SOP 32

Modale Dialoge, Pop-ups und Hamburger-Menüs zugänglich machen

Projektleitung | Geschäftsführung | IT | Recht

So verhinderst du Tastaturfallen und Desorientierung

Weil jede Komponente, die sich über den restlichen Seiteninhalt legt, eine massive Barriere darstellt, wenn sie nicht perfekt umgesetzt ist. Die häufigsten Fehler sind:

- Keine Tastaturfalle: Nutzer:innen können mit der Tab-Taste „hinter“ das Pop-up zu den Elementen auf der Seite springen, die sie gar nicht sehen können.
- Keine Ankündigung: Screenreader wissen nicht, dass ein Dialogfenster erschienen ist und lesen einfach auf der Seite dahinter weiter.
- Kein Entkommen: Der Dialog kann nicht mit der Tastatur (z. B. über die Escape-Taste oder einen Button) geschlossen werden.

Diese Fehler führen dazu, dass Tastatur-Nutzer:innen gefangen sind und die Seite verlassen müssen. Das ist nicht nur frustrierend, sondern ein klares K.o.-Kriterium für die Barrierefreiheit.

Wann musst du das machen?

- Immer, wenn du einen modalen Dialog verwendest (z. B. für eine „In den Warenkorb“-Bestätigung, eine Bildergalerie, ein Login-Formular).
- Immer, wenn du ein Pop-up nutzt (z. B. für eine Newsletter-Anmeldung).
- Immer, wenn du ein Hamburger-Menü oder eine andere Art von Off-Canvas-Navigation für mobile Ansichten oder Desktops einsetzt.

Welches Gesetz verlangt das?

- European Accessibility Act (EAA) / Barrierefreiheitsstärkungsgesetz (BFSG): Alle Funktionen und Inhalte müssen zugänglich und bedienbar sein.
- WCAG 2.1 , hier sind mehrere, sehr spezifische Kriterien entscheidend:

Der Prozess im Detail

1 🟧 **Phase 1: Die richtige Struktur – Sag dem Browser, was es ist**

Bevor du an die Interaktion denkst, muss das technische Fundament stimmen.

- Die Rolle definieren: Das Container-Element des Dialogs muss das Attribut role=“dialog“ haben.
- Den Modus festlegen: Es muss zudem aria-modal=“true“ gesetzt bekommen. Dieses Attribut ist entscheidend: Es signalisiert Screenreadern, den Inhalt hinter dem Dialog zu ignorieren.
- Einen Namen geben: Der Dialog braucht eine Beschriftung. Am besten geht das mit aria-labelledby=“dialog-titel-id“, wobei das Attribut auf die ID der sichtbaren Überschrift (<h1>, <h2> etc.) im Dialog verweist.

2 🟨 Phase 2: Das Öffnen – Der Fokus muss mitkommen

Wenn der Dialog erscheint, muss der Tastaturfokus aktiv in den Dialog bewegt werden.

- Der/die Nutzer:in klickt auf einen Button, um den Dialog zu öffnen.
- Per JavaScript wird der Dialog sichtbar gemacht.
- Direkt danach muss der Fokus per JavaScript (element.focus()) auf das erste fokussierbare Element innerhalb des Dialogs gesetzt werden. Das ist üblicherweise der Schließen-Button oder das erste Formularfeld.

3 🟩 Phase 3: Die Nutzung – Die Tastaturfalle (Keyboard Trap)

Das ist der technisch anspruchsvollste, aber wichtigste Teil. Während der Dialog offen ist, muss das Skript die Tab-Navigation „einfangen“.

- Vorwärts-Tab: Wenn der Fokus auf dem letzten interaktiven Element im Dialog ist (z. B. dem „Absenden“-Button) und der/die Nutzer:in erneut Tab drückt, muss der Fokus auf das erste interaktive Element (z. B. den Schließen-Button) zurückspringen.
- Rückwärts-Tab: Wenn der Fokus auf dem ersten Element ist und der/die Nutzer:in Shift + Tab drückt, muss der Fokus auf das letzte Element springen.

4 🟥 Phase 4: Das Schließen – Biete immer einen Ausweg an

Ein Dialog muss auf mindestens zwei Wegen geschlossen werden können:

- Ein sichtbarer Schließen-Button: Dies muss ein echtes <button>-Element sein. Ein einfaches „X“-Icon braucht einen zugänglichen Namen, z. B. <button aria-label=“Dialog schließen“>X</button>.
- Die Escape-Taste: Nutzer:innen erwarten, dass sie mit der Escape-Taste jeden Dialog verlassen können. Dies muss per JavaScript implementiert werden.

5 🟦 Phase 5: Das Schließen – Den Fokus zurückgeben

Wenn der Dialog geschlossen wird, muss der Tastaturfokus dorthin zurückkehren, von wo er gekommen ist.

- Speichere den Auslöser: Bevor der Fokus beim Öffnen in den Dialog gesetzt wird, muss dein Skript sich merken, welches Element den Dialog ausgelöst hat (z. B. der „Warenkorb öffnen“-Button).
- Setze den Fokus zurück: Sobald der Dialog geschlossen wird, muss der Fokus per JavaScript (element.focus()) wieder genau auf dieses ursprüngliche Auslöser-Element gesetzt werden. Das verhindert, dass der/die Nutzer:in die Orientierung verliert.

Ergebnis

Am Ende dieser SOP hast du:

- Robuste und vorhersagbare Overlay-Komponenten, die niemanden aussperren oder frustrieren.
- Die gefürchtete „Tastaturfalle“ sicher eliminiert und damit ein kritisches WCAG-Kriterium erfüllt.
- Eine professionelle Nutzererfahrung geschaffen, die Vertrauen aufbaut, weil sie Kontrolle ermöglicht.
- Ein wiederverwendbares Muster, das du für alle zukünftigen Pop-ups und Dialoge nutzen kannst.

Um die in dieser SOP festgelegten Anforderungen an Fokus-Management, Tastaturfalle, Schließ-Mechanismen und ARIA-Attribute für modale Dialoge und Overlays sofort und zuverlässig umzusetzen, ist es sinnvoll ein bereits erprobten Muster zu nutzen.

Das folgende Beispiel eines „Accessible Modal Dialog“ des W3C (World Wide Web Consortium) ist ein Industriestandard und erfüllt alle relevanten Kriterien der WCAG und des BFSG. Es dient als Direktive für die technische Implementierung .

Implementierungs-Referenz: Barrierefreies Modal-Muster

Deine Aufgabe:

- Analysiere das verlinkte Muster, um die technische Umsetzung der SOP-Phasen 1 bis 5 zu verstehen.
- Nutze diesen Code als Basis für neue Overlay-Komponenten oder zur Überprüfung und Anpassung bestehender Implementierungen.
- Achte bei der Integration auf die Spezifika deines Projekts (Design-Anpassungen, Event-Handler für Inhalt).

Direkter Link zur Muster-Implementierung (HTML, CSS, JavaScript):

- W3C ARIA Authoring Practices Guide (APG) – Dialog (Modal) Example:
- https://www.w3.org/WAI/ARIA/apg/patterns/dialog-modal/examples/dialog/

Hinweis für die Entwicklung: Bitte beachtee, dass die Integration dieses Musters in eure spezifische Codebasis und euer Design erfolgen muss. Dies ist keine fertige 1:1-Lösung , sondern ein best-practice-Beispiel , das die korrekte Struktur und das Verhalten demonstriert. Passt das Styling an eure UI/UX-Vorgaben an, aber bewahre die zugrunde liegende JavaScript-Logik und die ARIA-Attribute.

Fragen & Antworten

Beim Öffnen des Dialogs sollte der Fokius auf das erste fokussierbare Element gelegt werden. Was ist, wenn mein Dialog eine lange Textpassage enthält, die zuerst gelesen werden sollte?

Das ist eine wichtige Nuance. Während die Empfehlung, den Fokus auf das erste fokussierbare Element (oft ein Button oder Feld) zu setzen, für interaktive Dialoge (wie Login oder Formulare) sinnvoll ist, kann es bei reinen Informationsdialogen (z.B. eine lange AGB-Einblendung oder eine „Über uns“-Seite in einem Modal) kontraproduktiv sein. In solchen Fällen ist es besser, den Fokus direkt auf die Überschrift des Dialogs (aria-labelledby sollte darauf verweisen) zu legen. So beginnt der Screenreader mit dem Kontext und der Nutzer kann dann den Textinhalt lesen, bevor er zu interaktiven Elementen springt. Wichtig ist, dass der Fokus irgendwo sinnvoll im Dialog landet , nicht außerhalb.

Wie verhalte ich mich bei Pop-ups, die sich automatisch öffnen (z.B. für Newsletter-Anmeldungen) und die Seite überlagern? Muss ich hier auch den Fokus in das Pop-up verschieben?

Ja, absolut! Auch bei automatisch geöffneten Pop-ups gelten dieselben Regeln. Sobald das Pop-up erscheint, muss der Fokus sofort in das Pop-up verschoben werden. Das signalisiert dem Screenreader und Tastaturnutzer, dass sich der Kontext geändert hat und sie jetzt mit dem neuen Inhalt interagieren müssen. Stell dir vor, du navigierst mit einem Screenreader und plötzlich erscheint ein Pop-up, ohne dass du es bemerkst – du würdest einfach „dahinter“ weiterlesen und könntest das Pop-up nicht schließen oder ausfüllen. Das wäre eine massive Barriere.

Manche Overlays sind keine reinen Dialoge, sondern eher temporäre Informationen wie Toasts oder Snackbars. Brauchen die auch eine Tastaturfalle oder Fokus-Management?

Für „Toast-Messages“ oder „Snackbars“ (kurze, flüchtige Erfolgs- oder Fehlermeldungen, die von selbst verschwinden) gelten oft andere Regeln , da sie nichtmodal sind und keine Interaktion erfordern.

- Kein Fokus-Management: Der Fokus sollte nicht in einen Toast verschoben werden, da er den aktuellen Nutzer-Workflow unterbrechen würde.
- role="status" oder aria-live="polite" : Diese Elemente sollten role="status" oder eine aria-live="polite" Region sein, damit Screenreader sie automatisch und ohne Unterbrechung des Nutzungsflusses vorlesen, wenn sie erscheinen.
- Keine Tastaturfalle: Da keine Interaktion erwartet wird, ist keine Tastaturfalle nötig.

Modale Dialoge hingegen erfordern eine Nutzerinteraktion und müssen daher den Fokus bündeln.

Meine Hamburger-Menüs auf dem Handy sind keine Overlays, sondern schieben den Inhalt nur weg (Off-Canvas). Gelten die Regeln der SOP trotzdem?

Ja, unbedingt! Auch wenn sie den Inhalt wegschieben und nicht direkt überlagern, gelten Off-Canvas-Menüs (wie sie oft bei Hamburger-Menüs verwendet werden) als „modale“ oder kontextverändernde Komponenten aus Sicht der Barrierefreiheit. Der Grund: Sie blockieren den Zugriff auf den darunterliegenden Inhalt und ändern den sichtbaren Kontext der Seite. Daher müssen alle in der SOP beschriebenen Punkte beachtet werden:

- Fokusverschiebung: Der Fokus muss beim Öffnen ins Menü (z.B. auf den ersten Menüpunkt oder den Schließen-Button) verschoben werden.
- Tastaturfalle: Der Fokus muss innerhalb des geöffneten Menüs gefangen sein.
- Schließen: Das Menü muss per Escape-Taste und einem sichtbaren Button schließbar sein.
- Fokus-Rückgabe: Der Fokus muss nach dem Schließen exakt auf das Hamburger-Icon zurückkehren. Das stellt sicher, dass mobile Nutzer, die mit Tastatur oder Screenreader navigieren, nicht orientierungslos sind.

Wenn ich einen Screenreader-Test mache, worauf sollte ich besonders achten, wenn ein Dialog oder Pop-up geöffnet ist?

Beim Screenreader-Test solltest du speziell auf folgende Punkte achten, wenn ein Overlay aktiv ist:

- Ankündigung des Dialogs: Wird der Dialog sofort als „Dialog“ oder „Fenster“ angekündigt, und wird dessen Titel vorgelesen? (role="dialog" und aria-labelledby sind hier entscheidend).
- Fokusposition: Wo landet der Screenreader-Fokus, nachdem der Dialog geöffnet wurde? Ist es ein sinnvolles Element (z.B. Schließen-Button oder erste Eingabe)?
- Hintergrund-Ignorierung: Kannst du mit den Screenreader-Navigationsbefehlen (z.B. Tab oder Pfeiltasten) auf Elemente hinter dem Dialog zugreifen? Wenn ja, ist aria-modal="true" wahrscheinlich nicht korrekt gesetzt oder funktioniert nicht.
- Tastaturnavigation innerhalb des Dialogs: Verhält sich die Tab-Navigation innerhalb des Dialogs erwartungsgemäß? Springt sie vom letzten Element zum ersten zurück?
- Schließen-Funktionalität: Kannst du den Dialog mit der Escape-Taste und dem Schließen-Button verlassen?
- Fokus-Rückgabe: Landet der Fokus nach dem Schließen des Dialogs wieder auf dem Element, das ihn geöffnet hat?

Wenn diese Punkte nicht passen, weißt du, dass noch Nachbesserungsbedarf besteht, um die Barrierefreiheit zu gewährleisten.