SOP 42

Testing Tools

Projektleitung | Geschäftsführung | IT | Recht

– So findest du die „Low-Hanging Fruits“ der Barrierefreiheit in Minuten, nicht Stunden

Weil deine Zeit kostbar ist und manuelle Tests aufwendig sind. Automatisierte Tools sind deine erste Verteidigungslinie: Sie scannen eine Webseite in Sekunden und decken einen erheblichen Teil der rein technischen Barrierefreiheitsfehler auf. Sie sind wie ein unbestechlicher digitaler Kollege, der dich auf fehlende Alternativtexte, unzureichende Kontraste oder falsch verknüpfte Formularfelder hinweist, bevor du tief in die manuelle Prüfung einsteigst. Durch die Integration in den Entwicklungsprozess sparst du Zeit und Geld, da Fehler frühzeitig und kostengünstig behoben werden können. Aber Achtung: Sie sind nur ein Teil der Lösung und ersetzen niemals die manuelle Prüfung.

Das Ziel dieser SOP: Am Ende hast du einen klaren und effizienten Workflow, um die führenden automatisierten Test-Tools (axe, WAVE, Lighthouse) gewinnbringend in deinen Entwicklungs- und Qualitätssicherungsprozess zu integrieren. Du kennst die Stärken und Schwächen jedes Tools und weißt, wie du ihre Ergebnisse interpretieren und in konkrete Arbeitsaufträge umwandeln kannst.

Wann musst du das machen?

- Während der Entwicklung: Direkt auf der lokalen Maschine eines Entwicklers, bevor der Code überhaupt eingecheckt wird.
- Als Teil der Continuous Integration (CI/CD): Um automatisch zu verhindern, dass neuer Code mit kritischen Barrierefreiheitsfehlern live geht.
- Für schnelle Stichproben: Um den Zustand einer Live-Seite oder einer neuen Inhaltsseite schnell zu bewerten.
- Als erster Schritt in jedem umfassenden Barrierefreiheits-Audit.

Welches Gesetz verlangt das?

EAA / BFSG / WCAG 2.1: Die Tools selbst sind nicht gesetzlich vorgeschrieben, aber sie sind das effizienteste Mittel, um die Einhaltung vieler technischer WCAG-Kriterien zu überprüfen, die durch die Gesetze gefordert werden. Sie prüfen z.B. Kriterien wie 1.1.1 (Alternativtexte), 1.4.3 (Kontraste) oder 4.1.2 (Name, Rolle, Wert).

Der Prozess im Detail

1 Die richtigen Werkzeuge für den Job auswählen

Bevor du loslegst, brauchst du ein Set an verlässlichen Tools, die dir unterschiedliche Perspektiven auf mögliche Barrieren liefern. Kein einzelnes Tool deckt alles ab – doch gemeinsam bieten sie einen starken Einstieg in die automatisierte Prüfung. Hier erfährst du, welches Tool wofür besonders geeignet ist:

🔧 axe DevTools ist ideal für Entwickler:innen, die tief in den Code eintauchen. Es liefert präzise, technische Analysen direkt im Browser und zeigt exakt, an welcher Stelle im Code der Fehler auftritt. Dank der geringen Rate an Fehlalarmen (Falsch-Positiven) kannst du dich auf die Ergebnisse verlassen. axe eignet sich besonders gut für die schnelle, technische Überprüfung während der Entwicklungsarbeit.

👀 WAVE punktet vor allem durch seine visuelle Darstellung. Es legt Icons und Farbcodes direkt über die Website und ist damit perfekt, um auf einen Blick strukturelle Fehler, problematische Kontraste oder fehlende Alternativtexte zu erkennen. Besonders nützlich ist WAVE für Content-Verantwortliche, Designer:innen oder Projektleiter:innen, die kein technisches Vorwissen mitbringen, aber trotzdem erste Barrieren identifizieren möchten.

📊 Lighthouse , direkt im Chrome-Browser integriert, ist ein schneller Gesundheitscheck für jede Seite. Es generiert einen leicht verständlichen Accessibility-Score (0–100) und eignet sich ideal für Management-Reportings oder zur ersten Orientierung. Zwar ist Lighthouse weniger tiefgehend als axe oder WAVE, dafür aber jederzeit und ohne zusätzliche Installation verfügbar.

2 Dein Test-Workflow – Vom Code zum Klick

Jetzt geht's an die Praxis. Mit dem folgenden Workflow sorgst du dafür, dass Fehler frühzeitig erkannt, gezielt dokumentiert und effizient behoben werden. Du brauchst nur drei Schritte – dann hast du einen belastbaren automatisierten Prüfpfad.

1. Starte mit einem Entwickler-Check (axe DevTools) Bevor du deinen Code committest, mach einen schnellen Check mit axe DevTools direkt im Browser. Scanne die bearbeitete Seite und konzentriere dich auf alle Fehler, die als „Critical" oder „Serious" eingestuft sind. Diese solltest du sofort beheben – damit sie gar nicht erst in den Testzyklus rutschen. So hältst du die Fehlerlast niedrig und sparst dir später viel Aufwand.

2. Führe in der QA einen umfassenden Scan durch (WAVE oder Lighthouse) Sobald dein Feature in der QA-Phase ist, führst du (oder jemand aus dem QA-Team) einen vollständigen automatisierten Test durch – mit WAVE oder Lighthouse . Ziel ist es, einen ersten Überblick zu bekommen: Welche offensichtlichen Barrieren gibt es? Wie sieht es mit der Seitenstruktur, Kontrasten oder fehlenden Labels aus? Erfasse alle relevanten Punkte direkt im Testprotokoll – das spart dir später viel Sucherei.

3. Dokumentiere deine Funde sauber Exportiere den Bericht oder erstelle gezielte Screenshots der wichtigsten Probleme. Hänge sie an das passende Ticket oder dein Testprotokoll an. Wichtig: Notiere genau , wo der Fehler auftritt, wie er sich zeigt und auf welches WCAG-Kriterium er sich bezieht. So gibst du dem Entwicklungsteam alles an die Hand, um den Fehler schnell zu beheben – ohne Rückfragen.

👉 Dein Ziel: Finde früh, dokumentiere klar und gib deinen Kolleg:innen alle Infos, die sie brauchen. Je systematischer du arbeitest, desto effizienter läuft die Fehlerbehebung – und desto näher kommst du einer wirklich barrierefreien Website.

3 Interpretiere die Ergebnisse – Nicht alles, was rot ist, ist ein Fehler

Die Tools nehmen dir das Denken nicht ab.

- Priorisieren nach Schweregrad: Konzentriere dich zuerst auf die als „Kritisch“ (Critical/Blocker) eingestuften Fehler. Das sind in der Regel eindeutige WCAG-Verstöße.
- Falsch-Positive erkennen: Manchmal meldet ein Tool einen Fehler, der im spezifischen Kontext keiner ist (z. B. ein Kontrastfehler bei einem rein dekorativen Element). Hier ist menschliche Bewertung gefragt.
- „Needs Review“ verstehen: Viele Tools markieren Punkte, die sie nicht eindeutig bewerten können (z. B. Farbkontrast bei einem Bild mit Farbverlauf). Diese Punkte sind direkte Arbeitsanweisungen für die manuelle Prüfung.

4 Kenne die Grenzen – Was automatisierte Tools NICHT finden

- Dies ist der wichtigste Punkt, um falsche Sicherheit zu vermeiden. Ein 100%-Score in Lighthouse bedeutet NICHT, dass deine Seite barrierefrei ist. Tools können grundsätzlich nicht prüfen:
- Logische Bedienbarkeit: Ist die Tab-Reihenfolge sinnvoll? Gibt es eine Tastaturfalle?
- Sinnhaftigkeit von Texten: Ist der Alternativtext für ein Bild wirklich hilfreich? Ist die Sprache der Fehlermeldungen verständlich?
- Tatsächliche Sichtbarkeit: Ist der Fokus-Indikator klar zu sehen oder wird er von einem Cookie-Banner verdeckt?
- Gestensteuerung in mobilen Ansichten.

👉 Faustregel: Automatisierte Tools finden ca. 30-50% aller möglichen Barrierefreiheits-Probleme. Der Rest erfordert manuelle Tests.

5 Vom Ergebnis zum Ticket

Ein gefundener Fehler, der nicht dokumentiert wird, existiert nicht.

- Nutze die Export-Funktion der Tools, um einen permanenten Link zum Ergebnis oder einen JSON-Report zu erstellen.
- Erstelle ein Ticket in deinem Projektmanagement-System (z. B. Jira).
- Füge alle relevanten Informationen hinzu: URL, Screenshot, das betroffene WCAG-Kriterium und eine klare Beschreibung des Problems.
- Verweise auf den automatisierten Test, der den Fehler gefunden hat.

Ergebnis

Am Ende dieser SOP hast du:

- Einen Prozess, der die Effizienz deiner Barrierefreiheits-Tests massiv erhöht.
- Die Fähigkeit, schnell und kostengünstig eine große Anzahl von technischen Fehlern zu identifizieren.
- Ein klares Verständnis für die Ergebnisse der Tools und kannst diese in konkrete Handlungsanweisungen umsetzen.
- Die perfekte Grundlage und Zeitersparnis, um dich bei der manuellen Prüfung auf die wirklich komplexen Probleme zu konzentrieren.

Die Barrierefreiheit von Apps erfordert präzise Planung und detaillierte Tests. Dieser Baustein dient als direkt nutzbares Kit für deine Entwicklungs- und QA-Teams, um sicherzustellen, dass jede App-Komponente und jeder Screen den hohen Standards der Zugänglichkeit entspricht. Es ist eine handlungsleitende Spezifikation und Testanweisung in einem, die über reine Richtlinien hinausgeht und die Umsetzung signifikant beschleunigt.

Betreff: Anforderung: Automatisierter Accessibility-Check vor jedem Commit

Hallo Team,

um die Code-Qualität zu erhöhen und Barrieren frühzeitig zu finden, ist vor jedem Commit/Merge-Request ein automatisierter Accessibility-Check mit axe DevTools für die bearbeiteten Komponenten durchzuführen.

Workflow:

- Führe den axe-Scan im Browser auf deiner lokalen Entwicklungs-Seite durch.
- Behebe alle als „Critical“ oder „Serious“ gemeldeten Fehler.
- Bestätige im Commit oder Pull-Request, dass der Scan durchgeführt wurde und keine kritischen Fehler mehr vorhanden sind.

Dies ersetzt nicht die manuelle QA, stellt aber sicher, dass wir eine saubere technische Basis haben. Danke!

Viele Grüße [Dein Name]

Fragen & Antworten

Warum ist es so wichtig, dass meine App die im Betriebssystem eingestellte Schriftgröße des Nutzers respektiert (Dynamic Type/Font Size), anstatt eine feste Schriftgröße zu verwenden?

Das ist entscheidend für Nutzer:innen mit Sehschwäche oder ältere Menschen , die eine größere Schrift benötigen, um Texte lesen zu können. Wenn deine App die systemweite Einstellung ignoriert, zwingst du diese Nutzer:innen, die App zu deinstallieren oder ein Vergrößerungsglas zu benutzen. Die Betriebssysteme (iOS, Android) bieten hierfür Mechanismen (wie Apples Dynamic Type oder Androids skalierbare Pixel sp), die Entwickler:innen nutzen müssen. Das Ignorieren dieser Einstellungen führt dazu, dass Text abgeschnitten wird oder überlappt, wenn der Nutzer die Schriftgröße erhöht, was die App unbenutzbar macht.

Die SOP erwähnt %22reine Farbinformationen%22 als Problem. Was ist, wenn mein Design ausschließlich farbige Indikatoren verwendet, um den Status anzuzeigen (z.B. ein roter Punkt für Fehler, ein grüner für Erfolg)?

Das ist ein häufiger Designfehler Für farbenblinde Nutzer:innen sind diese Informationen nicht erkennbar. Du musst immer einen zusätzlichen, nicht-farblichen Indikator verwenden. Beispiele:

- Fehler/Erfolg: Neben einem roten/grünen Punkt einen Text („Fehler“, „Erfolgreich“), ein Icon (Häkchen, Ausrufezeichen) oder eine Formänderung hinzufügen.
- Aktiv/Inaktiv: Bei einem aktiven Tab nicht nur die Farbe ändern, sondern auch eine Unterstreichung , eine fette Schrift oder ein Icon hinzufügen.

Die Farbe kann weiterhin als visuelle Verstärkung dienen, aber niemals als einziger Informationsträger.

Wie teste ich am besten die %22logische Reihenfolge%22 von Elementen mit VoiceOver oder TalkBack, besonders wenn die visuelle Anordnung komplex ist?

Der Test der logischen Reihenfolge erfordert systematisches Vorgehen , du:

- Starte oben links: Beginne mit dem ersten Element auf dem Bildschirm.
- Wische immer nach rechts: Gehe Element für Element durch (mit dem Standard-Wischgeste nach rechts bei VoiceOver/TalkBack).
- Vergleiche mit visueller Reihenfolge: Notiere dir die Reihenfolge, in der die Elemente vorgelesen werden. Entspricht diese der erwarteten Reihenfolge, in der ein sehender Nutzer die Elemente von oben nach unten, links nach rechts erfassen würde?
- Achte auf Sprünge: Wenn der Fokus unerwartet springt oder Elemente übersprungen werden, ist das ein Problem. Typische Fehler sind unsichtbare Elemente im Fokusbaum oder eine falsche Gruppierung.
- Element-Rotor (VoiceOver): Bei iOS kannst du auch den Rotor nutzen, um nach Überschriften, Links oder Bedienelementen zu springen und zu prüfen, ob alle vorhanden sind und die Hierarchie stimmt.

Ein langsames, bewusstes Durchgehen des Bildschirms ist hier der Schlüssel.

Meine App verwendet viele benutzerdefinierte Gesten (z.B. Wischen in bestimmten Mustern, langes Drücken mit mehreren Fingern). Wie kann ich hier Barrierefreiheit gewährleisten?

as ist eine große Herausforderung, da Screenreader selbst komplexe Gesten nutzen und deine App-Gesten damit kollidieren könnten.

- Vermeide komplexe Gesten: Wann immer möglich, setze auf Standard-Gesten oder Klicks/Taps.
- Biete Alternativen: Wenn eine spezielle Geste unverzichtbar ist, muss es IMMER eine gleichwertige, alternative Bedienung über Standard-UI-Elemente geben. Beispiele:
- Statt „Pinch-to-Zoom“ in einer Karte: Biete „Zoom-In“ und „Zoom-Out“ Buttons (+ und -).
- Statt Wischen für Navigation: Biete sichtbare Vor-/Zurück-Pfeile oder eine Liste.
- Statt Langdrücken für Kontextmenü: Biete ein Kontextmenü an, das über einen normalen Button (... -Icon) geöffnet wird.
- Dokumentation: Wenn du doch benutzerdefinierte Gesten nutzen musst, dokumentiere sie klar in der Hilfe der App und weise auf die Alternativen hin.

Das Ziel ist, dass keine Funktion der App ausschließlich über eine Geste bedienbar ist, die nicht allen Nutzern zugänglich ist.

Die SOP erwähnt, Barrierefreiheit in der App-Beschreibung zu erwähnen. Gibt es weitere Marketing- oder Kommunikationsmaßnahmen, die ich ergreifen sollte, um unsere Bemühungen zu zeigen?

Ja, du kannst deine Bemühungen auf verschiedenen Kanälen hervorheben:

- Eigene App-Website/Support-Bereich: Erstelle eine spezielle Seite oder einen FAQ-Bereich, der detaillierter auf die Barrierefreiheitsfunktionen deiner App eingeht. Hier kannst du auch Tutorials zur Nutzung mit Screenreadern anbieten.
- Blogposts/Pressemitteilungen: Berichte über wichtige Barrierefreiheits-Updates oder die Implementierung neuer Funktionen.
- Soziale Medien: Teile kurze Demos oder Tipps zur Nutzung der App mit assistiven Technologien.
- Nutzer-Feedback: Ermutige Nutzer:innen aktiv, Barrieren zu melden, und reagiere transparent auf Feedback und behebe gemeldete Probleme. Nichts schafft mehr Vertrauen als eine sichtbare Reaktion auf Nutzerbedürfnisse.

Das Zeigen deines Engagements für Barrierefreiheit kann die Wahrnehmung deiner Marke positiv beeinflussen und eine breitere Nutzerbasis ansprechen