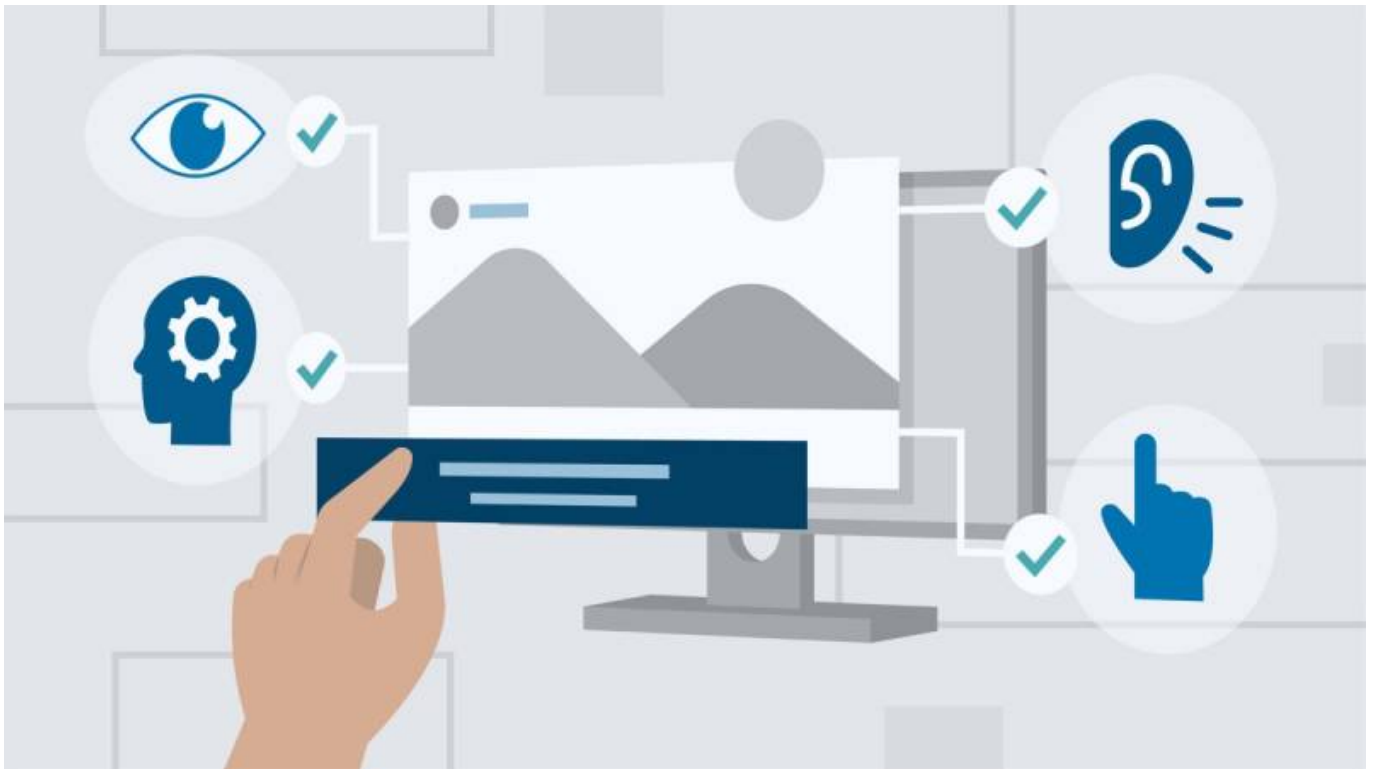


LU06b - Accessibility (A11y)



Weiterführende Informationen: Dieser Block stützt sich auf den Google-Kurs [web.dev - Learn Accessibility](https://web.dev/learn-accessibility/).

Was ist Accessibility (A11y)?

Accessibility (Barrierefreiheit, abgekürzt **a11y** - «a», 11 Buchstaben, «y») bedeutet: Websites und Apps so gestalten, dass Menschen mit Beeinträchtigungen sie gleichwertig nutzen können.

Laut WHO haben über **15 % der Weltbevölkerung** - rund 1,3 Milliarden Menschen - eine Beeinträchtigung. Dazu gehören:

- **Sehbeeinträchtigungen** - Blindheit, Sehschwäche, Farbenblindheit
- **Motorische Einschränkungen** - kein Maus-Einsatz, nur Tastatur oder Spracheingabe
- **Kognitive Einschränkungen** - Leseschwäche, ADHS, Konzentrationsschwierigkeiten
- **Hörbeeinträchtigungen** - relevant besonders für Audio- und Video-Inhalte

Eine schlecht zugängliche Website kann für jemanden, der einen Screenreader benutzt, komplett unbenutzbar sein - genau wie eine fehlende Rampe jemanden im Rollstuhl ausschliesst.

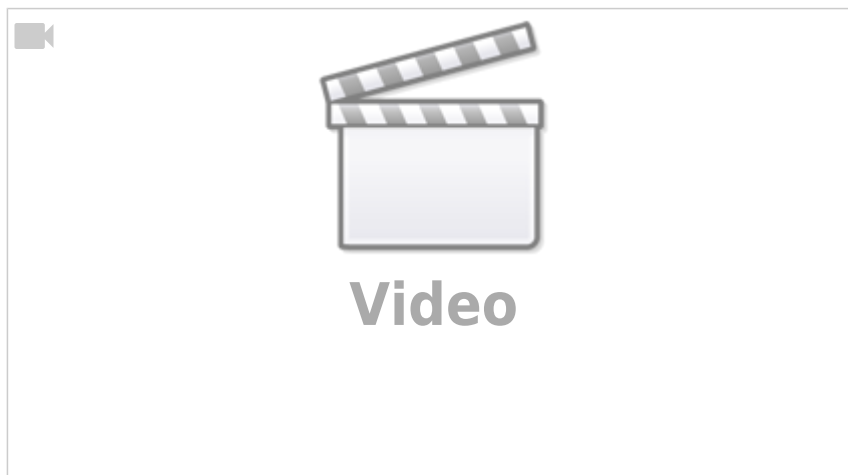
Warum ist A11y wichtig?



Barrierefreiheit ist kein «Nice-to-have» mehr – sie ist aus drei Gründen relevant:

- **Inklusion:** Das Hauptziel ist, allen Menschen die uneingeschränkte Teilnahme am digitalen Leben zu ermöglichen.
- **Gesetzliche Anforderungen:** Viele Länder haben Gesetze, die auf den WCAG basieren. Seit 2025 ist digitale Barrierefreiheit in der EU für die meisten Websites und Online-Shops gesetzlich vorgeschrieben.
- **Bessere UX für alle:** Barrierefreie Massnahmen – wie klare Fehlermeldungen, logischer Aufbau und gute Kontraste – verbessern die Nutzung für **alle** User, nicht nur für Menschen mit Einschränkungen.

WCAG - Der internationale Standard




Die **Web Content Accessibility Guidelines (WCAG)** werden vom World Wide Web Consortium (W3C) entwickelt und gelten als internationaler Standard für digitale Barrierefreiheit. Sie bieten Entwickler:innen klare, prüfbare Kriterien dafür, wie eine zugängliche Website aussehen muss.

POUR - Die vier Grundprinzipien


WAHRNEHMBARKEIT (PERCEIVABLE)

- ALT-TEXT BEI BILDERN
- UNTERTITEL BEI VIDEOS
- AUSREICHEND KONTRAST



```

```




```
<video controls>
  <source src="tutorial.mp4" />
  <track src="subtitle.vtt" kind="subtitles" srclang="de" />
</video>
```

AUSREICHEND KONTRAST
(zwischen Text und Hintergrund)


BEDIENBARKEIT (OPERABLE)

- NAVIGATION VIA TASTATUR, SPRACHEINGABE & ASSISTIVE TECHNOLOGIEN
- AUSSAGEKRÄFTIGE LINK-TEXTE
- AUSSAGEKRÄFTIGE LABELS UND SEITENTITEL




```
<button>
  Weitere Produkte anzeigen
</button>
```

- automatisch fokussierbar
- funktioniert mit Tastatur
- Screenreader erkennt Button




```
<head>
  <title>
    Online-Shop - Produktliste
  </title>
</head>
```



```
<form>
  <label for="name">
    Vollständiger Name
  </label>
  <input id="name" type="text" />
</form>
```


VERSTÄNDLICHKEIT (UNDERSTANDABLE)

- ANGEBEN, IN WELCHER SPRACHE DIE SEITE VERFASST IST
- KOMPLIZIERTE TEXTE VERMEIDEN
- FORMULAR MIT KLAREN FEHLERMELDUNGEN



```
<html lang="de">
```


- Sprache richtig setzen - Screenreader lesen in der passenden Sprache vor



```
<label for="email">
  E-Mail-Adresse
</label>
<input id="email" type="email" />
<p class="error">
  Bitte geben Sie eine gültige E-Mail-Adresse ein.
</p>
```


ROBUSTHEIT (ROBUST)

- KOMPATIBILITÄT MIT VERSCHIEDENEN BROWSERN UND SCREEMREADERN
- MUSS AUCH MIT ZUKÜNFTIGEN TECHNOLOGIEN FUNKTIONIEREN
- SEMANTISCHES HTML UND WENN NÖTIG ARIA VERWENDEN



```
<button type="submit">
  Senden
</button>
```

- funktioniert mit Tastatur
- Screenreader erkennt Funktion



```
<button
  aria-expanded="true"
  aria-controls="menu">
  Menü öffnen
</button>
<ul id="menu">
  <li>
    <a href="#">Start</a>
  </li>
</ul>
```

- ARIA ergnzt Informationen fr Screenreader
- Wechselzustand hat das Element (offen/zu)
- Plus kontrolliert der Button (Men)

Die WCAG sind um vier Grundprinzipien aufgebaut, die unter dem Krzel **POUR** zusammengefasst werden:

1. Wahrnehmbarkeit (Perceivable)

Alt text helps screen readers read visual content



Without Alt Text



Img_1234



With Alt Text

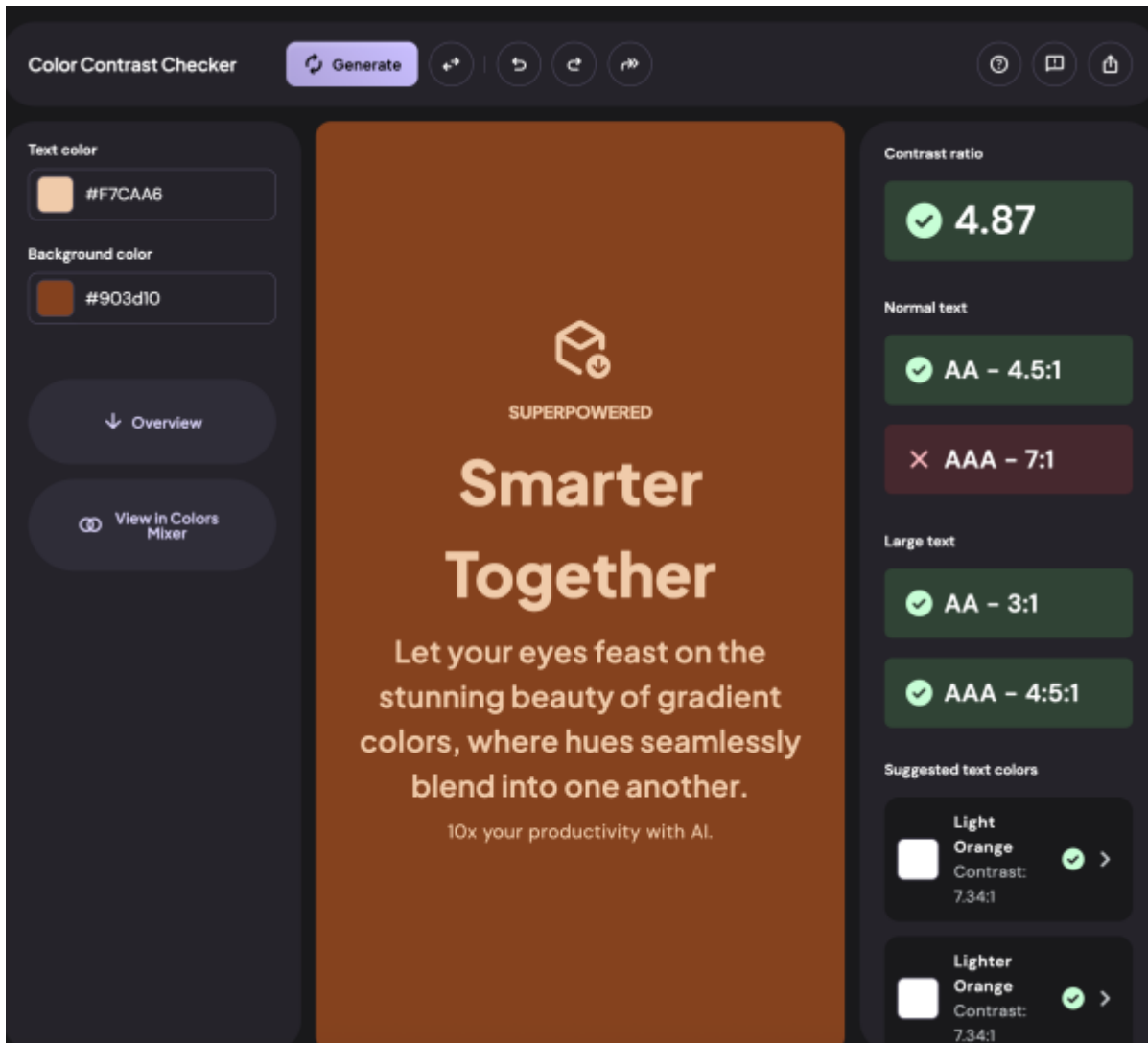


Airplane landing

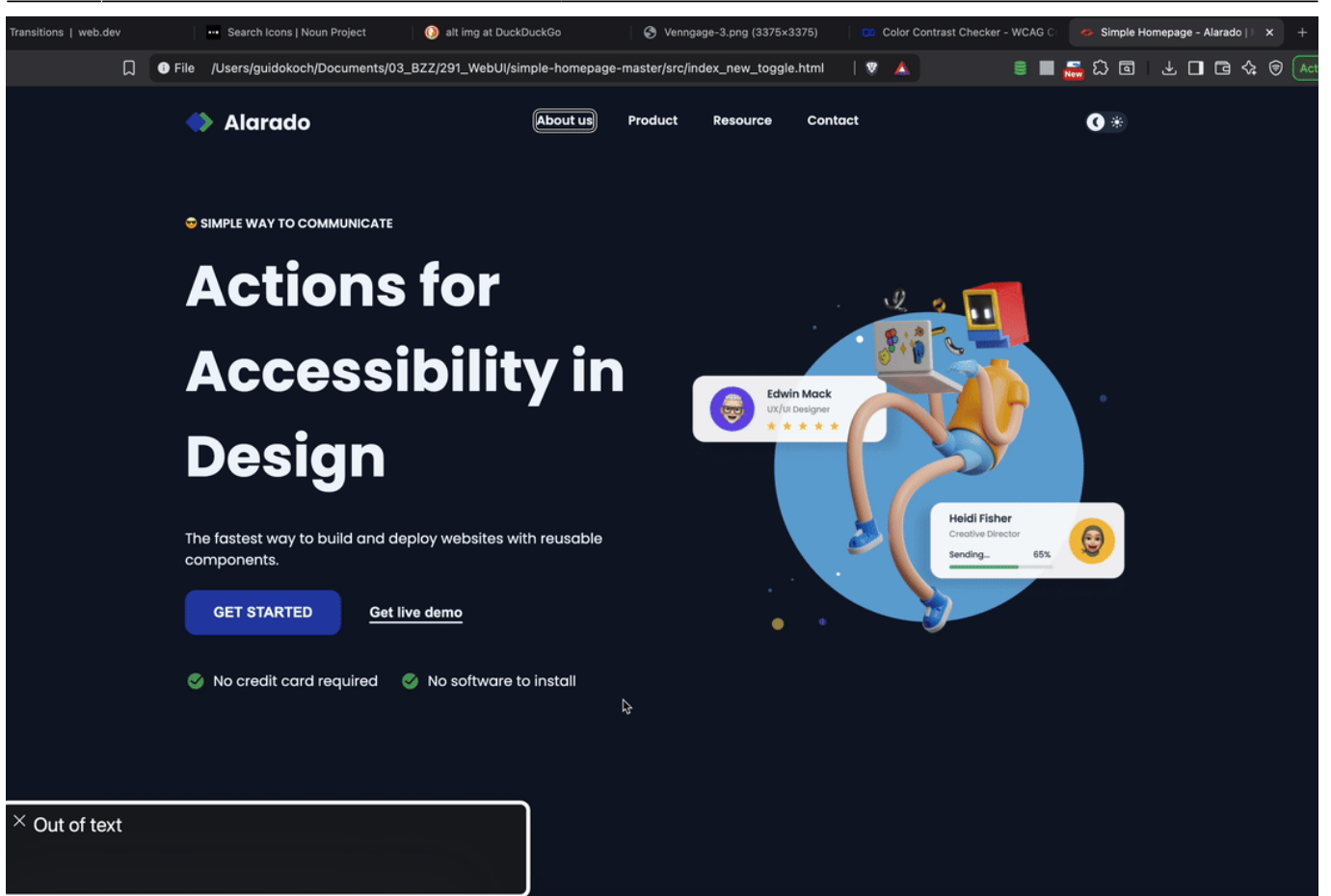
Read more: venngage.com/blog/image-alt-text

Inhalte müssen für Augen, Ohren oder Screenreader erkennbar sein.

- Bilder brauchen aussagekräftige alt-Attribute
- Videos benötigen Untertitel oder Transkripte
- Texte müssen ausreichend Kontrast zum Hintergrund haben (Mindest-Verhältnis: **4.5:1** für Fliesstext) → das Kontrastverhältnis von zwei Farben kann z.B. mittels diesem Online-Tools herausgefunden werden: colorffy.com [Text-Hintergrund-Kontrast-Checker](#)



2. Bedienbarkeit (Operable)



Die Website muss ohne Maus – also nur mit Tastatur oder Sprachsteuerung – vollständig nutzbar sein.

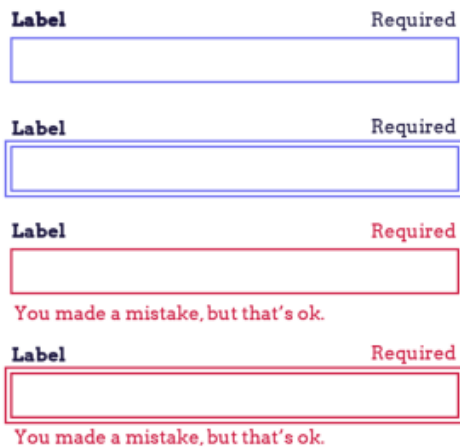
- Focus-Styles beibehalten
- Keine blinkenden Inhalte, die Anfälle auslösen könnten
- In unserem Projekt: `aria-expanded` beim Accordion und `aria-checked` beim Switch machen Zustände für Screenreader lesbar

3. Verständlichkeit (Understandable)



Sprache und Aufbau müssen logisch und vorhersehbar sein.

- Fehlermeldungen in Formularen müssen präzise erklären, was falsch ist (statt kryptischer Codes)
- Die Sprache des Dokuments muss im `<html>` deklariert sein: `<html lang=„de“>`
- Interaktionen sollen sich erwartungsgemäss verhalten



4. Robustheit (Robust)



Inhalte müssen auf verschiedenen Geräten und mit Hilfstechnologien (Screenreader, Braille-Ausgabegeräte) zuverlässig funktionieren.

- Semantisches HTML verwenden statt `<div>` für alles
- Schriftgrößen in rem definieren, damit Browser-Zoom korrekt skaliert
- ARIA-Attribute korrekt und sparsam einsetzen

Screenreader

Ein **Screenreader** ist ein Hilfsprogramm, das den Bildschirminhalt vorliest. Der Browser baut parallel zum DOM-Baum einen **Accessibility Tree** auf – eine vereinfachte Darstellung der Seite mit Rollen,

Zuständen und Namen aller Elemente. Der Screenreader liest diesen Tree aus. Unter Mac gibt es in den System-Einstellungen zur „Accessibility“ (Bedienungshilfen) die Möglichkeit VoiceOver zu aktivieren. Unter Windows heisst es Sprachausgabe.



Windows: Sprachausgabe einschalten →

<https://support.microsoft.com/de-de/windows/kapitel-1-einf%C3%BChrung-in-die-sprachausgabe-7fe8fd72-541f-4536-7658-bfc37ddaf9c6>

Mac: VoiceOver einschalten → <https://support.apple.com/de-ch/guide/voiceover/mchlp2687/mac>

Semantisches HTML als Grundlage

Die wichtigste Grundregel der Accessibility:

Verwenden Sie das richtige HTML-Element für den richtigen Zweck.

Browser und Screenreader kennen die eingebaute Semantik von HTML-Elementen. Ein `<button>` ist automatisch:

- Per Tastatur fokussierbar (Tab-Taste)
- Auslösbar mit Enter und Space
- Für Screenreader als «Schaltfläche» erkennbar

Erste ARIA-Regel: Verwenden Sie ARIA **nicht**, wenn ein natives HTML-Element denselben Zweck erfüllt. Ein `<button>` braucht kein `role=„button“`. ARIA ist nur dann nötig, wenn HTML allein nicht ausreicht.

ARIA - Accessible Rich Internet Applications

ARIA ist eine Sammlung von HTML-Attributen, die Screenreadern und Hilfstechnologien zusätzliche Bedeutung kommunizieren - Rollen, Zustände und Eigenschaften, die HTML allein nicht ausdrücken kann.

Drei ARIA-Konzepte:

Konzept	Attribut	Beispiele
Rolle	role=„...“	role=„switch“, role=„dialog“, role=„alert“
Zustand	aria-*	aria-expanded=„true“, aria-checked=„false“

Konzept	Attribut	Beispiele
Eigenschaft	aria-*	aria-label=„Menü öffnen“, aria-hidden=„true“

aria-expanded - Zustand kommunizieren

Das Attribut `aria-expanded` teilt Screenreadern mit, ob ein steuerbarer Bereich gerade auf- oder zugeklappt ist. Beim Accordion ist das essenziell.

Im HTML setzen wir `aria-expanded=„false“` als Ausgangszustand:

```
<button class="accordion-btn" aria-expanded="false">
  What is this project, and how will it help me?
</button>
<p class="panel">
  It's a small but mighty mission...
</p>
```

Was der Screenreader vorliest:

- Zugeklappt: «What is this project, Schaltfläche, zugeklappt»
- Aufgeklappt: «What is this project, Schaltfläche, aufgeklappt»

Tastaturbedienbarkeit

Nicht alle Nutzenden arbeiten mit einer Maus. Menschen mit motorischen Einschränkungen, Sehbehinderungen oder temporären Verletzungen navigieren mit der Tastatur.

Praxisbeispiel: Dark-/Light-Mode-Switch in Alarado

Das Problem

Im Alarado-Projekt wurde der `<input type="checkbox">` mit `display: none` versteckt:

```
/* Alarado CSS - bisheriger Code */
.switch input {
  display: none;
}
```

Das entfernt das Eingabefeld **komplett** aus dem Accessibility Tree. Screenreader

und Tastaturnavigation überspringen den Switch – er ist für Nutzende von Hilfstechnologien unsichtbar und nicht bedienbar.

Das HTML im Alarado-Projekt

```
<label class="switch">
  <input type="checkbox" />
  <span class="slider"></span>
  <span class="switch-icons"></span>
</label>
```

Die Lösung: Visuell verstecken, aber zugänglich halten

Statt `display: none` verwenden wir eine CSS-Technik, die das Element **optisch unsichtbar** macht, es aber im Accessibility Tree **belässt**. Diese Technik nennt man «Visually Hidden»:

```
/* Visuell versteckt, aber für Screenreader sichtbar */
.switch input {
  position: absolute;
  width: 1px;
  height: 1px;
  margin: -1px;
  padding: 0;
  border: 0;
  overflow: hidden;
  clip: rect(0, 0, 0, 0);
  white-space: nowrap;
}
```

Damit der Screenreader die **Rolle** und den **Zustand** des Switches versteht, ergänzen wir das HTML mit ARIA-Attributen:

```
<label class="switch">
  <input
    type="checkbox"
    role="switch"
    aria-checked="false"
    aria-label="Dark Mode aktivieren"
  />
  <span class="slider"></span>
  <span class="switch-icons"></span>
```

`</label>`

- `role=„switch“` – teilt mit, dass es sich um einen Ein-/Aus-Schalter handelt (nicht nur eine Checkbox)
- `aria-checked=„false/true“` – kommuniziert den aktuellen Zustand
- `aria-label=„...“` – gibt dem Switch einen lesbaren Namen (da kein sichtbarer Label-Text vorhanden ist)

Und im JavaScript aktualisieren wir `aria-checked` beim Umschalten:

```
const toggle = document.querySelector('.switch input');

toggle.addEventListener('change', () => {
  const isDark = toggle.checked;
  // Dark Mode umschalten ...
  toggle.setAttribute('aria-checked', isDark ? 'true' :
    'false');
});
```

Was der Screenreader jetzt vorliest:

- «Dark Mode aktivieren, Schalter, ausgeschaltet»
- Nach Klick: «Dark Mode aktivieren, Schalter, eingeschaltet»

A11y-Grundsätze für jedes Projekt

- Semantische HTML-Elemente verwendet (`<button>`, `<nav>`, `<main>`, ...)?
- Sprache des Dokuments deklariert: `<html lang=„de“>`?
- Alle interaktiven Elemente per Tastatur erreichbar (Tab-Navigation)?
- Focus-Styles sichtbar - `outline: none` nirgends gesetzt?
- Texte haben ausreichend Kontrast (Mindest-Verhältnis 4,5:1)?
- Schriftgrößen in `rem` definiert (skaliert korrekt bei Browser-Zoom)?
- Bilder haben ein aussagekräftiges `alt`-Attribut (oder `alt=„“` wenn dekorativ)?
- Interaktive Elemente haben ein zugängliches Label (`aria-label` oder sichtbarer Text)?
- Accordion kommuniziert Zustand mit `aria-expanded`?
- Toggle/Switch kommuniziert Zustand mit `aria-checked` und `role=„switch“`?
- Versteckte Elemente, die zugänglich sein sollen, mit `Visually Hidden - nicht display: none`?

From:

<https://wiki.bzz.ch/> - **BZZ - Modulwiki**

Permanent link:

https://wiki.bzz.ch/de/modul/m291/learningunits/lu06/theorie/c_accessibility?rev=1773658609

Last update: **2026/03/16 11:56**

