TDL SCHOOL

Web A11y for Developers

Day 4: Best Practices

TDL SCHOOL

Agenda

Day 1: Introduction

Day 2: Components

Day 3: Accessible Forms

Day 4: Best Practices

Best Practices

So far we focused on more **higher level concepts** (building UI components and accessible forms).

Today we will look into more **detailed principles of accessibility**.

On today's agenda

- General best practices
- Bite-sized tips & tricks
- Moving beyond development

Best Practices: Focus management

2.4.3 Focus Order

We need to make sure the focus order follows the **natural visual flow** of our page.

Users need to predictably navigate through the page.

Avoid “tabindex” values larger than 0.

Sometimes we may have to **intentionally direct the focus indicator** to specific locations to ensure this natural flow.

Best Practices: Focus management

Clearable Inputs

Inputs (text, email etc.) with a “Clear” icon button that resets the value of the input field.

How should it work?

After pressing “Clear” button - **move focus to the input**.

If we don’t, focus will be moved on the **<body>** element - “dead focus” done by the browser.

index.html

```html
<div class="clearable-input">
  <input type="text" id="clearable-name" placeholder="Your name here..." >
  <button type="button" class="clear invisible" aria-label="Clear">×</button>
</div>
```

input.js

```
const input = document.querySelector('.clearable-input input');
const clearButton = input.nextElementSibling;

// Show the "Clear" button only when the input has a value entered
input.addEventListener('input', () => {
  clearButton.classList.toggle('invisible', input.value.length === 0);
});

// Reset the input and move focus on it
clearButton.addEventListener('click', () => {
  clearButton.classList.add('invisible');

  input.value = '';
  input.focus();
});
```

Demo Time!

Focus management - Clearable input

Best Practices: Focus management

Deletable options

List of items such that **any item can be removed from the list** by pressing a “Remove” button on the item itself.

How should it work?

Once an item is deleted (“Remove” button is pressed) - **move focus to the appropriate logical next position**.

Deleting the last element → move focus to the item **before it**.
Deleting any other element → move focus to the item **after it**.

index.html

```html
<ul class="options-list">
  <li>
    <p>Clean up bedroom and kitchen</p>
    <button class="remove" aria-label="Remove item">&times;</button>
  </li>
  <li>
    <p>Take the clothes off the drying rack</p>
    <button class="remove" aria-label="Remove item">&times;</button>
  </li>
  <li>
    <p>Read "Three Body Problem"</p>
    <button class="remove" aria-label="Remove item">&times;</button>
  </li>
  <li>
    <p>Finish accessibility course</p>
    <button class="remove" aria-label="Remove item">&times;</button>
  </li>
  <li>
    <p>Get a good weekend's worth of sleep</p>
    <button class="remove" aria-label="Remove item">&times;</button>
  </li>
</ul>

<button>Reset list</button>
```

list.js

```
const optionsList = document.querySelector('.options-list');
const resetListButton = optionsList.nextElementSibling;

// Event delegation for better performance - fewer click listeners
optionsList.addEventListener('click', (e) => {
  const removeButton = e.target.closest('button.remove');
  if (!removeButton) return;

  const currentOptions = Array.from(optionsList.querySelectorAll('li'));
  const deletingLastElement = currentOptions.length === 1;

  const listElement = removeButton.parentElement;
  const idx = currentOptions.indexOf(listElement);

  if (idx === -1) return;

  let newIdx;

  if (idx === currentOptions.length - 1) {
    // Deleting last element - move focus to previous element
    newIdx = Math.max(idx - 1, 0);
  } else {
    // Deleting any other element - move focus to next element
    newIdx = idx + 1;
  }

  if (deletingLastElement) {
    // Focus an "Add item" button or something similar
    resetListButton.focus();
  } else {
    // Focus the new item (depends on what you need to focus on your own items)
    currentOptions[newIdx].querySelector('button.remove')?.focus();
  }
  // Delete the clicked item
  listElement.remove();
});
```

Demo Time!

Focus management - Deletable options

Best Practices: Tooltips

One of the **most widely used pieces of UI** in any Web application.
It is often implemented or used incorrectly.

Tooltips (“tips for tools”)

Small popups that appear next to an interactive element.
They help us understand that element’s meaning/purpose.

What are they used as?

Tooltips can be used as **primary labels** or **additional description** for any given interactive element.

Best Practices: Tooltips

Tooltips as primary labels

Only do this if there is **absolutely no space to include a visible label** - last resort.

Programmatically connect the tooltip element to the element it is labeling using **aria-labelledby**.

The tooltip needs to have the **role** of **“tooltip”**.
It must open when we **“hover”** or **“focus”** on the interactive element.

It must also stay open while we hover over the tooltip itself.

index.html

```html
<button aria-labelledby="notifications-tooltip">
  <notification-bell-icon>
</button>

<div id="notifications-tooltip" role="tooltip">
  <div class="tooltip-content">Notifications</div>
</div>
```

Best Practices: Tooltips

Traditional positioning

Set up by using **JavaScript** to calculate the interactive element's position and size.

We can also add a **wrapper element** around both and use **absolute positioning** on the tooltip.

Modern positioning

Use the Baseline 2026 newly available **CSS anchor positioning**.

styles.css

```css
button {
  anchor-name: --tooltip-anchor;

  & + [role="tooltip"] {
    position: absolute;
    position-anchor: --tooltip-anchor;
    top: anchor(bottom);
    left: anchor(left);

    visibility: hidden;
    opacity: 0;
  }
}
```

styles.css

```css
button:is(:hover, :focus-visible) + [role="tooltip"] {
  visibility: visible;
  opacity: 1;
}
```

styles.css

```css
button:is(:hover, :focus-visible) + [role="tooltip"],
[role="tooltip"]:hover {
  visibility: visible;
  opacity: 1;
}
```

Demo Time!

Tooltips as labels

Best Practices: Tooltips

Tooltips as additional descriptions

We can use them as descriptions on elements that are **already labeled** (visually or not).

The set up is similar, we just connect the tooltip and interactive element using **aria-describedby**.

index.html

```html
<button aria-label="Notifications" aria-describedby="notifications-tooltip">
  <notification-bell-icon>
</button>

<div id="notifications-tooltip" role="tooltip">
  <div class="tooltip-content">
    Check out the latest activity across your network
  </div>
</div>
```

Let’s discuss!

How should we handle tooltips on mobile devices?

Best Practices: Tooltips

Toggletips

Similar to tooltips, but they are **toggled by the “click” event**.

Toggletips are buttons whose sole purpose is to **show the additional information it is hiding**.

They can only provide additional descriptions - not labels.

They’re content is initially hidden. Users must intentionally click on them to reveal it - we can use **live regions**.

index.html

```html
<label for="emp-id">Employee ID</label>
<input id="emp-id" type="text" inputmode="numeric" placeholder="Employee ID">

<button aria-label="More info" data-toggletip="Last 4 digits of your birthday">
  i
  <div role="status"></div>
</button>
```

toggletip.js

```
const toggletips = document.querySelectorAll('[data-toggletip]');

toggletips.forEach((toggletip) => {
  const liveRegion = toggletip.querySelector('[role="status"]');
  const content = toggletip.getAttribute('data-toggletip');

  toggletip.addEventListener('click', (e) => {
    liveRegion.textContent = e.target === toggletip ? content : '';
  });

  toggletip.addEventListener('blur', () => {
    liveRegion.textContent = '';
  });
});
```

Demo Time!

Toggletips

Tips & Tricks

document.activeElement

Property on the **`document`** object that always references the **currently focused element on the page**.

Very helpful for **accessibility debugging** (ex. big hidden menus/sidebars with focusable elements).

Monitor the value of this property within Chromium browsers' DevTools Live Expressions feature.

Demo Time!

document.activeElement + Live Expressions

Tips & Tricks

Linter stylesheets

During development, we can forget things like **adding alt-text to images**, **adding an accessible label to an icon** (svg) etc.

Instead of relying on automated tools and third party linter libraries for these small mistakes - we can **leverage CSS**.

We can write custom CSS styles written for **specific selectors of inaccessibly set up elements**.

styles.css

```css
/* Images that don't have an "alt" attribute */
img:not([alt]) {
  outline: 6px solid red !important;
  outline-offset: 2px;
}

/* SVG elements that are not hidden and not labelled */
svg:not([aria-hidden="true"]):not([aria-label]):not([aria-labelledby]) {
  outline: 6px solid darkorange !important;
  outline-offset: 2px;
}
```

Demo Time!

Linter stylesheets

Tips & Tricks

Visually hidden elements

Elements that are **present in the DOM and the Accessibility tree**, but are not visually shown on the screen.

They shouldn’t interrupt or break the visual layout.

We can use them to add additional screen-reader-only descriptions and provide more context to certain elements/sections.

styles.css

```css
.sr-only {
  clip: rect(0 0 0 0);
  clip-path: inset(50%);
  height: 1px;
  overflow: hidden;
  position: absolute;
  white-space: nowrap;
  width: 1px;
}
```

index.html

```html
<!-- "Loading" message within a loading indicator -->
<div>
  <div class="spinner"></div>
  <span class="sr-only">Loading images...</span>
</div>

<!-- Invisible Live regions that announce changes -->
<div role="status">
  <span class="sr-only">15 results found</span>
</div>

<!-- Providing additional context to vaguely named links/buttons -->
<a href="https://testdevlab.com">
  Read more <span class="sr-only">about the company I work at</span>
</a>
```

Tips & Tricks

High Contrast mode

Accessibility feature that users can enable on their devices.

Uses a **limited color palette with a high contrast** to help users with low vision read the content on their screens easier.

Custom focus indicators like the ones made using **`box-shadow`** can break on High Contrast mode, if we disabled the outline using **`outline: none`**.

styles.css

```css
button:focus-visible {
  /* This will cause you trouble in High Contrast mode */
  outline: none;

  /* Use this instead */
  /* outline-color: transparent; */

  box-shadow: 0px 0px 10px 10px var(--brand-primary);
}
```

Demo Time!

High Contrast mode

Tips & Tricks

DevTools features

Modern browsers (especially Chromium based ones) provide an extensive list of helpful tools.

Emulator

- Emulate “Reduced Motion” directly in the browser
- Emulate High Contrast mode (**forced colors**)
- Emulate color deficiencies like Protanopia, Deuteranopia, Tritanopia, and Achromatopsia

Tips & Tricks

DevTools features

Modern browsers like Chrome, Firefox and Edge have an **Accessibility Tree** visualized built into the “Inspect” panel.

Accessibility tree

Hierarchical structure similar to the DOM.

Assistive technologies use it to parse the page’s contents and announce it.

Demo Time!

DevTools features

Beyond Development: A11y in the Pipeline

Design → Development → Code Review → Testing → Deployment

Q: Where should “Accessibility” as a “task” be placed here?

A: Everywhere.

Shift-Left testing

Testing mechanism that intends to **catch and resolve issues as early as possible in the development pipeline.**

Beyond Development: A11y in the Pipeline

Design stage

Developers and designers need to be on the same page and in **constant communication** to be able to **identify accessibility issues early in the design process**.

Look out for **color contrast issues**, **missing focus indicators**, ask questions about **animations**, **font sizes**, **spacing**, **hover and errors states** etc.

Make sure you **lead with empathy**.

Beyond Development: A11y in the Pipeline

Development stage

Apply lessons we learned on building **accessible UI components**, rules and guidelines for creating **accessible forms**.

Utilize the best practices and small-win tips & tricks we covered today - they require very little effort.

Do your best to **use native HTML** as much as possible and **understand the ARIA attributes** you are using.

Make sure to cover any **keyboard controls** or **focus management** flows.

Beyond Development: A11y in the Pipeline

Testing stage

Make sure **accessibility requirements** are included in any feature's **definition of done**.

Our testers need to **enforce these requirements** themselves.

Accessibility is just as important as functionality.

Beyond Development: A11y in the Pipeline

Code Review & Deployment stages

Make sure to establish a **comprehensive automated** (and manual code review) **process** that can catch obvious accessibility issues.

One of the best ways to do that using automation is to integrate **accessibility linters** into our codebase and CI/CD pipeline.

Beyond Development: A11y in the Pipeline

Accessibility linters: React

eslint-plugin-jsx-a11y

Plugin for your ESLint configuration file.
It **analyzes static code** so that you can be notified of a potential accessibility issue as you are writing it.

@axe-core/react

Can be integrated in any React application and it will look for **accessibility issues on the actual rendered pages** in your browser.

Beyond Development: A11y in the Pipeline

Accessibility linters: Vue

eslint-plugin-vuejs-accessibility

Plugin for your ESLint configuration file.
Designed to look for issues in `**.vue**` files.

vue-axe-next

Analyzes and reports on accessibility issues found during the runtime of your application. Based on the axe-core engine.

Beyond Development: A11y in the Pipeline

Accessibility linters: Angular

@angular-eslint/eslint-plugin-template

Designed for linting Angular template files and includes accessibility rules as well.

We can also manually integrate the base axe-core engine into any Angular application to get accessibility scanning at runtime.

Beyond Development: A11y in the Pipeline

Accessibility linters: Svelte

Worth mentioning as a new Web framework.

Svelte's custom compiler designed to compile **`.svelte`** files - has its own built-in accessibility linter out of the box.

Beyond Development: A11y in the Pipeline

Accessibility testing frameworks

We can also integrate accessibility scanning and reporting in our **automation testing framework**.

Popular examples include @axe-core/playwright and cypress-axe.

Conclusion

Accessibility needs to be present throughout the entire development pipeline, from start to finish.

Beyond Development: Browser extensions

Chrome's Lighthouse is probably the most well known. It provides Accessibility, SEO, Performance and other similar audits for free.

The Axe DevTools extension is a very popular one - it has a free and a paid version.
They also have a VS Code extension called axe Accessibility Linter.

You can also check out WAVE by WebAIM.

We should ideally try to **use multiple of these tools**, to make sure we have as much coverage as possible.

Beyond Development: No mouse days

Started by The A11y Project.

Movement meant to **spread awareness of accessibility** and bring the experience of keyboard-only users closer to everyone else.

Designate 1 day in the week where the entire team will just not use a mouse for the whole day*.

* Lunch break and urgent production issues do not apply

NPM package no-mouse-days that hides the mouse pointer.

Beyond Development: Knowledge sharing

Write **comprehensive and understandable documentation** on proper accessible usage and implementation of your code.

Share your knowledge by **holding accessibility training sessions, courses** or even **small team-wide presentations**.

We must be the ones that will champion this movement of bringing the culture of accessibility closer to our teams.

We must set the example and **lead with empathy**.

Conclusion

Creating a more accessible Web is impossible without **collaboration**.

If we can help even one additional person have a better experience on the Web - it is definitely worth the effort.

Thank you all for joining, listening and being a part of this course - I truly hope you learned something.

If you have any questions, you can reach me at daniel.shijakovski@testdevlab.com.

Thank you!