SOP 36

Eingebettete Inhalte absichern

Projektleitung | Geschäftsführung | IT | Recht

So schließt du die Hintertüren in deiner barrierefreien Website

Weil jeder eingebettete Inhalt von Drittanbietern – sei es ein YouTube-Video, eine Google-Maps-Karte oder ein Social-Media-Feed – wie ein „Loch“ in deiner Website ist. Du hast keine Kontrolle über den Code innerhalb dieses externen Rahmens (genannt iframe), bist aber nach dem Gesetz trotzdem für die Barrierefreiheit der gesamten Nutzererfahrung auf deiner Seite verantwortlich.

Diese Einbettungen sind oft eine Quelle schwerwiegender Barrieren: Sie haben keinen Titel, sind nicht per Tastatur bedienbar oder fangen den Tastaturfokus der Nutzer:innen ein, sodass diese nicht mehr wegnavigieren können.

Wann musst du das machen?

- Immer, wenn du Inhalte von einem anderen Dienst auf deiner Seite einbettest.
- Konkret bei: YouTube-/Vimeo-Videos, Google-Maps-Karten, Social-Media-Feeds (Twitter, Instagram etc.), Buchungs-Widgets von Drittanbietern und allen anderen Inhalten, die per <iframe> geladen werden.

Welches Gesetz verlangt das?

- European Accessibility Act (EAA) / Barrierefreiheitsstärkungsgesetz (BFSG): Als Anbieter deiner Website bist du für das Gesamterlebnis verantwortlich, auch wenn Teile davon von Dritten stammen.
- WCAG 2.1 , mehrere Kriterien sind hier direkt betroffen:
- 4.1.2 Name, Role, Value: Das ist das Wichtigste. Ein iframe muss einen zugänglichen Namen (ein title-Attribut) haben, damit Screenreader-Nutzer:innen wissen, was sich darin verbirgt, bevor sie damit interagieren.
- 2.1.1 Keyboard: Nutzer:innen müssen mit der Tastatur in den iframe hinein- und wieder hinausnavigieren können.
- 2.1.2 No Keyboard Trap: Der eingebettete Inhalt selbst darf keine Tastaturfalle darstellen.

Der Prozess im Detail

1 🟧 **Phase 1: Der iframe-Titel – Die absolute Pflicht**

Dies ist nicht verhandelbar. Jeder iframe auf deiner Website muss ein aussagekräftiges title-Attribut haben.

- Warum? Der Titel ist das Einzige, was ein Screenreader vorliest, bevor der/die Nutzer:in entscheidet, mit dem Inhalt zu interagieren. Ohne Titel hört der/die Nutzer:in nur „Frame“ und weiß nicht, ob sich dahinter eine gefährliche Tastaturfalle oder ein nützliches Video verbirgt.
- Gutes Beispiel: <iframe src="…“ title="Interaktive Karte unseres Standorts in Berlin"></iframe>
- Gutes Beispiel: <iframe src="…“ title="YouTube-Videoplayer: Unsere Produktvorstellung"></iframe>
- Schlechte Beispiel: <iframe src="…“ title="iframe"> (nicht aussagekräftig) oder gar kein Titel.

2 🟨 Phase 2: Die Tastatur-Prüfung – Rein und wieder raus

Führe vor jeder Veröffentlichung diesen einfachen 5-Sekunden-Test durch:

- Navigiere mit der Tab-Taste bis zum Element vor dem eingebetteten Inhalt.
- Drücke erneut Tab. Der Fokus sollte sich nun innerhalb des iframes befinden (z. B. auf dem „Play“-Button des YouTube-Players).
- Drücke Tab einige Male, um innerhalb des iframes zu navigieren.
- Drücke so lange Tab, bis du auf dem letzten Element im iframe bist.
- Drücke Tab ein weiteres Mal. Der Fokus muss den iframe jetzt verlassen und zum nächsten Element auf deiner Hauptseite springen. Wenn du stattdessen wieder am Anfang des iframes landest, hast du eine Tastaturfalle.

3 🟩 Phase 3: Das Innenleben bewerten – Wie zugänglich ist der Drittanbieter?

Du kannst den Code im iframe nicht ändern, aber du kannst das Risiko bewerten.

- Visuelle Prüfung: Haben die Bedienelemente in der Karte oder im Player ausreichende Kontraste? Sind sie groß genug?
- Tastatur-Prüfung (innen): Kannst du alle wichtigen Funktionen im iframe nur mit der Tastatur bedienen? Kannst du in die Karte zoomen? Kannst du im Video vor- und zurückspulen?
- Problemfall: Wenn der Inhalt (z. B. ein Buchungs-Widget) per Tastatur unbenutzbar ist, hast du ein Problem, das du mit Plan B lösen musst.

4 🟥 Phase 4: Plan B – Eine barrierefreie Alternative anbieten

Wenn der eingebettete Inhalt essenziell, aber nicht barrierefrei ist, bist du verpflichtet, eine Alternative bereitzustellen.

- Für Karten: Gib die Adresse als reinen Text an und füge einen normalen Link zu Google Maps hinzu.
- Beispiel: „Unsere Adresse: Musterstraße 1, 12345 Berlin. [Route in Google Maps öffnen]“
- Für Buchungs-Widgets: Biete eine Telefonnummer und eine E-Mail-Adresse als alternativen Weg zur Buchung an.
- Für Social-Media-Feeds: Verlinke einfach direkt auf dein Social-Media-Profil, anstatt einen oft unzugänglichen Feed einzubetten.
- Beispiel: „Unsere neuesten Beiträge finden Sie direkt auf unserem [Twitter-Profil].“

5 🟦 Phase 5: Performance optimieren mit loading=%22lazy%22 (Best Practice)

iframes verlangsamen deine Seite, da der Browser sofort den gesamten externen Inhalt laden muss. Nutze das loading=“lazy“-Attribut, um das Laden zu verzögern, bis der/die Nutzer:in in die Nähe des iframes scrollt. Das verbessert die Ladezeit für alle.

- Beispiel: <iframe src=“…“ title=“Interaktive Karte…“ loading=“lazy“></iframe>

Ergebnis

Am Ende dieser SOP hast du:

- Sichergestellt, dass eingebettete Inhalte deine Barrierefreiheit nicht untergraben.
- Nutzer:innen transparent darüber informiert, was sie in einem iframe erwartet.
- Verhindert, dass Tastatur-Nutzer:innen auf deiner Seite gefangen werden.
- Einen Notfallplan für unzugängliche, aber notwendige Widgets von Drittanbietern.
- Deine Website-Performance durch den Einsatz von Lazy Loading verbessert.

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 .

Betreff: Checkliste für das Einbetten externer Inhalte (YouTube, Karten etc.)

Hallo Team,

bevor ihr einen externen Inhalt per iframe auf der Website einbettet, prüft bitte diese drei Punkte:

- [ ] Titel vergeben? Der iframe-Code muss ein aussagekräftiges title-Attribut enthalten (z. B. title=“Anfahrtskarte zu unserem Büro“).
- [ ] Tastatur-Test bestanden? Man muss mit der Tab-Taste in den iframe hinein- und vor allem auch wieder hinausnavigieren können.
- [ ] Alternative vorhanden? Wenn der Inhalt selbst nicht gut bedienbar ist (z. B. eine komplexe Anwendung), muss direkt daneben eine textliche Alternative angeboten werden (z. B. Adresse und Telefonnummer).

Nur wenn alle drei Punkte erfüllt sind, darf der Inhalt online gehen.

Danke!

Viele Grüße [Dein Name]

Fragen & Antworten

Warum ist es so wichtig, HTML5-Elemente (</p> <header>, <nav>, <main> usw.) anstelle von s mit ARIA-Rollen zu verwenden, auch wenn das Ergebnis für Screenreader dasselbe ist?

Das ist die Frage nach der progressiven Verbesserung (Progressive Enhancement), du! HTML5-Elemente sind nativ semantisch . Das bedeutet, Browser und assistive Technologien (wie Screenreader) verstehen ihre Bedeutung „out of the box“, ohne dass zusätzliche Attribute interpretiert werden müssen. Das macht den Code robuster, zukunftssicherer und oft auch einfacher zu warten . Wenn du s mit ARIA-Rollen verwendest, verlässt du dich darauf, dass ARIA korrekt interpretiert wird. Sollte ARIA aus irgendeinem Grund nicht geladen oder unterstützt werden, verliert dein seine Bedeutung, während ein <nav> -Element immer eine Navigation bleibt. Es ist die „natürlichere“ und damit die stabilere Lösung.

Ich habe ein CMS, das viele </p> s generiert und mir wenig Kontrolle über die HTML-Struktur gibt. Wie kann ich hier am besten vorgehen, um die SOP einzuhalten?

Das ist ein häufiges Problem, du. In solchen Fällen ist es entscheidend, die Grenzen deines CMS zu verstehen und clevere Workarounds zu finden:

- Nutze Themes/Plugins, die semantisches HTML verwenden: Prüfe, ob es Themes oder Plugins gibt, die von Haus aus HTML5-Elemente korrekt einsetzen.
- Greife mit JavaScript ein (als letzte Instanz): Wenn das CMS nur s ausspuckt, kannst du nach dem Laden der Seite mit JavaScript diese s erkennen und ihnen die entsprechenden role -Attribute hinzufügen. Dies ist allerdings ein Notnagel und sollte nur eingesetzt werden, wenn du die HTML-Ausgabe des CMS absolut nicht direkt beeinflussen kannst, da es eine clientseitige Abhängigkeit schafft.
- Nutze das CMS bewusst: Auch wenn das Grundgerüst aus s besteht, kannst du oft innerhalb von Inhaltsbereichen (<section> , <article>) die Struktur mit HTML5-Elementen verbessern.
- Spreche mit dem Anbieter: Leite die Anforderungen an deinen CMS-Anbieter weiter und fordere Verbesserungen bei der semantischen Ausgabe.

Ich habe verstanden, dass es nur ein <main>-Element pro Seite geben darf. Aber was ist, wenn ich mehrere Hauptinhaltsbereiche habe, die logisch eigenständig sind (z.B. bei einer Landingpage mit mehreren großen Abschnitten)?

Das ist ein klassisches Missverständnis der <main> -Rolle, du. Das <main> -Element sollte den einzigartigen Inhalt der aktuellen Seite umfassen. Wenn du auf einer Seite mehrere große, thematisch eigenständige Abschnitte hast, gruppierst du diese innerhalb des <main> -Elements mit <section> -Elementen. Jede <section> sollte dann eine eigene Überschrift (<h2> oder tiefer) haben, um die Struktur klar zu gliedern. Das <main> ist der Container für all diese Hauptinhalte, auch wenn sie aus mehreren logischen „Teilen“ bestehen. Es hilft, den Screenreader-Nutzern einen einzigen zentralen Ankerpunkt für den Kern der Seite zu bieten.

Meine Seite hat ein Suchfeld im Header und ein weiteres, detaillierteres Suchformular auf einer separaten Suchseite. Wie benenne ich diese role=%22search%22-Elemente korrekt, wenn ich zwei habe?

Ähnlich wie bei multiplen <nav> -Elementen musst du auch hier mit aria-label arbeiten, du. Auch wenn die role="search" -Attribute an sich schon von Screenreadern erkannt werden, gibst du ihnen für die Unterscheidung eindeutige Namen:

- ... (für das Suchfeld im Header)
- ... (für das detailliertere Formular auf der Suchseite) So weiß der Nutzer genau, welches Suchfeld er gerade bedient und wofür es gedacht ist, wenn er sich durch die Landmarks navigiert.

Wir haben dynamische Inhalte, die sich auf der Seite ändern (z.B. Ajax-geladene Inhalte oder Tabs, die neue Inhalte anzeigen). Bleiben die Landmarks und die Struktur bei solchen Änderungen erhalten oder muss ich etwas Besonderes beachten?

Das ist ein kritischer Punkt bei Single Page Applications (SPAs) oder Seiten mit viel dynamischem Laden, du!

- Stabile Landmarks: Die Haupt-Landmarks (<header> , <nav> , <main> , <footer>) sollten idealerweise statisch und immer präsent bleiben, auch wenn sich der Inhalt im <main> -Bereich ändert.
- Dynamische Inhalte im <main> : Wenn du neue Inhalte in den <main> -Bereich lädst (z.B. beim Wechsel eines Tabs), solltest du sicherstellen, dass die Überschriftenhierarchie innerhalb des <main> neu gesetzt oder aktualisiert wird und die neuen Inhalte semantisch korrekt (z.B. in <section> -Elementen mit Überschriften) strukturiert sind.
- Fokus-Management: Nach dem Laden neuer Inhalte ist es oft sinnvoll, den Fokus programmatisch auf die neue Hauptüberschrift des geladenen Inhalts zu setzen. Das signalisiert dem Screenreader-Nutzer, dass sich der Inhalt geändert hat und wo er mit dem Lesen beginnen soll.
- ARIA Live Regions: Für Statusmeldungen bei dynamischem Laden (z.B. „Inhalte werden geladen…“) solltest du aria-live="polite" oder role="status" nutzen, wie in deiner anderen SOP erwähnt.

Das Zusammenspiel von stabiler Grundstruktur und dynamischer, aber semantisch korrekter Befüllung ist hier der Schlüssel.