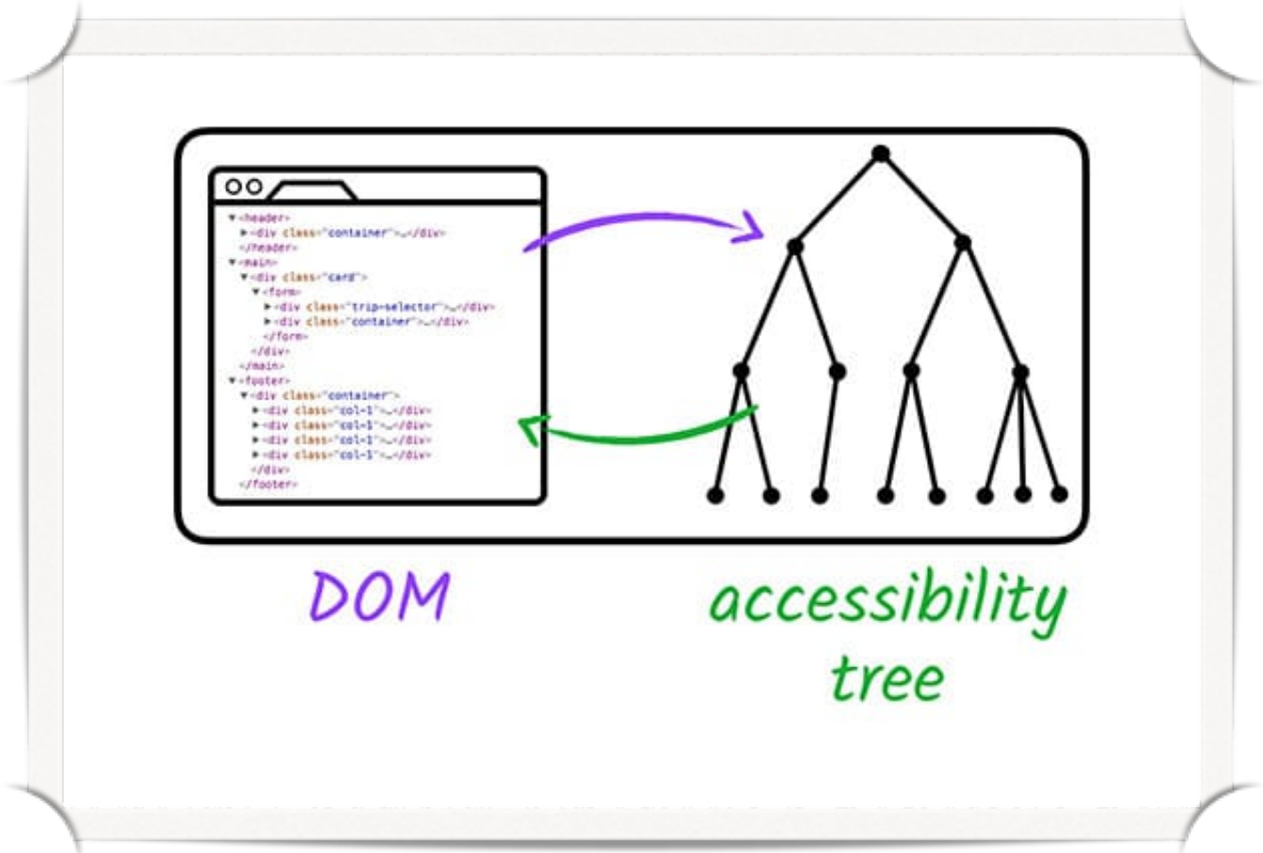
WAI-ARIA

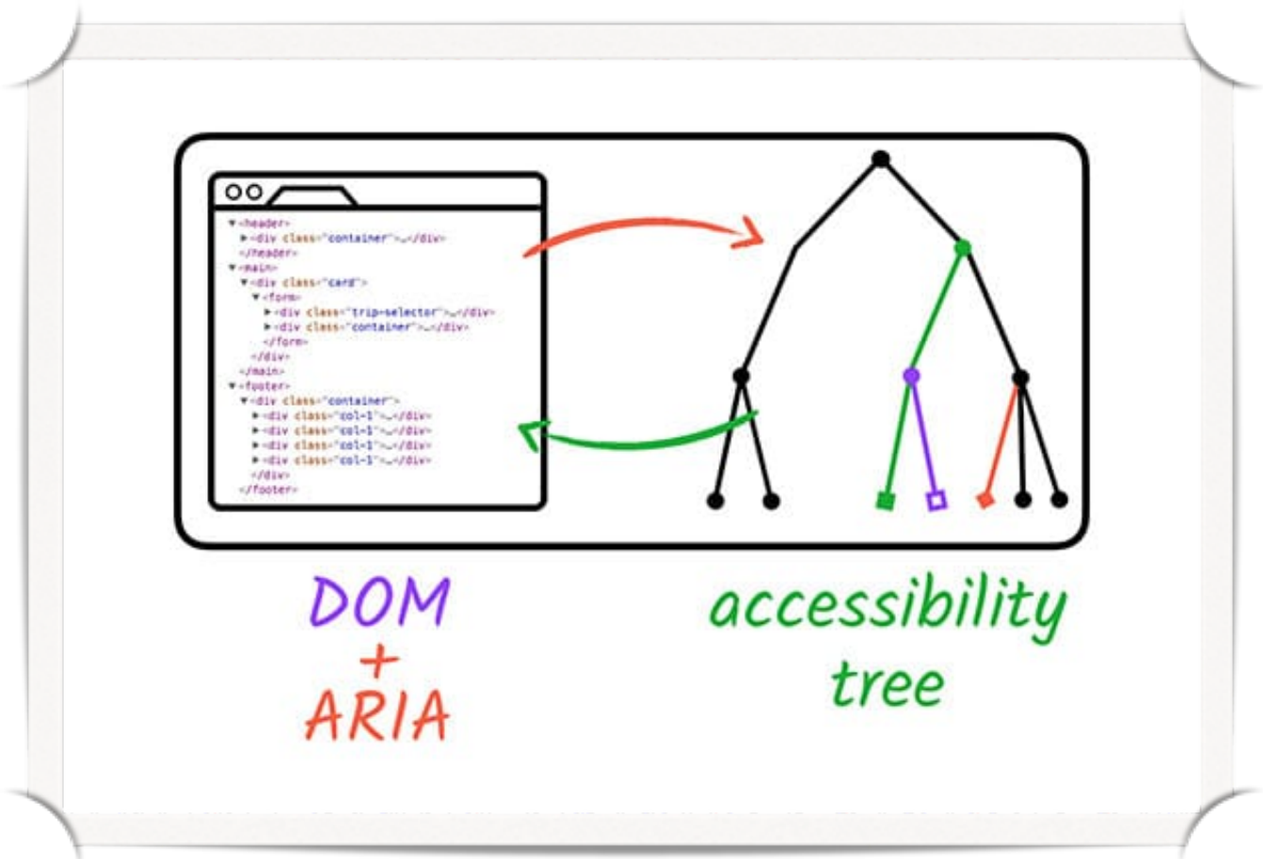
Web Accessibility Initiative - Accessibly Rich Internet Applications

ARIA

ARIA works by changing and augmenting the standard DOM accessibility tree.

ARIA doesn't augment any of the element's inherent behavior; it won't make the element focusable or give it keyboard event listeners.





https://developers.google.com/web/fundamentals/accessibility/semantics-aria/

ARIA Attributes

Most Common ARIA attributes

- aria-label
- aria-expanded
- aria-live
- aria-checked
- aria-hidden

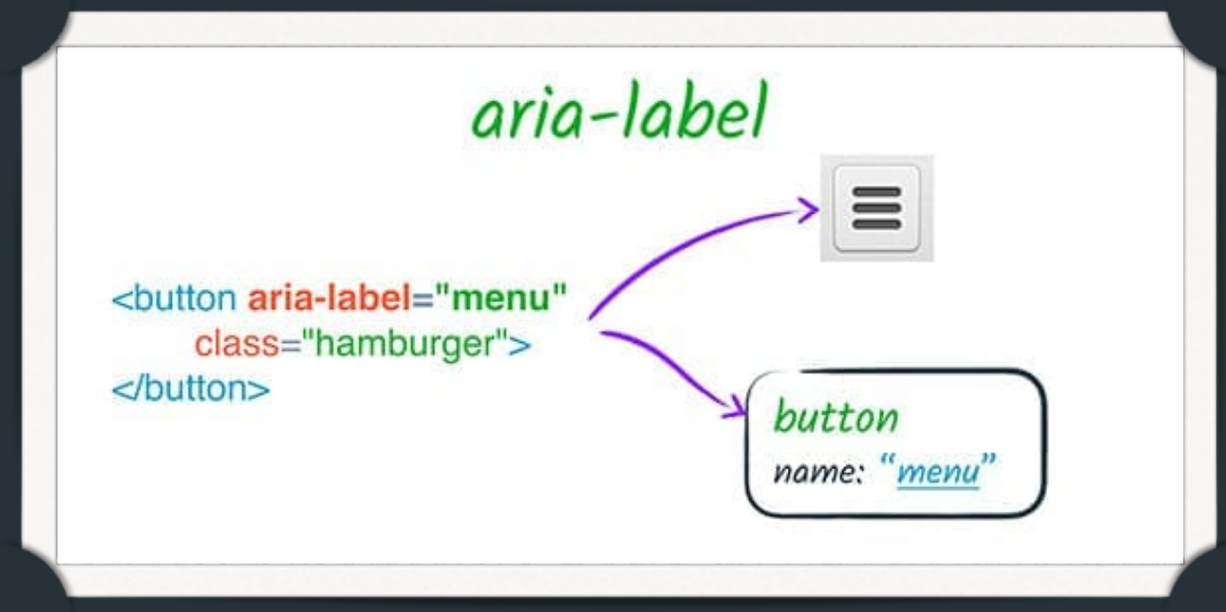
Other Common HTML Accessibility Attributes

- role="button"
- role="link"
- role="dialog"
- role="alert"
- tabindex

How do we implement these in React?

ARIA-Label

Used for applying a description for elements that need to be labeled, such as a button using a character for representation or a button that has an image.



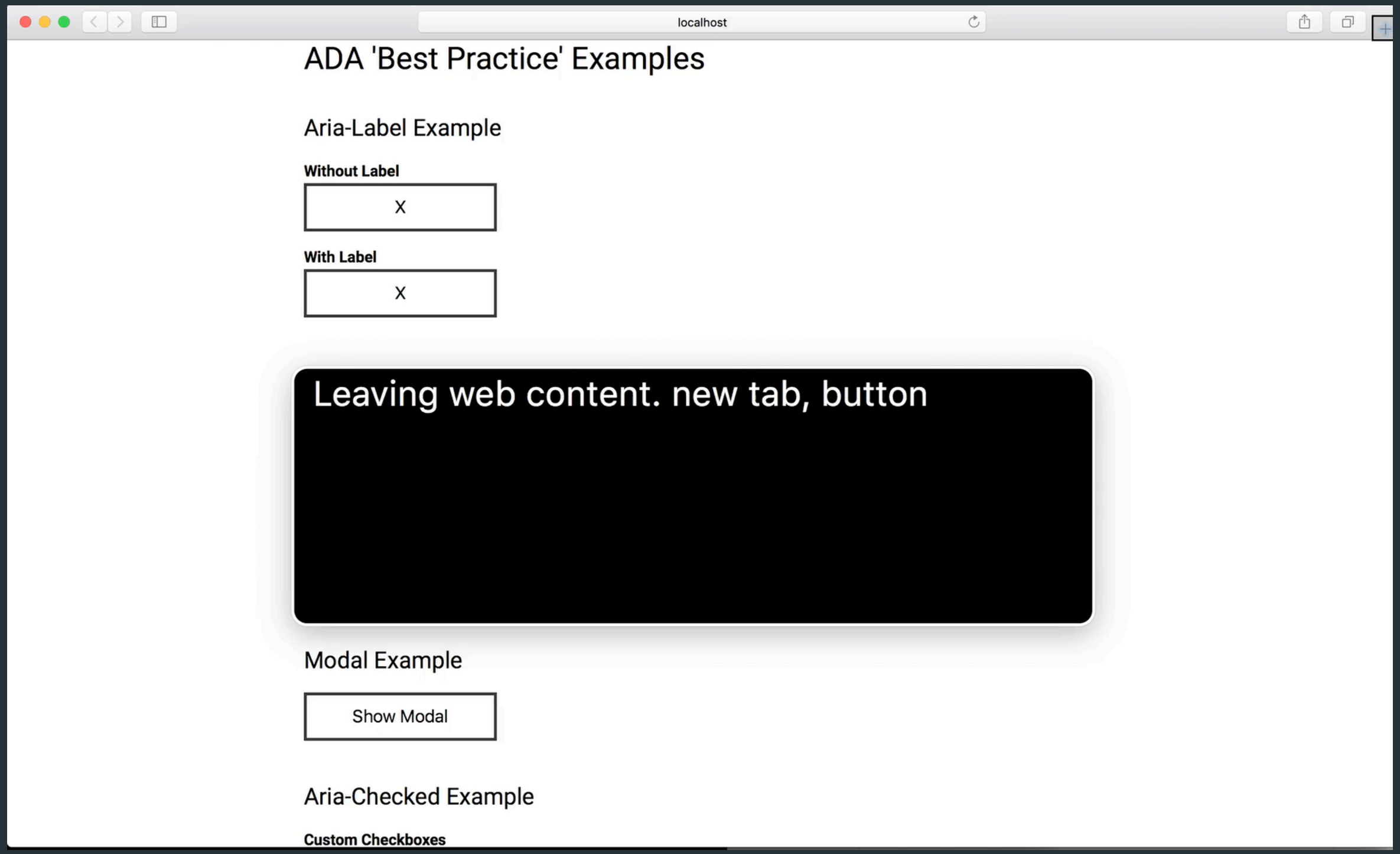
Source: https://developers.google.com/web/fundamentals/accessibility/semantics-aria/aria-labels-and-relationships

```
// Example 01
<button aria-label="Close Button">X</button>

// Example 02
<button aria-label="Close Button">
  <img src="/close-button.png" alt="Close Icon" />
</button>
```

ARIA-Label React Example

```
1  render() {
2      return (
3          <h3>Without Label</h3>
4          <Button
5              label="X"
6          />
7          <h3>With Label</h3>
8          <Button
9              ariaLabel="Close Menu"
10             label="X"
11         />
12     )
13 }
14
```

ARIA-Label Demo



ARIA-Expanded

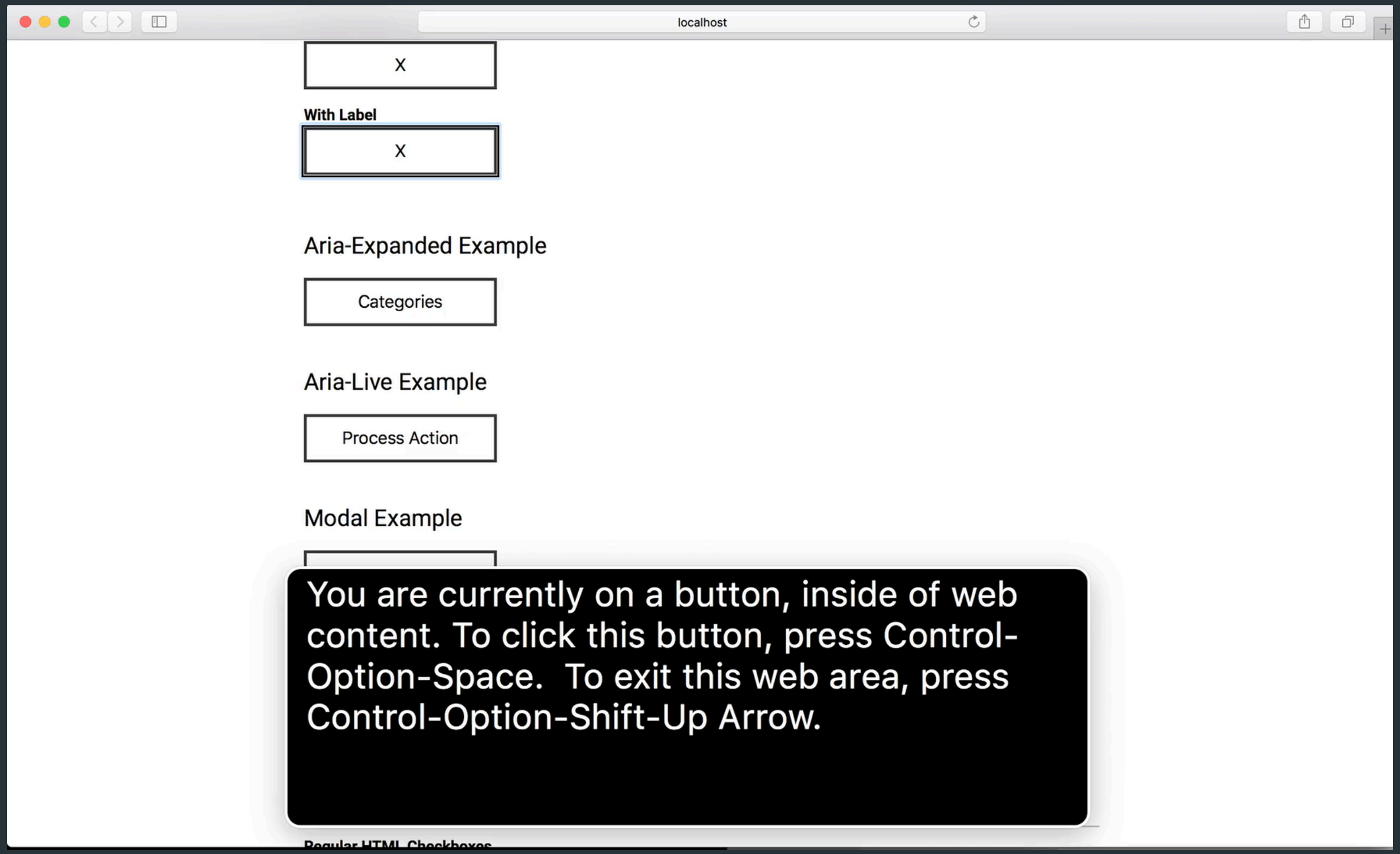
For every dropdown or display of hidden content, the aria-expanded attribute needs to be applied.

```
// When the dropdown or hidden content is not visible:
<button aria-expanded="false">
    <span>More</span>
</button>
<div style="visibility: hidden;">
    <span>Content</span>
</div>

// When the dropdown or hidden content is visible:
<button aria-expanded="true">
    <span>Less</span>
</button>
<div style="visibility: visible;">
    <span>Content</span>
</div>
```

ARIA-Expanded React Example

```
1 constructor() {
2     super();
3     this.state = {
4       expanded: false,
5     };
6     this.toggleExpanded = this.toggleExpanded.bind(this);
7 }
8
9 toggleExpanded() {
10     this.setState({ expanded: !this.state.expanded });
11 }
12
13 render() {
14     return (
15         <button aria-expanded={this.state.expanded} onClick={this.toggleExpanded}>
16           <span>Categories</span>
17         </button>
18
19         <ul className={!this.state.expanded ? ' invisible' : ''}>
20           <li><a href="/category-01"><span>Category 01</span></a></li>
21           <li><a href="/category-02"><span>Category 02</span></a></li>
22           <li><a href="/category-03"><span>Category 03</span></a></li>
23           <li><a href="/category-04"><span>Category 04</span></a></li>
24           <li><a href="/category-05"><span>Category 05</span></a></li>
25         </ul>
26     );
27 }
28
```

ARIA-Expanded Demo



ARIA-Live

This is used for all announcements, errors, and alerts. If anything on the page has changed and needs the attention of the user, this attribute should be used.

```
// Interrupts the current queue of messages and announces immediately.
<p aria-live="assertive">This action did not process</p>

// Does not interrupt the current queue of messages and adds it to the end.
<p aria-live="polite">This action did not process</p>

// Does not interrupt the user.
<p aria-live="none">This action did not process</p>

```

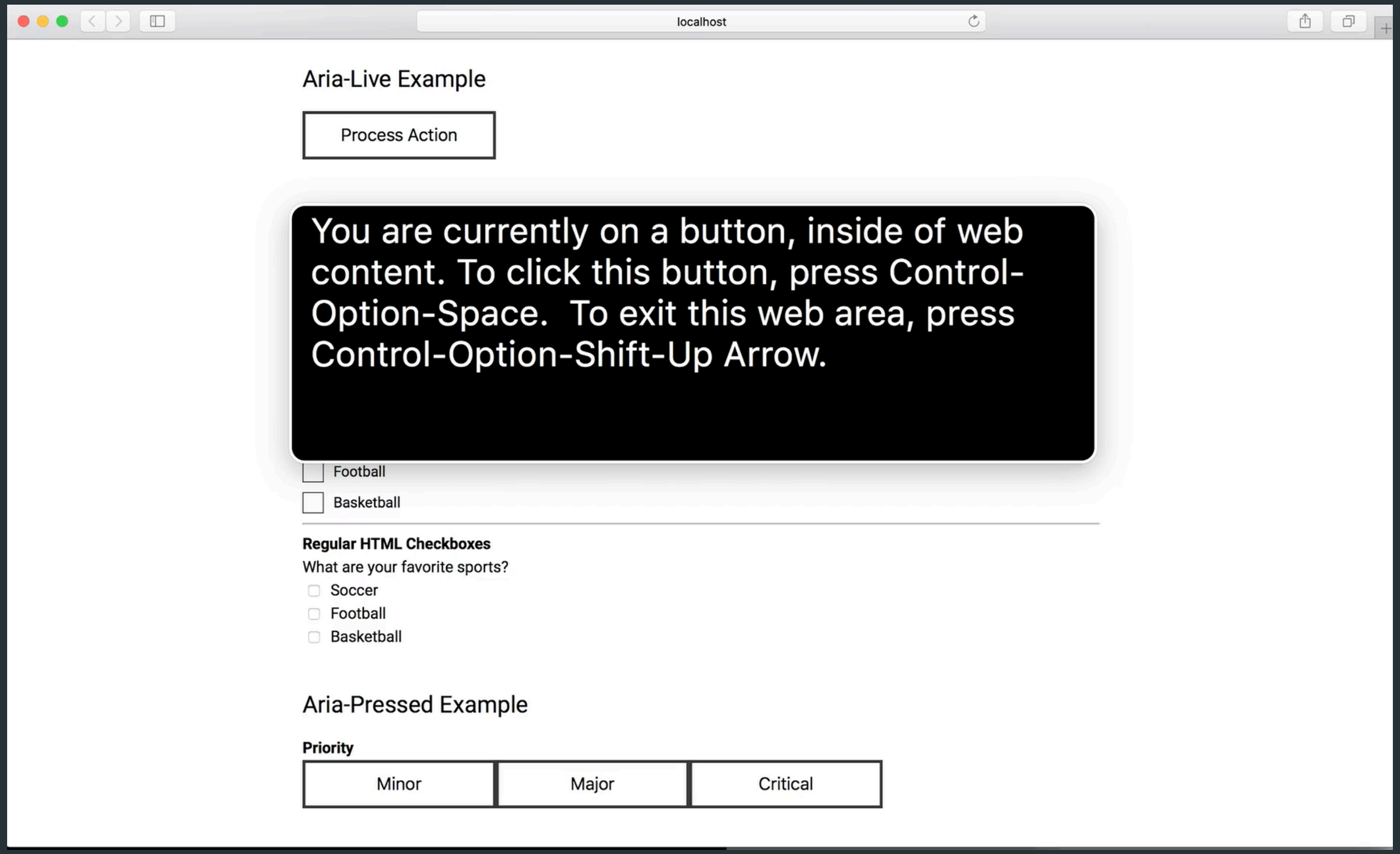
ARIA-Live React Example

```
constructor() {
    super();
    this.state = {
        errorMessage: true,
        repeatErrorMessage: false,
    };
    this.handleError = this.handleError.bind(this);
    this.handleErrorAction = this.handleErrorAction.bind(this);
}
handleError() {
    const error = '<p aria-live="assertive">This action did not process</p>';
    const errorWithSpace = '<p aria-live="assertive">This action did not process </p>';
    if (!this.state.repeatErrorMessage) {
        this.setState({ repeatErrorMessage: true });
        return errorWithSpace;
    }
    this.setState({ repeatErrorMessage: false });
    return error;
}

handleErrorAction() {
    const error = this.handleError();
    const errorMessageContainer = document.getElementById('error-message');
    const srErrorMessageContainer = document.getElementById('sr-error-message');
    if (this.state.errorMessage) {
        errorMessageContainer.removeAttribute('class');
        errorMessageContainer.innerHTML = error;
        srErrorMessageContainer.innerHTML = error;
    }
}

```

ARIA-Live React Example

```
// Render()
render() {
    return (
        <div id="container">
            <div className="sr-only">
                <div id="sr-error-message" />
            </div>
            <div id="error-message" className="hide" />
        </div>
    )
}

// CSS Classes
.sr-only {
    position: absolute;
    left: -10000px;
    width: 1px;
    height: 1px;
    overflow: hidden;
}
.hide {
    display: none;
}

```

ARIA-Live Demo



ARIA-Checked

This is used for custom checkboxes that are not in a form and don't use an input tag.

```
// aria-checked="true": The checkbox is checked
<span
    aria-checked="true"
    aria-labelledby="chk1-label"
    role="checkbox"
    tabindex="0"
/>
<label id="chk1-label">Remember my settings</label>

// aria-checked="false": The checkbox is not checked
<span
    aria-checked="false"
    aria-labelledby="chk1-label"
    role="checkbox"
    tabindex="0"
/>
<label id="chk1-label">Remember my settings</label>

```

ARIA-Checked React Example

```
1 constructor() {
2     super();
3     this.handleAriaChecked = this.handleAriaChecked.bind(this);
4 }
5
6 handleAriaChecked(event) {
7     let node;
8     if (event.target.id === 'soccer') {
9         node = this.ariaCheckedSoccer.current;
10    } else if (event.target.id === 'football') {
11        node = this.ariaCheckedFootball.current;
12    } else if (event.target.id === 'basketball') {
13        node = this.ariaCheckedBasketball.current;
14    }
15    const state = node.getAttribute('aria-checked');
16    if (event.type === 'click' || event.keyCode === 13 || event.keyCode === 32) {
17        if (state === 'true') {
18            node.setAttribute('aria-checked', 'false');
19            node.firstChild.setAttribute('class', 'custom-checkbox-input');
20        } else {
21            node.setAttribute('aria-checked', 'true');
22            node.firstChild.setAttribute('class', 'custom-checkbox-checked');
23        }
24        event.preventDefault();
25        event.stopPropagation();
26    }
27 }
28
```

ARIA-Checked React Example

```
1 render() {
2     return (
3         <h3>Custom Checkboxes</h3>
4         <h4>What are your favorite sports?</h4>
5         <div
6             aria-checked="false"
7             className="custom-checkbox-field"
8             id="soccer"
9             onClick={this.handleAriaChecked}
10            onKeyDown={this.handleAriaChecked}
11            role="checkbox"
12            tabIndex="0"
13            ref={this.ariaCheckedSoccer}
14        >
15            <span className="custom-checkbox-input" />
16            <span className="custom-checkbox-label">Soccer</span>
17        </div>
18        <div
19            aria-checked="false"
20            className="custom-checkbox-field"
21            id="football"
22            onClick={this.handleAriaChecked}
23            onKeyDown={this.handleAriaChecked}
24            role="checkbox"
25            tabIndex="0"
26            ref={this.ariaCheckedFootball}
27        >
28            <span className="custom-checkbox-input" />
29            <span className="custom-checkbox-label">Football</span>
30        </div>
31        <div
32            aria-checked="false"
33            className="custom-checkbox-field"
34            id="basketball"
35            onClick={this.handleAriaChecked}
36            onKeyDown={this.handleAriaChecked}
37            role="checkbox"
38            tabIndex="0"
39            ref={this.ariaCheckedBasketball}
40        >
41            <span className="custom-checkbox-input" />
42            <span className="custom-checkbox-label">Basketball</span>
43        </div>
44    );
45 }
46
```

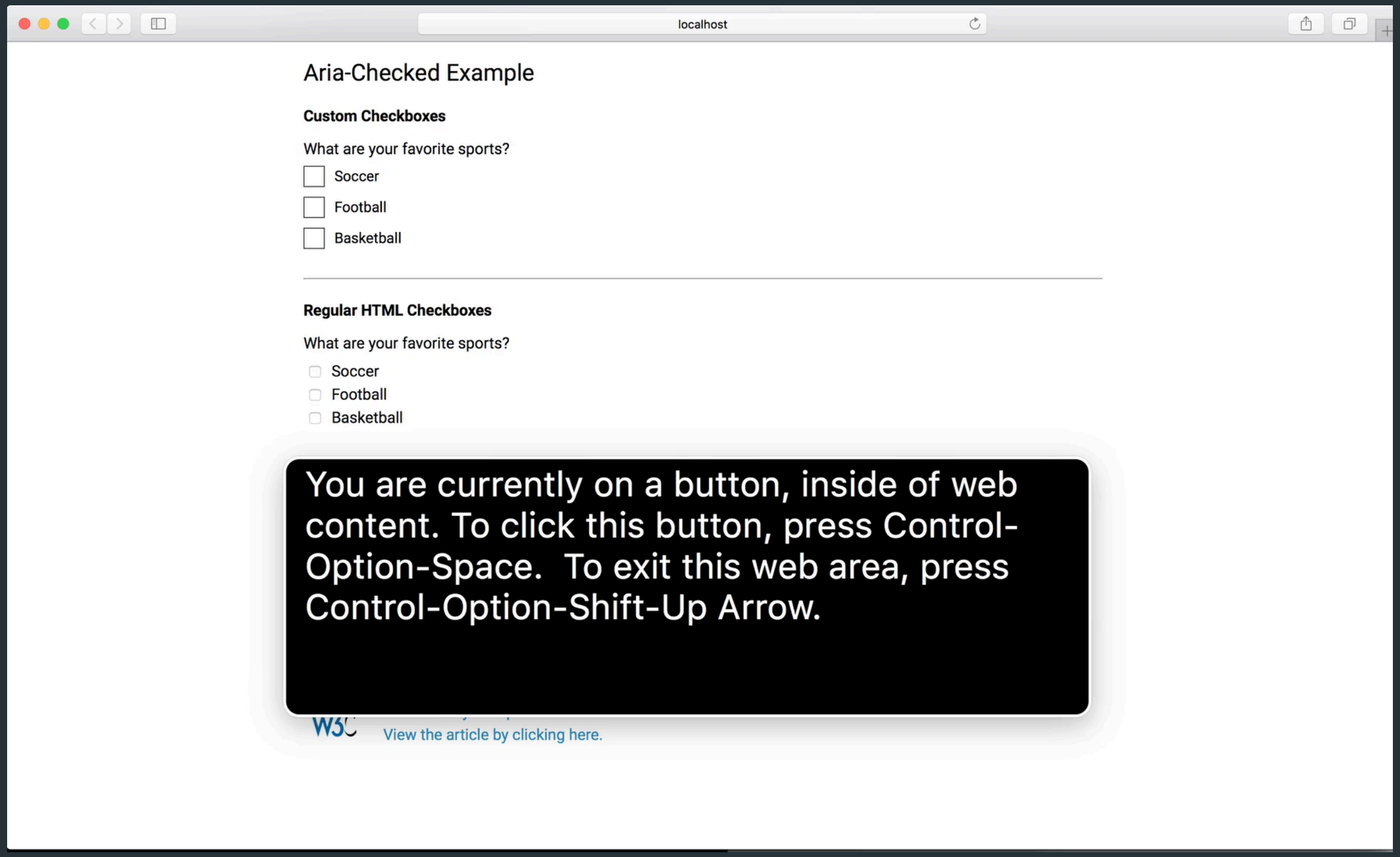
ARIA-Checked React Example

```
1  <div
2      aria-checked="false"
3      className="custom-checkbox-field"
4      id="soccer"
5      onClick={this.handleAriaChecked}
6      onKeyDown={this.handleAriaChecked}
7      role="checkbox"
8      tabIndex="0"
9      ref={this.ariaCheckedSoccer}
10 >
11     <span className="custom-checkbox-input" />
12     <span className="custom-checkbox-label">Soccer</span>
13 </div>
14
```

ARIA-Checked HTML Example

```
1 render() {
2     return (
3         <h3>Regular HTML Checkboxes</h3>
4         <h4>What are your favorite sports?</h4>
5         <form>
6             <input id="soccer" type="checkbox" />
7             <label htmlFor="soccer">Soccer</label>
8             <input id="football" type="checkbox" />
9             <label htmlFor="football">Football</label>
10            <input id="basketball" type="checkbox" />
11            <label htmlFor="basketball">Basketball</label>
12        </form>
13    );
14 }
15
```

ARIA-Checked Demo



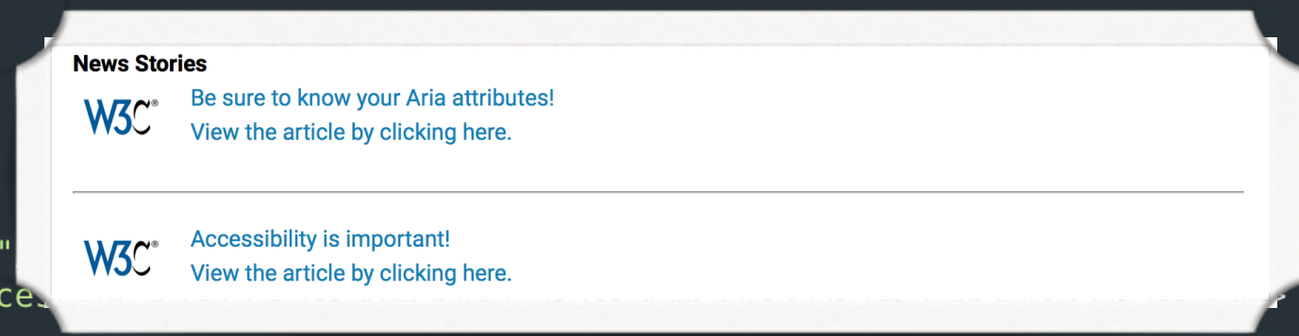
ARIA-Hidden

This is used for elements that you want to hide from the screen reader because they are visual only or hidden visually on the page.

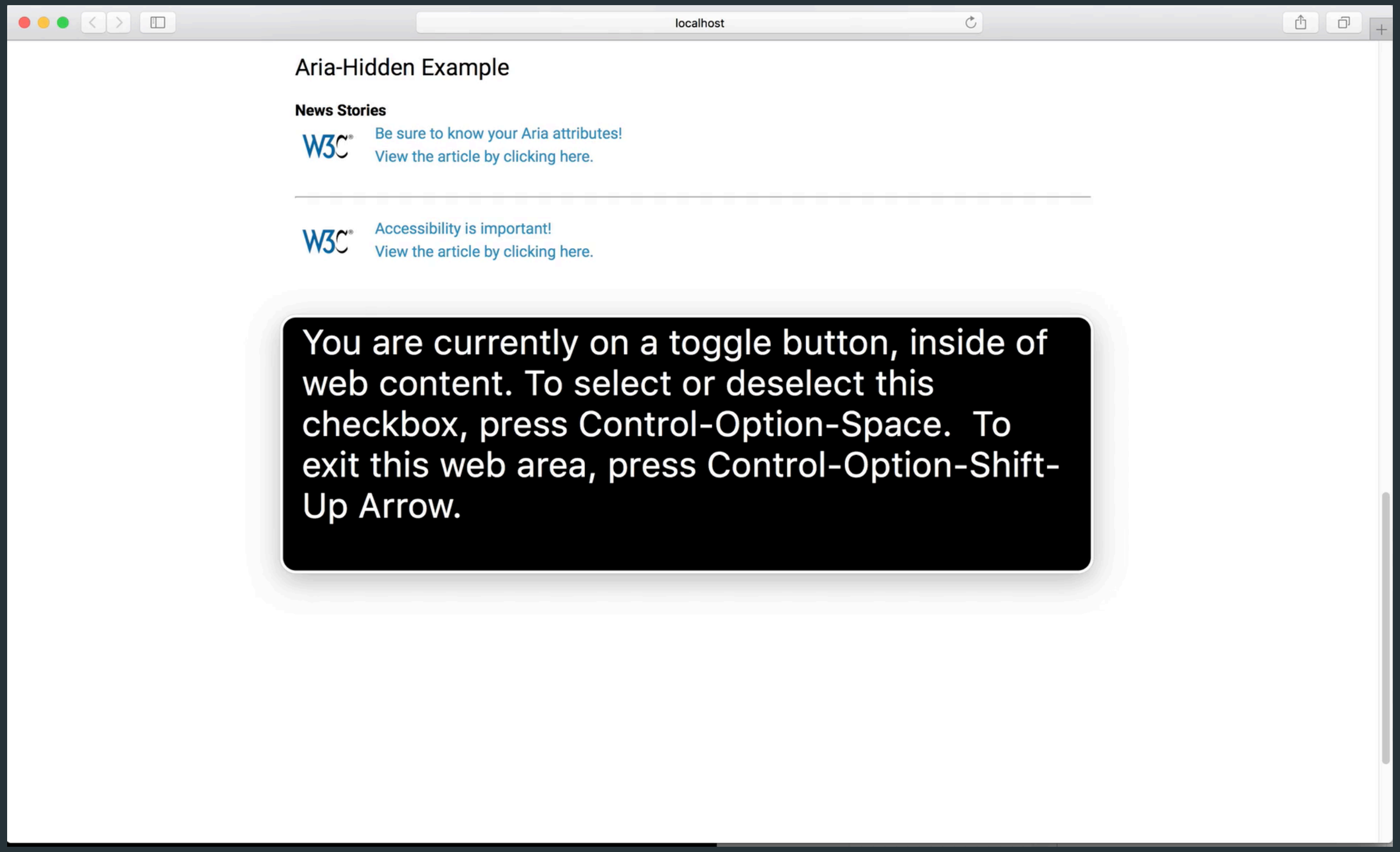
```html
<img
    alt="Decorative Top Border"
    aria-hidden="true"
    height="5"
    src="/images/decortive-top-border.png"
    width="50"
/>
<img
    alt="Oak Tree"
    aria-hidden="false"
    height="50"
    src="/images/oak-tree.png"
    width="50"
/>

```

ARIA-Hidden React Example

```
1  render() {
2      return (
3          <h3>News Stories</h3>
4          <div className="news-story">
5              <a className="news-story-image" href="
6                  <img alt="W3 Icon" aria-label="Acce
7              </a>
8              <a aria-hidden="false" className="news-story-link" href="/news/story.html">
9                  Be sure to know your Aria attributes!
10             </a>
11             <a aria-hidden="false" className="news-story-link" href="/news/story.html">
12                 View the article by clicking here.
13             </a>
14         </div>
15         <hr />
16         <div className="news-story">
17             <a className="news-story-image" href="/news/story.html">
18                 <img alt="W3 Icon" aria-label="Accessibility is important!" height="50" src={W3icon} width="68" />
19             </a>
20             <a aria-hidden="true" className="news-story-link" href="/news/story.html" tabIndex="-1">
21                 Accessibility is important!
22             </a>
23             <a aria-hidden="true" className="news-story-link" href="/news/story.html" tabIndex="-1">
24                 View the article by clicking here.
25             </a>
26         </div>
27     );
28 }
29
```



ARIA-Hidden Demo



Most Common Accessibility Concepts

Buttons

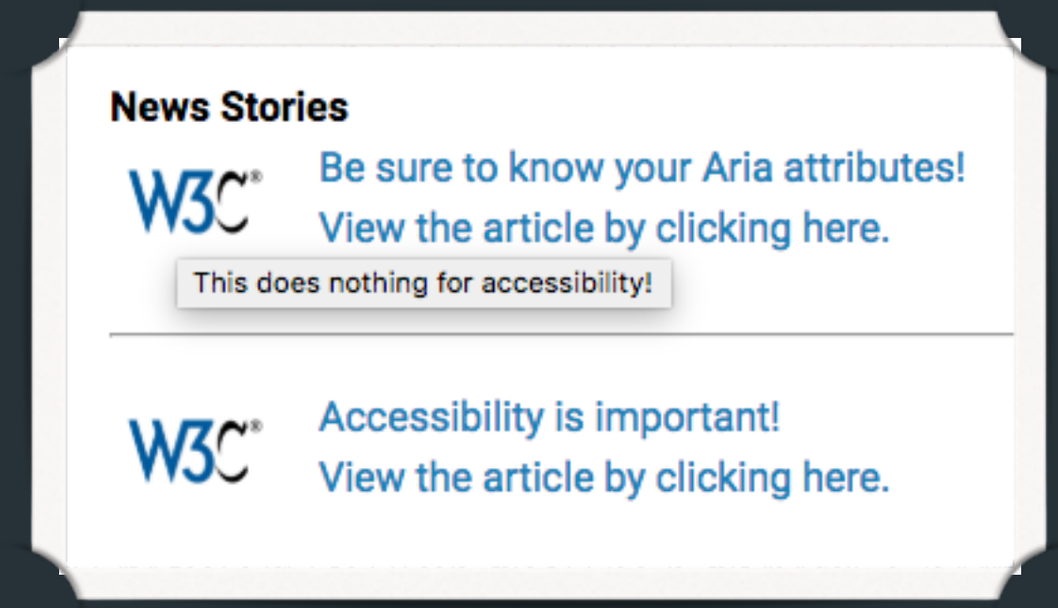
- Buttons are used for any action that keeps the user on the current page
- Use a Button component that uses a native HTML button (<button />) tag
- Make sure to include onKeyDown actions
- Toggle buttons need to use aria-expanded attribute and change the value based on state

Links

- Use links for any action that is leading the user to a new page/url
- Use native anchor (**<a />**) tag when possible
- Use **role="link"** for divs that do not use the native anchor tag
- If anchor tag doesn't have an href, then a **tabIndex="0"** must be applied. Otherwise the element is skipped and not detected as interactive

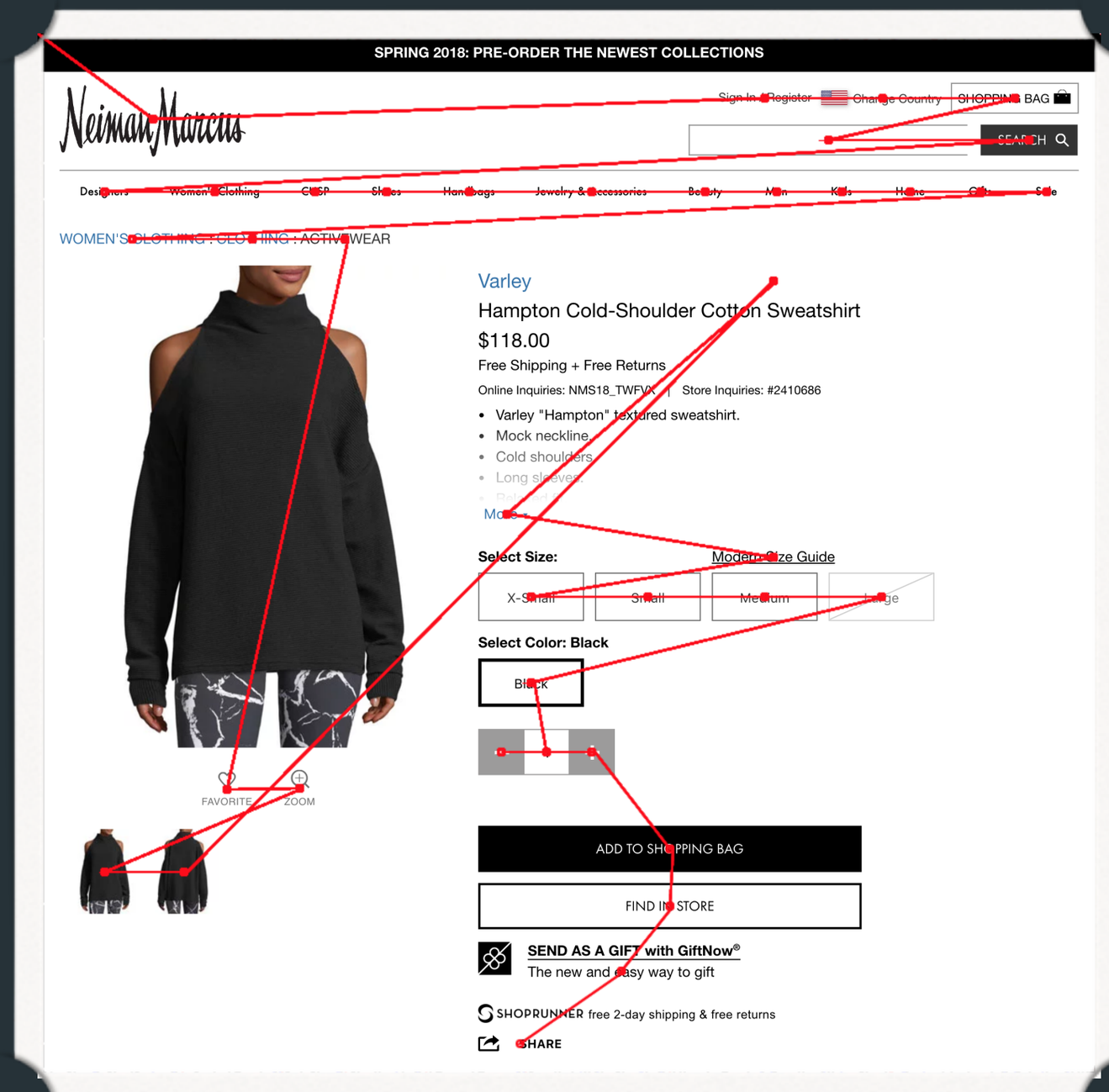
Images

- Use native image (**<img />**) tag
- Every image must have **alt** text
- Alt text should present the content and function of the image. It should also be succinct.
- If image is decorative only, then use the aria-hidden="true" attribute
- Make sure the images are high enough quality to be decipherable when the page is zoomed.
- Title text



Tabbing Order

- Extremely important for keyboard users
- Ideally should follow the DOM order
- Needs to make sense from a visual representation of the page
- Moves sequentially from left to right and top to bottom
- Consists of interactive elements only



Google Chrome Extension: ChromeLens

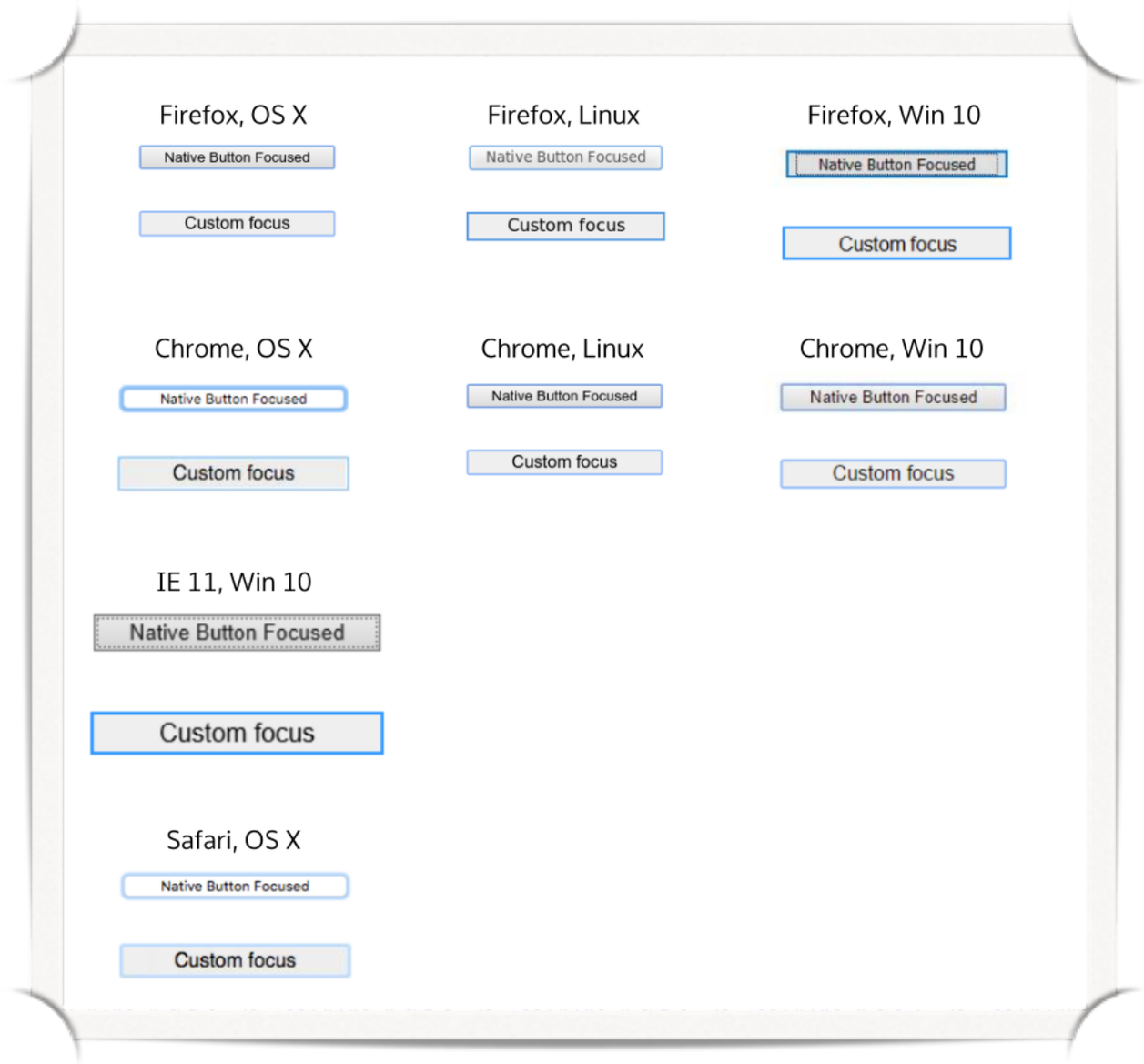
Tabindex

- **tabindex="1"** (or any number greater than 1) defines an explicit tab order. This is almost always a bad idea.
- **tabindex="0"** allows elements besides links and form elements to receive keyboard focus. It does not change the tab order, but places the element in the logical navigation flow, as if it were a link on the page.
- **tabindex="-1"** allows things besides links and form elements to receive "programmatic" focus, meaning focus can be set to the element through scripting, links, etc.

Source: https://webaim.org/techniques/keyboard/tabindex

Focus

- Never manipulate focus unless it is absolutely necessary.
- **NEVER** remove the CSS **outline** attribute unless it is replaced by a custom version. It makes your site completely unaccessible for keyboard users.
- Use **document.activeElement** to get the currently focused item.
- Each browser has its own focus outline



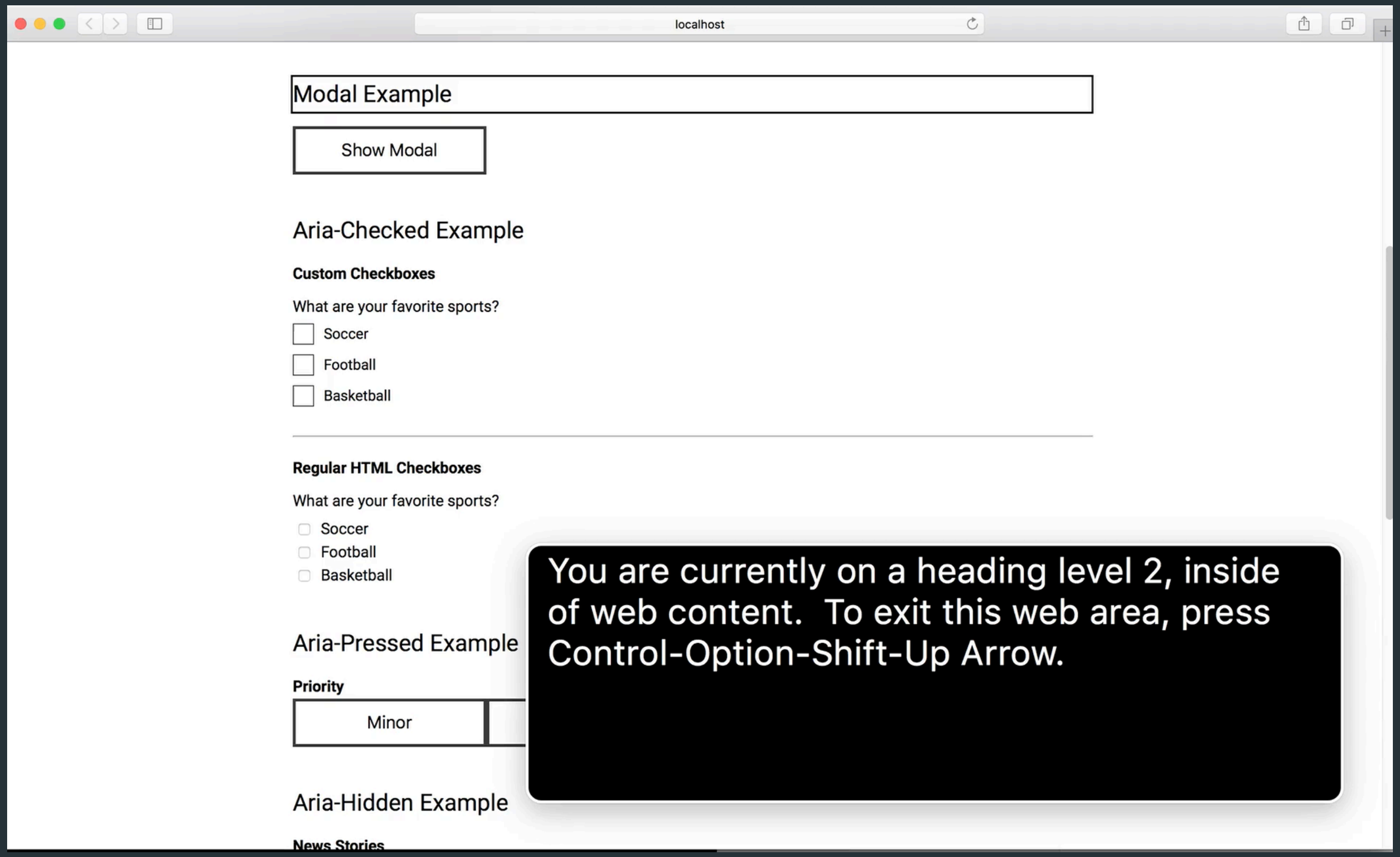
Source: https://ghinda.net/article/mimic-native-focus-css/

Modals

- Use **role="dialog"**
- Store previously focused element
- Once opened, the modal should trap focus so that no background elements can be tabbed.
- Focus should go to either the modal window itself or the first interactive element.
- **ESC** should close the modal
- When the modal is closed, it should go back to the element that triggered the opening action.

```javascript
const tabbableNode = /input|select|textarea|button|object/;

function hidesContents(element) {
  const zeroSize = element.offsetWidth <= 0 && element.offsetHeight <= 0;

  // If the node is empty, this is good enough
  if (zeroSize && !element.innerHTML) return true;

  // Otherwise we need to check some styles
  const style = window.getComputedStyle(element);
  return zeroSize
    ? style.getPropertyValue('overflow') !== 'visible'
    : style.getPropertyValue('display') === 'none';
}

function visible(element) {
  let parentElement = element;
  while (parentElement) {
    if (parentElement === document.body) break;
    if (hidesContents(parentElement)) return false;
    parentElement = parentElement.parentNode;
  }
  return true;
}

function focusable(element, isTabIndexNotNaN) {
  const nodeName = element.nodeName.toLowerCase();
  const classes = hasClass(element, 'slick-disabled');
  let res =
    ((tabbableNode.test(nodeName) && !element.disabled) ||
    (nodeName === 'a' ? element.href || isTabIndexNotNaN : isTabIndexNotNaN));
  if (classes) {
    res = !classes;
  }
  return res && visible(element);
}

function hasClass(element, classes) {
  return (` ${element.className} `).indexOf(` ${classes} `) > -1;
}

function tabbable(element) {
  let tabIndex = element.getAttribute('tabindex');
  if (tabIndex === null) tabIndex = undefined;
  const isTabIndexNaN = isNaN(tabIndex);
  return (isTabIndexNaN || tabIndex >= 0) && focusable(element, !isTabIndexNaN);
}

export default function findTabbableDescendants(element) {
  return [].slice.call(element.querySelectorAll('*'), 0).filter(tabbable);
}

```

Modal Demo


localhost
Modal Example
Show Modal
Aria-Checked Example
Custom Checkboxes
What are your favorite sports?
Soccer
Football
Basketball
Regular HTML Checkboxes
What are your favorite sports?
Soccer
Football
Basketball
Aria-Pressed Example
Priority
Minor
Aria-Hidden Example
News Stories
You are currently on a heading level 2, inside of web content. To exit this web area, press Control-Option-Shift-Up Arrow.

Conclusion

- Web accessibility means making the web available to everyone.
- As developers it is our responsibility.
- We get many benefits from making our site accessible.
- Be sure to test your stories on screen readers manually
- Use native elements whenever possible
- Adding ARIA attributes helps our site become more accessible.
- Be aware of the tabbing order
- Make sure your focus ring is easily visible
- Mind your modals

Resources

Google’s Accessibility Web Fundamentals

https://developers.google.com/web/fundamentals/accessibility/

WebAIM's WCAG 2.0 Checklist

https://webaim.org/standards/wcag/checklist

WebAIM Color Contrast Checker

https://webaim.org/resources/contrastchecker/

WAI-ARIA Authoring Practices 1.1

https://www.w3.org/TR/wai-aria-practices-1.1/

Questions?

interactivejohn.com
@interactivejohn
github.com/interactivejohn

Thank You!