LU05c – Accessibility Basics (A11y)

Weiterführende Ressource: Dieser Block stützt sich auf den Google-Kurs 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 Goldstandard für digitale Barrierefreiheit. Sie bieten Entwickler:innen klare, prüfbare Kriterien dafür, wie eine zugängliche Website aussehen muss.

Konformitätsstufen

Stufe	Bedeutung	Einsatz
A	Mindestanforderungen	Ohne diese ist die Nutzung fast unmöglich
AA	Internationaler Standard	Pflicht für Unternehmen und öffentliche Stellen
AAA	Höchste Stufe	Oft schwer vollständig umsetzbar

Für die meisten Projekte gilt **Stufe AA** als Ziel.

POUR – Die vier Grundprinzipien

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

1. Wahrnehmbarkeit (Perceivable)

Inhalte müssen für Augen, Ohren oder Screenreader erkennbar sein.

- Bilder brauchen aussagekräftige `alt`-Attribute
- Videos benötigen Untertitel
- Texte müssen ausreichend Kontrast zum Hintergrund haben (Mindest-Verhältnis: **4,5:1** für Fliesstext)

2. Bedienbarkeit (Operable)

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

- Focus-Styles beibehalten (nie `outline: none` setzen)
- 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.

Screenreader	Plattform	Kosten
VoiceOver	macOS / iOS	integriert (kostenlos)
NVDA	Windows	kostenlos (Open Source)
JAWS	Windows	kostenpflichtig
TalkBack	Android	integriert (kostenlos)

Live-Demo: Alarado Landingpage mit VoiceOver

In der Lektion demonstrieren wir, was ein Screenreader auf der Alarado Landingpage vorliest. Die Beobachtungen:

- **Navigation:** Die Links in der Nav-Bar werden korrekt erkannt und vorgelesen, z. B. «Features, Link».
- **Buttons:** Die Haupt-Buttons im Hero-Bereich sind per Tab erreichbar und werden vorgelesen.
- **Dark-/Light-Mode-Switch: □ Nicht erreichbar!** Der Switch erscheint für den Screenreader nicht – er kann ihn weder finden noch bedienen.

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

Ein

`mit onclick hat **keine** dieser Eigenschaften von Haus aus. <code html> <!— □ Nicht barrierefrei ohne viel Zusatzarbeit —>`

Frage anzeigen

<!– □ Semantisch korrekt – sofort barrierefrei –> <button onclick=„openPanel()“>Frage anzeigen</button> </code>

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“ | | **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: <code html> <button class=„accordion-btn“ aria-expanded=„false“> What is this project, and how will it help me? </button> It's a small but mighty mission… </code> **Was der Screenreader vorliest:** * Zugeklappt: «What is this project, Schaltfläche, zugeklappt» * Aufgeklappt: «What is this project, Schaltfläche, aufgeklappt» Im JavaScript aktualisieren wir aria-expanded beim Klick: <code javascript> buttons.forEach((button) ⇒ { button.addEventListener('click', (e) ⇒ { const btn = e.currentTarget; const panelElement = btn.nextElementSibling; const panelIsOpen = panelElement.classList.contains('open'); *Alle Panels schliessen + aria-expanded zurücksetzen buttons.forEach((andererBtn) ⇒ { andererBtn.nextElementSibling.classList.remove('open'); andererBtn.setAttribute('aria-expanded', 'false');* ← State kommunizieren }); *Dieses Panel öffnen if (!panelIsOpen) { panelElement.classList.add('open'); btn.setAttribute('aria-expanded', 'true');* ← State kommunizieren } }); }); </code> ===== Tastaturbedienbarkeit ===== Nicht alle Nutzenden arbeiten mit einer Maus. Menschen mit motorischen Einschränkungen, Sehbehinderungen oder temporären Verletzungen navigieren mit der Tastatur. ^ Taste ^ Aktion ^ | Tab | Zum nächsten fokussierbaren Element springen | | Shift + Tab | Zum vorherigen Element springen | | Enter / Space | Button oder Link aktivieren | | Esc | Dialog oder Dropdown schliessen | | Pfeiltasten | Navigation innerhalb von Komponenten | ==== Focus-Styles nicht entfernen! ==== Der sichtbare Fokusrahmen (outline) ist für Tastaturnutzende das einzige visuelle Feedback, wo sie sich auf der Seite befinden. Ihn zu entfernen ist einer der häufigsten Barrierefreiheitsfehler überhaupt: <code css> button:focus { outline: none; } button:focus-visible { outline: 3px solid var(–violet-600); outline-offset: 3px; border-radius: 4px; } </code>

:focus-visible statt :focus: Die Pseudo-Klasse `:focus-visible` zeigt den Fokusrahmen nur bei Tastaturnavigation – nicht beim Mausklick. So bleibt das Design sauber, ohne Tastaturnutzende zu benachteiligen.

===== Praxisbeispiel: Dark-/Light-Mode-Switch in Alarado ===== ==== Das Problem ==== Im Alarado-Projekt wurde der <input type=„checkbox“> mit display: none versteckt: <code css> .switch input { display: none; } </code> 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 ==== <code html> <label class=„switch“> <input type=„checkbox“ /> </label> </code> ==== 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»: <code css> .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; } </code> Damit der Screenreader die **Rolle** und den **Zustand** des Switches versteht, ergänzen wir das HTML mit ARIA-Attributen: <code html> <label class=„switch“> <input type=„checkbox“ role=„switch“ aria-checked=„false“ aria-label=„Dark Mode aktivieren“ /> </label> </code> * 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: <code javascript> const toggle = document.querySelector('.switch input'); toggle.addEventListener('change', () ⇒ { const isDark = toggle.checked; *Dark Mode umschalten … toggle.setAttribute('aria-checked', isDark ? 'true' : 'false'); }); </code> ***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?*