Hochschule München

Fakultät für Mathematik und Informatik

Bachelorarbeit

Automatisiertes Testen von Barrierefreiheit

Betreuer:

Prof. Dr. Ullrich Hafner

Dr. Pascal Bihler

Autor:

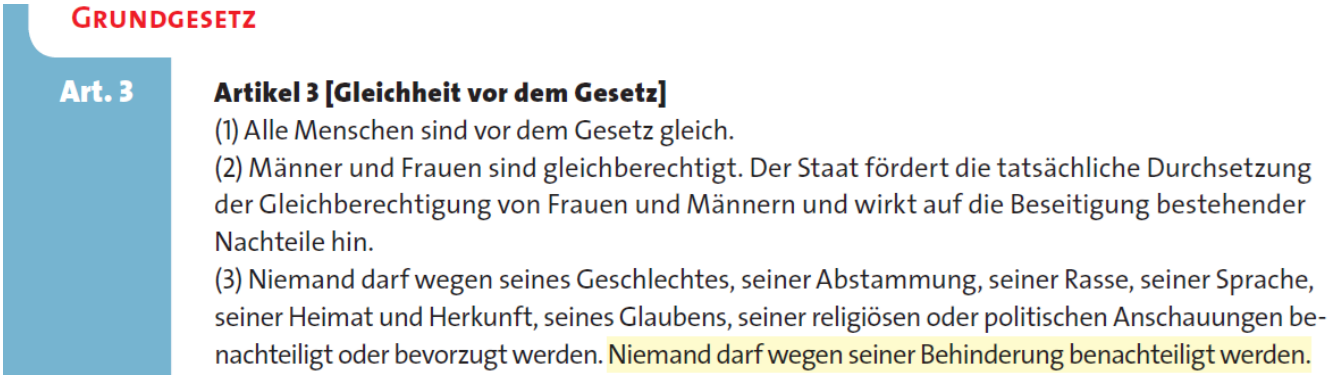
Daniel Rieth

Wintersemester 2024/2025

Inhaltsverzeichnis

1 Einleitung

Seit dem 15. November 1994 heißt es im Grundgesetz der Bundesrepublik Deutschland:



GRUNDGESETZ

Art. 3

Artikel 3 [Gleichheit vor dem Gesetz]

(1) Alle Menschen sind vor dem Gesetz gleich.

(2) Männer und Frauen sind gleichberechtigt. Der Staat fördert die tatsächliche Durchsetzung der Gleichberechtigung von Frauen und Männern und wirkt auf die Beseitigung bestehender Nachteile hin.

(3) Niemand darf wegen seines Geschlechtes, seiner Abstammung, seiner Rasse, seiner Sprache, seiner Heimat und Herkunft, seines Glaubens, seiner religiösen oder politischen Anschauungen benachteiligt oder bevorzugt werden. Niemand darf wegen seiner Behinderung benachteiligt werden.

Abbildung 1: Artikel 3 des Grundgesetzes der Bundesrepublik Deutschland

Der Anhang "*Niemand darf wegen seiner Behinderung benachteiligt werden*" wurde, nach langen Anstrengungen der Behindertenrechtsbewegung, nachträglich[1] im Jahr 1994 in den Gesetzestext aufgenommen.
Dieser Vorgang zeigt, dass es einen Wandel im Gesellschaftsverständnis gab - die Gleichberechtigung wurde explizit auf Menschen mit Behinderung ausgeweitet - dementsprechend wurde das Gesetz angepasst. Die rechtliche Stellung Behinderter wurde gestärkt, mehrere Gesetze und Regulationen zur Gleichstellung, Inklusion und barrierefreiem Zugang zu öffentlichen Einrichtungen wurden beschlossen.

Doch nicht nur auf der sozialen Ebene gab es Veränderungen. Die Technologie, besonders im Hinblick auf das Internet, hat sich seitdem rasant weiterentwickelt und tiefgreifende Veränderungen in nahezu allen Lebensbereichen hervorgebracht. In den Jahren vor und nach der Grundgesetzänderung erschienen mit NCSA Mosaic, Netscape Navigator und dem bekannten Internet Explorer die ersten grafischen Browser[2] und markierten den Beginn des massenhaften Zugangs zum Internet, welches bis dahin größtenteils nur Fachleuten und Akademikern zugänglich war.

Spätestens seit dem Erfolg des Smartphones ist das Internet einer der zentralen Pfeiler des gesellschaftlichen Lebens geworden. In im Jahr 2024 veröffentlichten Zahlen geht das Statistische Bundesamt Deutschland davon aus, dass 96% aller Einwohner Deutschlands Internetnutzer sind - bei Personen unter dem Alter von 65 Jahren ist der Anteil sogar noch größer [29].

Dieser technologische Wandel hat natürlich auch Auswirkungen auf den Gleichberechtigungsgedanken. Wo 1994 unter Barrierefreiheit vermutlich nur die Installation von Rampen für Rollstuhlfahrer oder vergleichbares verstanden wurde, muss der Begriff heute auch auf den digitalen Raum ausgeweitet werden. Gerade in Deutschland ist das Thema sehr aktuell, da mit dem Barrierefreiheitsstärkungsgesetz Mitte 2025 ein Gesetz in Kraft tritt, das auch Unternehmen dazu verpflichtet, Webseiten und Software barrierefrei zu gestalten.

Doch was bedeutet "barrierefreies Web" konkret? Was muss auf technischer Ebene für einen barrierefreien Webauftritt unternommen werden? Und wie kann die eigene Webseite auf Barrierefreiheit überprüft werden?

Diese Fragen soll diese Arbeit beantworten.

[1] Interview mit dem Behindertenbeauftragten der Bundesregierung
[2] Mozilla: The History of Web Browsers

1.1 Ziel der Arbeit

Das Ziel der Arbeit besteht darin, einen Überblick über das Thema "Testen von digitaler Barrierefreiheit" zu liefern, mit einem besonderen Fokus auf Barrierefreiheit im Web. Die dabei gewonnenen Erkenntnisse sollen verwendet werden, um die Funktionalität des UI-Testtools QF-Test um automatisierte Barrierefreiheitstests zu erweitern. Diese Tests werden in erster Linie für die Überprüfung von Webseiten entwickelt, dabei wird aber bewusst der Grundstein für eine Erweiterung der Tests auf andere Technologien, wie mobile Anwendungen oder PDFs gelegt. Bereits bestehende Testtools für Barrierefreiheitstests sollen verwendet und um Funktionalitäten erweitert werden, um Kundenanforderungen bezüglich Testabdeckung und Reporting beziehungsweise Protokollierung zu erfüllen.
Am Ende soll die Implementierung Kunden einen umfassenden, anschaulichen und intuitiv bedienbaren Zugang zur Überprüfung digitaler Barrierefreiheit des eigenen Webauftritts bieten.

1.2 Aufbau der Arbeit

Zunächst werden die Grundlagen zum Themengebiet Barrierefreiheit erläutert. Der Begriff "Barrierefreiheit" im Kontext von digitalen Inhalten wird erklärt, es werden Behinderungen und mögliche Barrieren aufgezeigt - und auch die technischen Hilfsmittel vorgestellt, die verwendet werden können, um diese Barrieren zu umgehen.
Zudem wird der rechtliche Rahmen beschrieben - also Gesetze und Standards, die insbesondere öffentliche Dienstleister und Unternehmen innerhalb der Europäischen Union, aber auch außereuropäische Firmen aktuell oder in absehbarer Zukunft betreffen. Insbesondere werden hier die Web Content Accessibility Guidelines (WCAG) und ihre grundlegenden Prinzipien erklärt. Diese Richtlinien liefern in der EU die Basis für die rechtliche Definition von digitaler Barrierefreiheit, sind aber generell in diesem Bereich als ein geltender Standard anerkannt.

Danach werden unterschiedliche Testmethoden der Barrierefreiheit vorgestellt und miteinander verglichen. Diese sind grob in zwei Kategorien zu unterteilen: manuelle und automatisierte Tests. Der Fokus dieser Arbeit liegt zwar auf automatisierten Tests. Doch aufgrund der Limitationen dieser automatisierten Überprüfungen müssen auch manuelle Tests berücksichtigt werden, wenn es um die Sicherstellung der Barrierefreiheit im Web geht.

Für automatische Tests gibt es bereits bestehende Bibliotheken, die eventuell in QF-Test integriert werden sollen. Deshalb wird ein Überblick über bestehende Tools gegeben, sowie Argumente für die Auswahl des letztendlich eingebundenen Tools erläutert.

Nach einem kurzen Überblick über die Software QF-Test selbst wird die tatsächliche Implementation beschrieben. Hierbei wird über die Einbindung der automatischen Testbibliothek und die eigen geschriebene Farbkontrastüberprüfung berichtet. Insbesondere die Gestaltung der für den Endnutzer wichtigen Testberichte und die dabei getroffenen Entscheidungen für deren Design wird genauer erläutert.

Danach wird das fertige Produkt in Verwendung gezeigt. In einer Fallstudie wird das ZPA (ein Organisationswerkzeug der Hochschule München für den Studienbetrieb) automatisiert auf Zugänglichkeit geprüft. Der Aufbau und die Ergebnisse der Tests sowie Lösungsvorschläge bei konkreten Fehlern in der Barrierefreiheit werden in diesem Kapitel präsentiert.

Zum Abschluss werden die Ergebnisse zusammengefasst und ein Ausblick auf die Entwicklungen im Bereich automatisierter Barrierefreiheitstests gegeben, sowohl im Kontext der Entwicklung des Accessibility-Projektes in QF-Test als auch in Bezug auf die Verwendung von künstlicher Intelligenz.

2 Grundlagen

2.1 Begriffserklärung: Barrierefreiheit (im Web)

Die Barrierefreiheit (*eng.* **Accessibility**, *Abk.* A11y von **A** + 11 Buchstaben "ccessibilit" + **y**) bezieht sich auf die Gestaltung von Lebensräumen, Produkten und Dienstleistungen, so, dass sie von allen Menschen, einschließlich Menschen mit Behinderungen, in gleicher Weise genutzt werden können. Das Ziel der Barrierefreiheit ist es, physische, kommunikative und digitale Barrieren abzubauen, um eine gleichberechtigte Teilhabe an der Gesellschaft zu ermöglichen.
Das Behindertengleichstellungsgesetz (BGG) [2, §4 Barrierefreiheit] nennt bei seiner Definition der Barrierefreiheit hierbei explizit die für diese Arbeit relevanten Gebiete "Systeme der Informationsverarbeitung" und "akustische und visuelle Informationsquellen und Kommunikationseinrichtungen". Die Nutzbarkeit soll laut BGG "grundsätzlich ohne fremde Hilfe" gewährleistet sein. Hierbei sind laut Gesetz allerdings auch "behinderungsbedingt notwendige Hilfsmittel" vorgesehen.

Als die gemeinhin gültige Definition von Barrierefreiheit im Web [4] wird die Definition der Web Accessibility Initiative (WAI) angesehen. Diese Untergruppe des World Wide Web Consortium (W3C) bezieht sich in ihrem Text direkt auf Websites und damit verbundene Tools und Technologien. Diese sollten so entworfen und entwickelt werden, dass Menschen trotz einer Behinderung das Web "wahrnehmen, verstehen, navigieren und damit interagieren" können und ihnen somit der Beitrag zum Web ermöglicht wird [43].

Das W3C sieht die Nutzen hier aber nicht nur für körperlich oder geistig behinderte Benutzer. Auch Menschen ohne Behinderungen profitieren von einer behindertengerechten Gestaltung des Internets. Konformität zu Accessibility-Standards bei der Entwicklung führen zu einer besseren User-Experience für Nutzer an Smartphones, Smartwatches, Fernseher mit Internetzugang oder anderen alternativen Endgeräten. Diese profitieren insbesondere von der Unterstützung alternativer Eingabemethoden und Bildschirmgrößen. Temporäre Einschränkungen, wie etwa ein gebrochener Arm, aber auch eine Reduktion der Sicht- oder Hörbarkeit (wie etwa durch starkes Sonnenlicht auf dem Bildschirm oder Baulärm) oder eine langsame Internetverbindung, stellen auf barrierefreien Web-Inhalten geringere Probleme dar.
Diese Einschätzung wird zum Beispiel durch eine Studie von Vollenwyder et al. unterstützt. Bei dieser wurden über 100 Versuchspersonen, zu gleichen Teilen bestehend aus Menschen mit und ohne visuellen Einschränkungen, bei ihrem Umgang mit einem selbstgebauten Online-Shop beobachtet. Nur eine dieser Webseiten wurde unter Berücksichtigung von Accessibility-Richtlinien entwickelt. Die Autoren der Studie schlossen aus den erhobenen Daten, dass auch Probanden ohne Sehschwächen ein besseres Nutzererlebnis bei der Verwendung des barrierefreien Online-Shops hatten. [34]

Auch andere Studien, etwa die Studienreihe von Schmutz et al. [28, 27, 26], belegen einen positiven Effekt von Accessibilitykonformität auf das Nutzererlebnis, Nutzerperformanz und Ästhetik von nicht-eingeschränkten Usern. Einzig die Verwendung von "einfacher Sprache" zeigte hier einen negativen Effekt: Die Testpersonen gaben an, den Text weniger ansprechend zu finden und weniger motiviert zu sein, die Website erneut zu besuchen [26].
Nicht zuletzt stärkt eine barrierefreie Website das Image eines Unternehmens oder einer Organisation als sozial verantwortungsbewusst und zukunftsorientiert. In einer zunehmend inklusiven Gesellschaft wird dies von vielen als wichtiges Differenzierungsmerkmal wahrgenommen. [49, S. 271f]

2.2 Benutzergruppen und ihre Bedürfnisse und Tools

Um zu verstehen, wie Barrierefreiheit im Umgang mit Informationstechnologie garantiert werden kann, ist es wichtig, die Einschränkungen zu kennen und zu verstehen, durch die Nutzer im Umgang mit der bereitgestellten Website oder Software beeinträchtigt sein können. Zudem muss den Entwicklern bekannt sein, welche Hilfsmittel ein solcher Nutzer verwenden kann und wie diese in ihrer Funktionsweise bestmöglich unterstützt werden können. Einen Überblick über mögliche Einschränkungen lässt sich auf der Website des WAI finden [42]. Dort werden die Kategorien Auditiv, Kognitiv/Lernen, Physikalisch, Rede/Sprache und Visuell gelistet. Yesilada et al. erweitern in ihrem Standardwerk [49] diese Aufzählung noch um "Alter" und "Situationelle Beeinträchtigungen und Behinderungen".

2.2.1 Eingeschränkte Sehfähigkeit

Ein großes Thema bei der Entwicklung von barrierefreier Software und Web-Inhalten sind Nutzer mit einer Sehstörung. Die WAI nennt drei Typen von eingeschränkter Sehkraft: Farbfehlsichtigkeit, eingeschränkte Sehkraft und Blindheit. [42]

Farbfehlsichtigkeit beschreibt die Schwierigkeit zwischen bestimmten Farben zu unterscheiden, im Volksmund oft auch als Farbenblindheit bekannt. Von der totalen Farbenblindheit spricht man im medizinischem Sinn erst bei der vollkommenen Unfähigkeit, jegliche Farben zu unterscheiden. Störungen im Farbsehsinn werden durch die Abwesenheit von funktionellen Farbzapfen hervorgerufen, die im Auge für das räumliche Auflösungsvermögen, aber eben auch für die Farbwahrnehmung zuständig sind [48].
Farbsehstörungen können Barrieren im Umgang mit Websites und Software darstellen. Nutzer mit einer solchen Einschränkung können Probleme haben inhaltliche Informationen aufzunehmen oder strukturelle Elemente zu erkennen [42]. Abhilfe kann hier beispielsweise ein Tool zur Farbkorrektur schaffen, welches im Browser oder der gesamten Benutzeroberfläche des jeweiligen Betriebssystems Farbtransformationen ausführen kann, um schwer unterscheidbare Farben in leicht zu unterscheidende zu ändern. Ein Beispiel dafür ist das Tool "Visolve"[1]. Zudem gibt es Websites und Browserextensions, wie etwa die Extension "Let's get Colorblind"[2], die es Entwicklern ermöglichen, die eigene Website durch die Augen einer Person mit Farbsehstörung zu betrachten, um sie auf diese Art zu testen und zu verbessern.

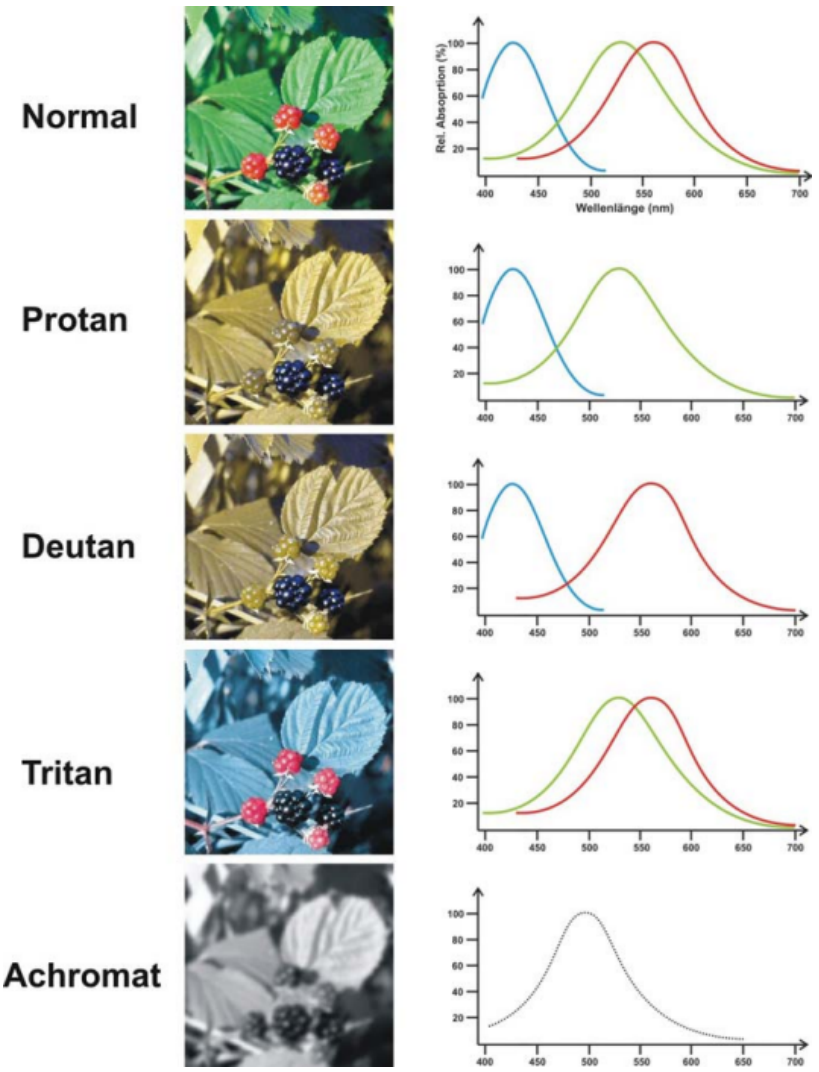


Abbildung 2: Farbwahrnehmung bei verschiedenen Farbsehschwächen: **Protan**opie (Rotblindheit), **Deuteran**opie (Grünblindheit), **Tritan**opie (Blaublindheit), **Achromat**opsie (Farbenblindheit)

Quelle: Universitäts-Augenklinik Tübingen 2005 [48]

Die eingeschränkte Sehkraft umfasst mehrere verschiedene Sehstörungen. Die WAI listet diese nach Effekt auf [42], da auch unterschiedliche Konditionen, wie etwa Kurzsichtigkeit, diabetische Retinopathie oder altersbedingte Makuladegeneration, für Nutzer die selben Probleme im Umgang mit Web-Technologie bedeuten [49, S. 8-11].

Das statistische Bundendesamt listet in der "Statistik der schwerbehinderten Menschen" 334.600 Menschen in Deutschland als sehbehindert auf. Davon gilt circa jeder fünfte (66.245) als blind, also mit einem Sehvermögen von weniger oder gleich 2% verglichen mit einem normalen Sehvermögen. [30]
Jedoch geht der Deutsche Blinden- und Sehbehindertenverband von einer höheren Zahl blinder und sehbehinderter Menschen in Deutschland aus, da für die Zählung des Statistischen Bundesamtes nur Menschen mit einem offiziellen

[1] Visolve
[2] Let's get colorblind

Schwerbehindertenausweis erfasst wurden [9]. Somit ist mindestens 0,07% der Gesamtbevölkerung (laut Destatis im Jahr 2021 etwa 84.000.000) blind, insgesamt gelten 0,3% der Bevölkerung als sehbehindert.

Die Effekte von Sehstörungen sind mannigfaltig. So kann die Sicht eines Nutzers nicht scharf oder klar sein, er kann sehr sensitiv auf Lichtquellen reagieren (**Photophobie**), Probleme bei der Unterscheidung von Farbkontrasten haben oder ein eingeschränktes Sichtfeld besitzen. Ein traditionelles Hilfsmittel bei beeinträchtigtem Sehvermögen, ein Vergrößerungsglas beziehungsweise eine Lupe, ist auch für Nutzer von Websites verwendbar. Viele Betriebssysteme, Browser und Softwares bieten bereits eine eingebaute Vergrößerungsoption an. Beispiele hierfür sind die Windows Bildschirmlupe unter den Barrierefreiheitsoptionen ("erleichterte Bedienung"), das Äquivalent auf MacOS ("Bedienungshilfen") oder die Zoomfunktion, die sich in allen gängigen Browsern finden lässt. Zudem lässt sich auch in vielen Browsern und Betriebssystemen die Schriftgröße umstellen.

Auch andere negative Effekte einer Sehstörung lassen sich mit digitalen Hilfsmitteln mildern. Um gegen Lichtempfindlichkeit vorzugehen, kann die Helligkeit der Anzeige reduziert werden. Ein beliebtes Hilfsmittel, das nicht nur bei Personen mit Photosensitivität großen Anklang findet, ist der in vielen Benutzerapplikationen vorhandene "Dark Mode", bei dem die Helligkeit einer Anwendung durch Verwendung von hellen Textelementen auf dunklem Hintergrund stark reduziert wird. Bei einer entsprechend stark verringerten Sehkraft kann auch die Verwendung eines sogenannten "Screen Readers" in Betracht gezogen werden. Ein Tool, das vor allem von Betroffenen der als nächsten beschriebenen Sehstörung Verwendung findet.

Als legal blind werden Personen in Deutschland angesehen, deren Sehvermögen auf dem "besseren Auge" maximal 2% des normalen Sehvermögens beträgt. Diese Prozentangabe kann sich hierbei je nach Art der Augenerkrankung auf die Wahrnehmung eines Anteils des normalen Sichtfeldes beziehen oder auch auf den Abstand, in welchem ein Gegenstand erkannt werden kann. [9]

Während Nutzer mit eingeschränkter, aber noch vorhandener Sehkraft mit Hilfsmitteln den Inhalt einer Website noch visuell aufnehmen können, muss für blinde Nutzer der Inhalt einer Website zwangsläufig über andere Sinne bereitgestellt werden. Das wohl geläufigste Tool hierfür ist der bereits erwähnte Screen Reader. Diese Software übersetzt visuelle Informationen auf dem Bildschirm in alternative Ausgabeformen. Meist präsentiert der Screen Reader die Information akustisch per Sprachsynthese, aber die Ausgabe kann auch taktil über ein Brailledisplay erfolgen. Neben sichtbaren Texten können Screen Reader ihren Nutzern auch andere Informationen zugänglich machen. Sie können versteckte Alternativtexte zu Bildern oder Grafiken vorlesen und auch die Struktur der aktuellen Website (Menüs, Eingabeelemente etc.) verdeutlichen. Bekannte Screen Reader sind das kommerzielle Tool JAWS[1] oder der kostenfreie, open-source Konkurrent NVDA[2]. Ebenso bieten gängige Betriebssysteme bereits vorinstallierte Screen Reader an. Beispiele sind etwa Windows *Narrator*, iOS/macOS *VoiceOver* oder Android *TalkBack*. Für die barrierefreie Gestaltung von Websites und Softwares ist es wichtig, die von Screen Readern genutzten Schnittstellen möglichst vollständig und fehlerfrei zu implementieren.

2.2.2 Eingeschränkte Motorik

Um eine Website oder Software optimal nutzen zu können, sind oft präzise Bewegungen des Nutzers nötig. Eine Maus muss zielsicher auf ein Element geführt werden, es sind Klicks, Doppelklicks oder auch "Drag & Drop"-Operationen im Einsatz. Auf einer Tastatur müssen teils mehrere Knöpfe gleichzeitig gedrückt werden. Die Bedienung über Touchscreens kann neben einfachem Tippen auch Wischbewegungen oder Bewegungen mit mehreren Fingern gleichzeitig umfassen. Hierbei ist Präzision und teilweise auch Geschwindigkeit wichtig; eine einzige ungewollte Berührung des Touchscreens kann die aktuelle Operation verlangsamen oder sogar abbrechen. Solche Aktionen stellen teils unüberwindbare Hürden für Nutzer mit motorischen Einschränkungen dar. [49, S. 20f]

Die Ursachen für motorische Einschränkungen, auch physische Einschränkungen genannt, können mannigfaltig sein. So kann beispielsweise die Muskelkontrolle gestört sein, sodass unkontrollierte Bewegungen stattfinden können. Etwa durch einen Tremor (Zittern), wie er oft bei Patienten mit Parkinson oder Multipler Sklerose auftreten kann [49, S. 22f].
Eine Bewegung kann für einen Nutzer auch aufgrund einer Lähmung, wie nach einem Schlaganfall, schwierig oder unmöglich sein. Weitere Hindernisse können zum Beispiel Schmerzen bei Bewegungen, hervorgerufen etwa durch

[1] JAWS (Job Access With Speech)
[2] NVDA (NonVisual Desktop Access)

Entzündungen wie einer rheumatoiden Arthritis, eine Einschränkung des Tastsinns oder gar das Fehlen von Fingern, Händen oder Armen durch Amputation sein. [42]

Bei geringen Einschränkungen reichen eventuell schon ergonomisch gestaltete Mäuse/Tastaturen aus, oder auch nur die Verwendung eines dieser beiden Steuervorrichtungen. Je nach Schweregrad der Behinderung müssen eventuell alternative Bedienelemente verwendet werden. Spezielle Hard- und Software kann selbst querschnittsgelähmten Personen den Zugang zum Internet erlauben. Medola et al. listen in [22] unter anderem durch Kopf-, Zungen- oder Augenbewegungen gesteuerte Bedienungsmöglichkeiten, Sprachsteuerungen oder sogar ein Hirn-Computer-Interface auf.

Um Websites und Software auch für diese Personengruppe zugänglich zu machen, sollte beispielsweise jede Funktionalität mit nur einem einzelnen Zeiger (Mauszeiger oder Kontaktpunkt mit einem Finger oder Stift auf dem Touchscreen) und ohne "Drag & Drop" ausführbar sein [45, WCAG §2.5.1, §2.5.7]. Zudem muss eine gewisse Mindestgröße bei Bedienelementen sichergestellt werden, die mit einem solchen Zeiger ausgewählt werden können [45, WCAG §2.5.8]. Für Nutzer, die nur eine Tastatur (ohne Maus) verwenden, muss die Navigation über die Oberfläche der Website beziehungsweise der Software möglich sein [45, WCAG §2.1].

2.2.3 Eingeschränkte Hörfähigkeit

Die Verwendung von multimedialen Inhalten in Software und auf Websites bieten Entwicklern eine gute Gelegenheit, die Benutzererfahrung zu verbessern und das Engagement der Nutzer zu steigern. Im Normalfall verarbeiten Menschen auditive und visuelle Informationen besser, was zu einer erhöhten Merkfähigkeit und Wiedererkennung führt. In Lerntools (*Learning Management Systems, LMS*) werden Multimediainhalte aus diesem Grund gerne genutzt, um einen gesteigerten Lerneffekt bei Schülern oder Studenten zu erzielen. [1]
Auch in ihrer Literaturanalyse [14] nennen Garett et al. die "*graphical representation*" als das zweitwichtigste Element (nach Navigierbarkeit) um "*user engagement*" zu steigern. Doch nicht für jeden Nutzer ist es möglich, das Potential dieser Inhalte auch voll auszuschöpfen.

Schwerhörige oder taube Menschen benötigen im Umgang mit Multimedia zusätzliche Unterstützung. Da sie Audioinhalte oft nicht wahrnehmen können, ist es oft hilfreich, diese auch auf alternativen Wegen bereitzustellen. Von Entwicklerseite bietet sich ein Transkript des zu Hörenden an. Bei Videos helfen Untertitel, den Inhalt zugänglich zu gestalten. Mit Hilfe von Tonbearbeitungsprogrammen kann auch der Kontrast des Gesprochenen zu Hintergrundgeräuschen erhöht werden, um Schwerhörigen die Informationsaufnahme zu erleichtern. Wenn möglich kann zu Videos zusätzlich ein Dolmetscher in Gebärdensprache angezeigt werden, wobei bei Weitem nicht alle Menschen mit eingeschränkter Hörfähigkeit Gebärdensprache verstehen. [42]

2.2.4 Eingeschränkte kognitive Fähigkeiten

Kognitive Einschränkungen haben sehr unterschiedliche Ursachen und Ausprägungen. Sie können sich unter anderem auf Gedächtnis, Lernvermögen, Aufmerksamkeit, Spracherfassung, Wissen und Verhalten in unterschiedlichster Weise auswirken. Ein häufiger Trugschluss ist, dass diese Behinderungen auch die Intelligenz der betroffenen Person mindern. Mit Intelligenz ist hier insbesondere der Intelligenzquotient (*IQ*) gemeint. Yesilada et al. widersprechen diesem Folgeschluss in [49]. In einem Beispiel listen sie Personen mit Down-Syndrom auf, die überall auf dem IQ-Spektrum zu finden sein können - von Analphabetismus bis Hochschulabschluss.
Auch die Autoren der WAI stimmen dieser Einschätzung zu. Sie erläutern zudem, dass eine Einschränkung in einem Bereich (beispielsweise eine Lese-Rechtschreibschwäche) nicht unbedingt einen Effekt auf andere Bereiche (beispielsweise mathematische Fähigkeit) hat. [42]

Usability-Probleme, auf die Menschen mit kognitiven Behinderungen oder Einschränkungen stoßen, sind weitesgehend die selben, über welche "normale" Nutzer stolpern. Im Allgemeinen unterscheidet sich nur die Schwere der Auswirkung [49, S. 54]. Somit ist die Beseitigung solcher Barrieren meist ein Schritt, um auch die eigene Software oder Website für alle Nutzer benutzerfreundlicher zu gestalten. Dies ist ein Trend, der im Bereich der Web-Accessibility häufiger zu beobachten ist und insbesondere im nächsten Abschnitt noch einmal veranschaulicht wird.

Probleme, die in besonderer Schwere kognitiv eingeschränkte Menschen betreffen, sind zum Beispiel schlecht strukturierter Inhalt, inkonsistent gestaltete Form oder Beschriftung von Bedienelementen, wenig intuitive Bedienung oder unübersichtliche Textblöcke. Dies kann in erster Linie durch durchdachtes Design und gute Präsentation der Inhalte vermieden werden.
Beispielsweise können große Textabschnitte anfangs in wenigen Worten zusammengefasst werden. Manche Webseiten, etwa die der Bundesregierung Deutschland[1], bieten wichtige Inhalte in "leichter Sprache" an, um Überforderung zu vermeiden. Der gesamte Abschnitt 3.2 der WCAG-Richtlinien beschäftigt sich mit der Vorhersehbarkeit bei der Bedienung, ein Punkt, der hier auch zu einem einfacheren Zugang für kognitiv eingeschränkte Personen beiträgt [45, §3.2].

2.2.5 Temporär eingeschränkte Fähigkeiten

Die oben genannten Einschränkungen (Sehfähigkeit, Motorik, Hörfähigkeit und kognitive Fähigkeit) können auch Nutzer ohne eine Behinderung oder dauerhafte Einschränkung betreffen. Für jede Kategorie lassen sich Beispiele finden, in denen ein ansonsten normaler Nutzer in seinen Fähigkeiten im Umgang mit Websites oder Software reduziert sein kann.

Eine motorische Einschränkung kann natürlich auch nur temporär sein. So kann beispielsweise ein gebrochener Arm oder die Bedienung eines Smartphones als Beifahrer auf einer ruckeligen Straße zu Problemen führen.
In lauten Umgebungen oder bei einem Hörsturz kann die auditive Wahrnehmung, bei starkem Sonnenlicht oder nach einer Operation am Auge die visuelle Wahrnehmung vorübergehend eingeschränkt sein.
Die oben genannten schlechten Bedingungen können sich natürlich auch auf Konzentration und Erinnerungsvermögen auswirken, womit eine temporäre kognitive Einschränkung gegeben ist. Menschen können beispielsweise auch nach einem Schlaganfall in ihrer Sprach- oder Lesefähigkeit eingeschränkt sein, diese aber vollständig wiedererlangen.

All diese Beispiele veranschaulichen, dass Barrierefreiheit auch für "ansonsten normale" Nutzer von Vorteil sein kann. Diese können sich in solchen Situationen der gleichen Hilfsmittel bedienen, die auch Nutzer mit Beeinträchtigungen verwenden. Außerdem profitieren sie von *Accessibility*-Schwerpunkten, wie etwa verbesserter Navigation und weniger fehleranfälliger Bedienung.

2.3 Gesetzliche Rahmenbedingungen und Standards

Dass Barrierefreiheit in der Webentwicklung in letzter Zeit immer mehr in den Fokus rückt, ist wohl insbesondere zahlreichen Fortschritten in der Gesetzgebung zu verdanken. Im Folgenden werden die in Europa und den USA geltenden und anstehenden Gesetze und Standards im Zusammenhang mit *Web-Accessibility* gelistet und beschrieben.

2.3.1 Richtlinie (EU) 2016/2102 - Barrierefreiheit im Internet

Um klare Rahmenbedingungen innerhalb der Europäische Union (EU) bezüglich der Barrierefreiheit im Web zu schaffen, verabschiedete das Europäische Parlament und der Rat der EU 2016 die *Richtlinie 2016/2102 "über den barrierefreien Zugang zu den Websites und mobilen Anwendungen öffentlicher Stellen"*. Die Richtlinie versucht die Web- und App-Repräsentation öffentlicher Dienste innerhalb der EU besser zugänglich zu machen und die innerhalb der Mitgliedsstaaten variierenden Standards zu harmonisieren. [5]

Wie in Artikel 288 des Vertrags über die Arbeitsweise der Europäischen Union festgelegt, sind Richtlinien ein Werkzeug der EU-Legislative, mit denen die EU ihren Mitgliedsstaaten rechtlich verbindliche Ziele vorgeben kann. Diese müssen innerhalb eines Zeitrahmens (in diesem Fall bis September 2018) in den Mitgliedsstaaten in geltendes Recht umgesetzt werden. Das genaue Vorgehen zum Erreichen des gemeinsamen Zieles bleibt hierbei den einzelnen Mitgliedern überlassen. [12]

Die Richtlinie 2016/2102 ist signifikant, da sie als erste Richtlinie das Thema Barrierefreiheit im Web als Hauptpunkt beinhaltet. Andere Richtlinien, wie etwa die Richtlinien 2014/24/EU (Öffentliche Auftragsvergabe), 2014/25/EU (Auftragsvergabe im Bereich von Wasser-, Energie- und Verkehrsversorgung) und 2000/78/EG (Gleichbehandlung in Beschäftigung und Beruf) beinhalten zwar Zielsetzungen zum Thema Web-Accessibility, beschäftigen

[1] Bundesregierung Deutschland - Leichte Sprache

sich aber hauptsächlich mit anderen Themen. [5]

Die Richtlinie listet im Bezug auf Web-Accessibility vier Grundsätze auf:

- **Wahrnehmbarkeit/Perceivable** - Informationen und Komponenten der Nutzerschnittstelle müssen den Nutzern in einer Weise dargestellt werden, dass sie sie wahrnehmen können
- **Bedienbarkeit/Operable** - Nutzer müssen die Komponenten der Nutzerschnittstelle und die Navigation handhaben können
- **Verständlichkeit/Understandable** - Informationen und Handhabung der Nutzerschnittstelle müssen verständlich sein
- **Robustheit/Robust** - Inhalte müssen robust genug sein, damit sie zuverlässig von einer Vielfalt von User Agents, einschließlich assistiver Technologien, interpretiert werden können

Die obige Auflistung wurde wörtlich aus [5, Abschnitt 37] zitiert und ist in der *EU-Norm EN 301 549* beschrieben [17]. Die Norm verweist auf die später genauer ausgeführten WCAG-Richtlinien.

Deutschland setzte die Richtlinie mit dem *"Gesetz [...] zur Umsetzung der Richtlinie (EU) 2016/2102 über den barrierefreien Zugang zu den Websites und mobilen Anwendungen öffentlicher Stellen"* [10] fristgerecht im Juli 2018 in nationales Recht um - hierbei wurden mehrere Paragraphen des Behindertengleichstellungsgesetzes angepasst. Zudem wurden sie 2019 in der Barrierefreie-Informationstechnik-Verordnung (**BITV 2.0**) verankert. Der Umfang der Regelungen umfasst nur die "öffentliche Verwaltung des Bundes", hat also im Gegensatz zu den folgenden Verordnungen der **EAA** und des **BFSG** keine Auswirkungen auf den privaten Sektor [15].

2.3.2 EAA (European Accessibility Act) - Digitale Barrierefreiheit

Bereits Ende 2015 begann die EU am European Accessibility Act (EAA) (Europäischer Rechtsakt zur Barrierefreiheit) zu arbeiten. Grund für die Arbeit an dem Gesetz ist die Behindertenrechtskonvention der Vereinten Nationen (**UN-BRK**). Die EU wollte diese mit einem zentralen Gesetz in Europa umsetzen. [8]

Nach vier Jahren Verhandlung trat der EAA als [6] *Richtlinie 2019/882 "über die Barrierefreiheitsanforderungen für Produkte und Dienstleistungen"* in Kraft. Kerninhalt der Richtlinie ist die Verbesserung der Barrierefreiheit von Produkten und Dienstleistungen für Menschen mit Behinderung. Der EAA betrifft verschiedene Bereiche, darunter die für Web-Accessibility relevanten Bereiche des Online-Handels und Zugangs zu elektronischen Dienstleistungen, aber auch verwandten Themen wie Computer, Betriebssysteme und Smartphones [11].

Wie auch *Richtlinie 2016/2102* bezieht sich der EAA auf die *EU-Norm EN 301 549*, also die WCAG-Richtlinien und die dort gelisteten Grundsätze. Der wichtigste Unterschied zur *Richtlinie 2016/2102* ist der Umfang der durch dieses Gesetz zur Einhaltung von Barrierefreiheitsregelungen verpflichteten Akteure: Im EAA betrifft dies nicht nur Behörden, sondern auch Unternehmen.

Die EU-Mitgliedsstaaten hatten bis 2022 Zeit, diese Richtlinie in nationales Recht umzusetzen. Firmen innerhalb der EU müssen die in der EAA geforderten Maßnahmen bis Ende Juni 2025 erfüllen. [6]

2.3.3 BFSG (Barrierefreiheitsstärkungsgesetz)

Die Umsetzung des EAA erfolgte in Deutschland mit dem 2022 verabschiedeten Barrierefreiheitsstärkungsgesetz (BFSG). Hier werden die "Barrierefreiheitsanforderungen für Produkte und Dienstleistungen" innerhalb Deutschlands definiert, sowie die betroffenen Sparten und Unternehmen.
Der Online-Handel, Hardware und Software und einige andere Bereiche müssen von Unternehmen (ausgenommen Kleinstunternehmen mit weniger als zehn Angestellten und unter einem Jahresumsatz von 2 Millionen Euro) nach diesem Gesetz barrierefrei gestaltet werden. Zudem regelt das Gesetz das Strafmaß bei Nichteinhaltung der Vorschriften - es können Vertriebsverbote, Abmahnungen und Bußgelder verhängt werden. [16]

Das Gesetz tritt zum 28. Juni 2025 in Kraft und ist auch der Grund für diese Arbeit. Unternehmen arbeiten an der Barrierefreiheit ihrer Web-Repräsentation, weswegen auch die Nachfrage nach Barrierefreiheitstests steigt. Im Bereich der Quality assurance (*Qualitätssicherung*) (QA) ist Web-Accessibility (neben AI) das größte Thema, wie zum Beispiel die Agenda der Testautomatisierungskonferenz *TACON*[1] oder das Programm der *expo:QA*[2] zeigt.

Wie auch im öffentlichen Dienst müssen Unternehmen in Deutschland nun die *EU-Norm EN 301 549* erfüllen. Somit gilt für Websites dieser Unternehmen, dass sie sich an die Standards der WCAG-Richtlinien halten müssen. [23]

2.4 WCAG (Web Content Accessibility Guidelines)

Die Web Content Accessibility Guidelines (WCAG) sind internationale Richtlinien, die von der Web Accessibility Initiative (WAI) des World Wide Web Consortium (W3C) entwickelt wurden. Sie legen Standards fest, um digitale Inhalte für Menschen mit Behinderungen zugänglich zu machen. Die WCAG umfassen die in der *EU-Norm EN 301 549* referenzierten vier Prinzipien: Wahrnehmbarkeit, Bedienbarkeit, Verständlichkeit und Robustheit. [45]

Abbildung 3: Die vier Prinzipien der Barrierefreiheit mit zugehörigen Anforderungen

Quelle: ITZBund, 2022 [18]

Die einzelnen Richtlinien der WCAG sind neben diesen Prinzipien auch in unterschiedliche Konformitätsstufen (Level) aufgeteilt. Hierbei gilt: je höher das Level einer Richtlinie, desto konformer mit den WCAG ist eine Website, die diese erfüllt. [18]

- **Level A** - niedrigste Stufe der Konformität, aber höchste Priorität. Die Erfüllung von Regeln mit Level A ist enorm wichtig, um einen barrierefreien Zugang zur Website zu haben.
  Beispiel: [45, WCAG §1.2.2] Für aufgezeichnete Audioinhalte müssen Untertitel bereitgestellt werden - hohe Priorität, da Inhalte sonst z.B. für Menschen mit Hörschädigung unzugänglich sind. Ausnahmen sind hier (wie auch in vergleichbaren Regeln) Inhalte, die klar als Alternativen zu Textinhalten ausgeschildert sind.

[1] TACON 2024 - Testing, Testautomatisierung und QM
[2] expo:QA - Conference on Software Testing & Quality Engineering

- **Level AA** - mittlere Konformitätsstufe, hohe Priorität. Level A und AA Richtlinien müssen erfüllt werden, damit eine Website nach der *EU-Norm EN 301 549* und damit dem [8, EAA]/[16, BFSG] den vorgeschriebenen Grad an Barrierefreiheit erfüllt.
  Beispiel:[45, WCAG §1.2.5] Für aufgezeichnete Videoinhalte gibt es eine Audiodeskription - mittlere Priorität, da für Screen Reader zugängige Untertitel blinden Personen bereits einen eingeschränkten Zugang zu Videoinhalten bieten können. Zwischen Level A und AA liegt der Unterschied bei Nichterfüllung oft zwischen komplettem (A) und teilweisem (AA) Ausschluss eingeschränkter Nutzer von Inhalten.
- **Level AAA** - höchste Stufe der Konformität. Die Einhaltung von Level AAA Richtlinien sind nach EU-Recht nicht erforderlich, aber gerade für zentrale Inhalte einer Seite erwünscht [18].
  Beispiel:[45, WCAG §1.2.6] Für aufgezeichnete Audioinhalte muss eine Übersetzung in Gebärdensprache bereitgestellt werden. Diese Regel bietet Menschen mit Hörschädigung eine Alternative zum Untertitel, welcher allerdings den Inhalt bereits zugänglich gemacht hat. Zudem beherrschen nicht alle betroffenen Personen Gebärdensprache. Beispielsweise nutzt in den USA nur etwa jede 60. Hörgeschädigte die American Sign Language, wobei der Großteil der Hörschäden altersbedingt ist [49, S. 36].

Der Webauftritt von Unternehmen muss innerhalb der EU alle WCAG des Levels A und AA erfüllen. Aktuell gültig sind nach *EN 301 549* die Regeln in Version 2.1, wobei zu erwarten ist, dass in Zukunft die im Dezember 2024 veröffentlichte Version 2.2 Pflicht wird [17]. Auch außerhalb der EU finden die WCAG Anwendung, beispielsweise werden sie auch in den USA verwendet, um die Einhaltung von Barrierefreiheit im Web nach dem *Americans with Disabilities Act* (ADA) zu prüfen [32].

2.4.1 WAI-ARIA (Web Accessibility Initiative - Accessible Rich Internet Applications)

WAI-ARIA ist ein von der W3C empfohlener Webstandard. Er bietet eine allgemeine Schnittstelle für assistive Technologien, über die Hypertext Markup Language (HTML) (und damit Webanwendungen) besser zugänglich gemacht werden können. Dies wird erreicht, indem weitere Attribute für HTML-Komponenten definiert werden, die von Hilfsmitteln wie Screen Readern benutzt werden können oder auch die Navigation per Tastatur erleichtern. [41]

Dies ist insbesondere notwendig, da viele fortgeschrittene Websites statt nativen HTML-Elementen Komponenten aus Frameworks oder Eigenbau verwenden, die von assistiven Technologien deshalb nicht standardmäßig erkannt werden können [47]. Die zusätzliche Verwendung von JavaScript für UI-Elemente kann diese Problematik noch verstärken.

Korrekt verwendete ARIA-Attribute helfen dieser Software dabei, die *Semantik* (Bedeutung) von Inhalten zu erkennen - Absätze werden als Absätze, Buttons als Buttons erkannt, selbst wenn sie nicht mit standardmäßigem HTML definiert sind. Die Autoren der WAI-ARIA Definition beschreiben das Verhältnis zwischen dem "*user agent*" (beispielsweise einem Browser) und Hilfsmitteln über die Barrierefreiheitsschnittstelle ARIA als eine Art "Vertrag". [38]

Obwohl nicht direkt in der Gesetzgebung oder den WCAG-Richtlinien erwähnt, hilft ARIA dabei, Webseiten barrierefrei zu gestalten. Insbesondere die Barrierefreiheitsprinzipien [45, WCAG §2 "Bedienbarkeit"] und [45, WCAG §1 "Wahrnehmbarkeit"] werden durch eine korrekt bediente ARIA-Schnittstelle unterstützt.

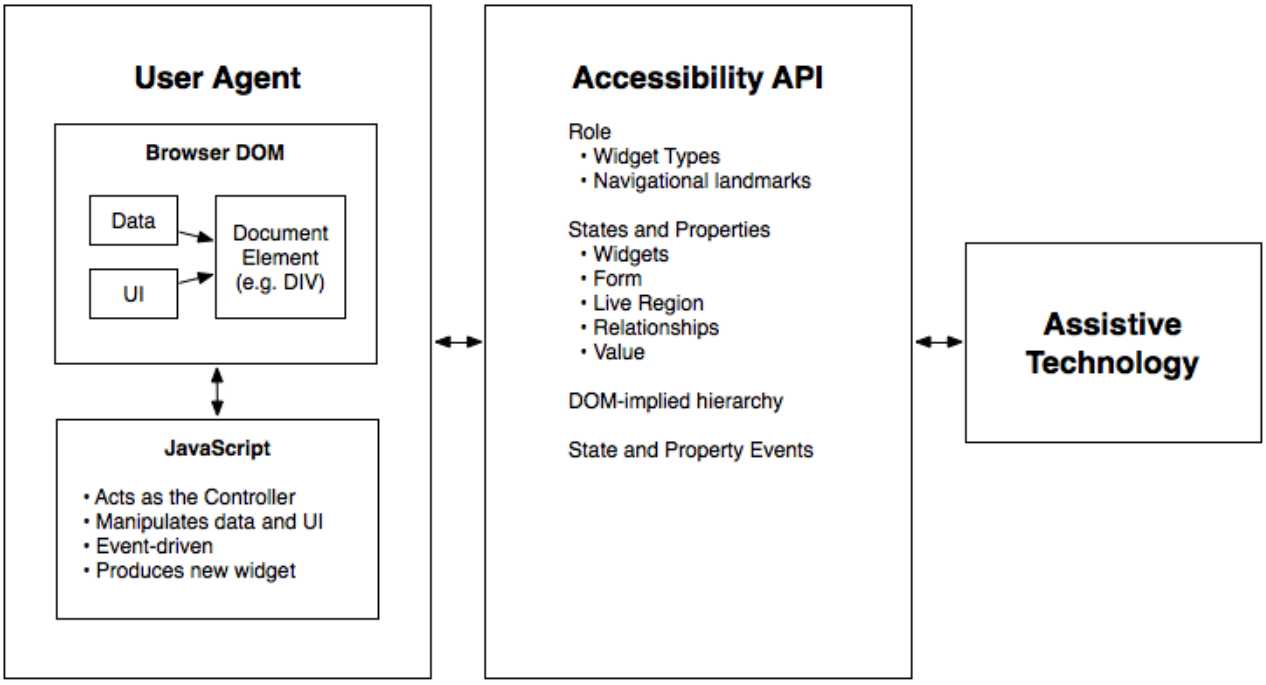


Abbildung 4: Vertragsmodell mit Hilfe einer Schnittstelle für Accessibility

Quelle: WAI-ARIA 1.3, 2024 [38]

3 Testmethoden der Barrierefreiheit

Eine barrierefreie Website ist für Unternehmen aus vielen, zum Teil bereits genannten Gründen sinnvoll: bessere Benutzererfahrung, ein besseres Image, eine größere Reichweite. Durch den beschriebenen gesetzlichen Rahmen ist Barrierefreiheit aber nicht mehr optional, Verstöße gegen Barrierefreiheitsstandards können rechtliche Konsequenzen haben. Um die Konformität des eigenen digitalen Auftritts mit der bestehenden Gesetzgebung - oder darüber hinaus - zu prüfen, gibt es mehrere Optionen.

3.1 Manuelle Tests

Bei manuellen Tests wird Accessibility von menschlichen Testern überprüft. Die wichtigsten Schwerpunkte sind Tastaturfunktion, visuelle Überprüfungen und Überprüfungen des Inhalts [46]. Hierbei gibt es zwei unterschiedliche Ansätze.

3.1.1 Expertentest

Hier werden Accessibility-Tests von geschulten Experten durchgeführt, beispielsweise durch von der *International Association of Accessibility Professionals* (**IAAP**) CPAAC[1]-zertifizierte Fachkräfte.

Im Allgemeinen wird bei Expertentests geprüft, ob eine Seite eine Liste von Zugänglichkeitskriterien erfüllt. Am häufigsten wird eine Konformitäts- oder Richtlinienprüfung durchgeführt. Dabei wird geprüft, ob eine Website eine Reihe von Accessibility Guidelines (beispielsweise die **WCAG**) erfüllt. [49, S. 481f]

Eine Methode etwa ist die von Giorgio Brajnik populär gemachte *Barrier Walkthrough* Methode. Anstatt Richtlinien für das Design einer Website direkt zu überprüfen, definiert Brajnik Barrieren, die bei Nichteinhaltung solcher Richtlinien entstehen können. Diese Barrieren entstehen aus der Perspektive verschiedener Nutzergruppen, wie etwa seh-, hörgeschädigte oder motorisch eingeschränkte Personen, zusätzlich aber auch aus technischen Perspektiven wie der einer Suchmaschine oder eines Browsers mit deaktiviertem JavaScript.
Danach durchlaufen ("Walkthrough") Experten die Benutzeroberfläche aus Sicht dieser Nutzergruppen und versuchen verschiedene Ziele zu erreichen. Dabei berücksichtigen sie die für die aktuelle Gruppe relevanten Barrieren und prüfen, ob diese bei der Erreichung des Ziels auftreten.
Zuletzt wird die Auswirkung einer Barriere auf den Nutzer bezüglich Parameter wie Effektivität, Sicherheit oder Zufriedenheit bewertet.[3].

Die WAI des W3C bietet ebenfalls eine Methode für manuelle Accessibility-Tests an. Die Website Accessibility Conformance Evaluation Methodology (WCAG-EM) wurde vom W3C entwickelt, um bei der Evaluierung und Verbesserung der Zugänglichkeit von Webinhalten zu unterstützen. Sie basiert auf den WCAG-Richtlinien und bietet eine strukturierte Herangehensweise, um sicherzustellen, dass digitale Inhalte den Anforderungen der Richtlinien genügen. Hierfür bietet die W3C auch ein Online-Tool[2] an, mit welchem sich die folgenden Schritte der WCAG-EM durchführen und dokumentieren lassen:
Nach der Definition des Ziels der Evaluation (beispielsweise Konformität bezüglich eines bestimmten Levels oder Version der WCAG) und der Identifikation der verwendeten Technologien (CSS, JavaScript, PDF...) durch eine "Exploration" der Website, muss eine repräsentative Stichprobe definiert werden. Das dient vor allem dazu, bei einem großem Webangebot den manuellen Test noch in einem vertretbaren Zeitrahmen zu bewältigen. Neben den ausgewählten Seiten sollten auch noch einige zufällige Seiten überprüft werden.
Im nächsten Schritt beginnt die eigentliche Evaluation. Die ausgewählten Seiten werden auf alle Richtlinien geprüft, die Ergebnisse werden dokumentiert. Zuletzt werden die Ergebnisse zusammengetragen und ein Score für die Barrierefreiheit berechnet. [44]

Inzwischen gibt es viele Anbieter, die Barrierefreiheitstests anbieten. Da es aber weder ein offizielles Zertifikat für Barrierefreiheit gibt, noch ein solches überhaupt sinnvoll ist (da sich Benutzeroberflächen häufig ändern), reichen einmalige Überprüfungen nicht aus. Die Tests müssen regelmäßig erfolgen, um eine tatsächlich barrierefreie Umgebung zu garantieren. Doch gerade bei einem großen Webangebot mit vielen, unterschiedlichen Seiten stoßen manuelle Tests an ihre Grenzen und werden zeitaufwendig und komplex [46].

[1] Certified Professional in Accessibility Core Competencies (CPACC)
[2] WCAG-EM Report Tool

3.1.2 Nutzertest

Ein intuitiver Weg, um Barrieren auf Webseiten zu entdecken, ist es, die Hilfe von Menschen mit Behinderungen in Anspruch zu nehmen. Hier werden Nutzer mit Einschränkungen oder Behinderungen damit beauftragt, bestimmte Tasks auf der zu prüfenden Seite durchzuführen.
Es müssen klare Ziele (in Form von Testaufgaben) gesteckt werden, die Personen der Testgruppe zu erfüllen versuchen sollen, wo benötigt mit Hilfe von ihnen genutzter assistiver Technologien. Idealerweise arbeiten die Probanden für authentische Ergebnisse in ihrer gewohnten Umgebung statt unter Laborbedingungen. Nicht nur das Feedback am Ende der Tests ist für die Evaluation wichtig, sondern auch Beobachtungen bezüglich des Verhaltens der Tester oder deren Äußerungen während des Prozesses. [19]

In [49] werden Nutzertests als "zuverlässigste Methode" zur Bewertung der tatsächlichen Accessibility einer Benutzeroberfläche beschrieben. Allerdings ist der Prozess sehr umständlich, zeitaufwändig und erfordert große Ressourcen und Zugang zu einer großen und diversen Gruppe an Testpersonen sowie geschulter Testleiter [21]. Auch muss beachtet werden, dass eine geringe Menge an Nutzern nicht unbedingt repräsentativ für den durchschnittlichen Nutzer stehen. Die Diversität von Behinderungen, aber auch von technischen Hilfsmitteln (und technischem Können), können zu sehr unterschiedlichen Benutzererlebnissen führen [19].

Für solche Nutzertests gibt es inzwischen auch Serviceanbieter[1], die diese Probleme für interessierte Unternehmen lösbar machen.

3.2 Automatisierte Tests

Automatisierte Tests sind auch im Kontext der Barrierefreiheit eine effiziente Methode, um Probleme in der eigenen Benutzeroberfläche zu finden. Unter Verwendung verschiedener Werkzeuge werden schnell große Teile einer Website analysiert und mögliche Verstöße gegen Barrierefreiheitsrichtlinien aufgedeckt.

3.2.1 Vor- und Nachteile im Vergleich zu manuellen Tests

Automatische und manuelle Barrierefreiheitstests bieten unterschiedliche Stärken und Schwächen bei der Überprüfung von Web-Accessibility.

Automatische Tests zeichnen sich durch hohe Effizienz und Geschwindigkeit aus, da sie schnell grundlegende, aus dem Code lesbare Fehler identifizieren können. Beispiele für solche Fehler sind etwa fehlende Alternativtexte [45, WCAG §1.1] oder Farbkontraste bei Textelementen [45, WCAG §§1.4.3, 1.4.6] - Tests, die manuell sehr zeitaufwändig und umständlich sein können [21].
Zudem sind automatische Tests gerade bei großen Projekten sinnvoll, da sie kostengünstig und leicht skalierbar sind. Ihre Fehlerabdeckung beschränkt sich jedoch auf technische Aspekte und standardisierte Prüfungen [46] - Inhalte, Zusammenhänge und Benutzererfahrung können nur sehr bedingt geprüft werden [49, S. 481, 483]. Meist produzieren automatische Testtools deswegen zwei mögliche Resultate: Fehler, bei denen das Tool eindeutige Barrieren feststellen konnte, und Warnungen, mit denen es auf Stellen hinweist, die Barrieren enthalten können, aber zu deren endgültiger Evaluation es menschliche Experten benötigt [49, S.484] - auch wenn es an dieser Stelle Entwicklungen mit Hilfe von künstlicher Intelligenz gibt.

In [33] prüfen beispielsweise Vigo et al. mehrere automatische Testtools in Bezug auf Vollständigkeit, Korrektheit und Abdeckung zu den WCAG-Richtlinien. Keines der geprüften Tools konnte eine Abdeckung von mehr als 50% der Richtlinien anbieten. Die Studie zeigte auch Probleme im Bereich der Vollständigkeit und Korrektheit der Ergebnisse auf, weswegen die Autoren vor "over-reliance" auf ausschließlich automatische Tests warnen.

Manuelle Tests hingegen bieten eine tiefere Fehlerabdeckung, da sie das Benutzererlebnis unter realen Bedingungen simulieren. Sie sind in der Lage, komplexe Interaktionen und kontextabhängige Barrieren zu erkennen, welche von automatisierten Tools eher übersehen werden [19]. Allerdings sind sie zeitaufwendig und kostspielig, da sie die kontinuierliche Beteiligung qualifizierter Tester (und eventuell Testpersonen) erfordern. Zudem sind sie im großen Rahmen schlecht skalierbar. Die Konsistenz der Ergebnisse und die Wiederholbarkeit dieser Tests kann variieren, da sie vom Urteil des Testers abhängen [46].

[1] acessiBe User Testing by Experts with Disabilities

Kriterium	Automatische Tests	Manuelle Tests
Geschwindigkeit	Schnell, leicht zu wiederholen	Langsam, aufwändig zu wiederholen
Kosten	Gering nach Implementierung	Hoch durch Personalkosten und Zeitaufwand
Fehlerabdeckung	Grundlegende, codebasierte Fehler	Potentiell allumfassend - abhängig von Tester
Komplexität	Normalerweise einfache, codebasierte Tests	Komplexe Zusammenhänge und Inhalte prüfbar
Fehleranfälligkeit	Praktisch nur bei Implementierung	Menschliche Fehler nicht ausgeschlossen
Skalierbarkeit	Leicht skalierbar auf große Projekte	Schwer skalierbar, erfordert qualifizierte Tester

Tabelle 1: Vergleich von automatischen und manuellen Accessibility-Tests

Die Kombination beider Testansätze ermöglicht eine umfassende Bewertung der Barrierefreiheit. Automatische Tests bieten eine schnelle Identifikation technischer Fehler, während manuelle Tests tiefere Einblicke in die tatsächliche Nutzererfahrung liefern. Diese Einschätzung wird von vielen Fachkundigen und Forschern geteilt (beispielsweise [31], [49], [21] und [46]).

4 Tools für automatisiertes Testen

Im Bereich der Web-Accessibility gibt es einige Optionen, um Seiten automatisch auf Barrieren überprüfen zu lassen. Diese Tools sind teilweise in unterschiedlichen Formen erhältlich, beispielsweise als Browser-Plugin oder Online-Tool für eine manuelle Inspektion oder als Skript oder API, welche in Testläufe eingebunden werden kann. Für den Rahmen dieser Arbeit sind insbesondere Skripte oder API-Aufrufe interessant, da die automatische Testengine in die Software QF-Test integriert werden soll.

4.1 Accessibility Conformance Testing (ACT) Regeln

Mit **ACT**-Regeln versucht die W3C einen Standard für Tests für Konformität gegenüber Barrierefreiheitsstandards, wie etwa WCAG oder WAI-ARIA, zu schaffen. Es soll das Testen von Barrierefreiheit im Web "transparenter" gestalten und dadurch "Verwirrung bei unterschiedlicher Interpretation von Accessibility-Richtlinien" beseitigen. [36]

Hierfür entwickeln die Autoren Regeln[1] nach einem standardisiertem Format. ACT-Regeln enthalten beispielsweise einen anschaulichen Titel, eine eindeutige Identification (ID), eine Beschreibung, den damit überprüften Barrierefreiheitsstandard und sie listen Testfälle auf, die zu einem bestimmten Ergebnis (bestehen/passed, nicht bestehen/failed, nicht anwendbar/inapplicable) führen sollen. [37]

Ein Beispiel hierfür ist die Regel "*HTML page has lang attribute*". Sie listet unter anderem die damit verwandte WCAG-Richtlinie auf ([45, WCAG §3.1.1 "Language of Page"]), definiert das Anwendungsgebiet (*document element* im HTML mit Zusatzbedingungen) und liefert minimale Codebeispiele für (in-)korrekt implementierte Seiten, aber auch Elemente, auf die die Regel nicht anwendbar ist.

Dabei beziehen sich ACT-Regeln sowohl auf automatische, semi-automatische als auch manuelle Testmethoden. Die Entwickler solcher Testprogramme oder Testmethoden können nun die Überprüfung dieser Regel in ihr Programm oder ihre Methode einbinden. [37]

Automatische Testtools, die sich bei der Implementierung ihrer Testregeln an ACT halten, lassen sich in der Auswertung ihrer Ergebnisse und ihrer Testabdeckung von Barrierefreiheitsstandards besser miteinander vergleichen [39]. Insbesondere definieren die Regeln auf technischer Ebene, was in den WCAG-Richtlinien definiert wurde. Auf das obige Beispiel bezogen: Wo die WCAG-Richtlinie 3.1.1 fordert, dass "[d]ie voreingestellte menschliche Sprache jeder Webseite [...] durch Software bestimmt werden [kann]", liefert die ACT-Regel konkrete Anweisung, wie das im HTML überprüft werden kann.

Allerdings ist zu erwähnen, dass ACT-Regeln lediglich informativ sind [40]. Wird eine ACT-Regel eingehalten, bedeutet das noch nicht, dass die Seite im Bezug auf die zugehörige WCAG-Richtlinie konform ist. Die Regeln bieten lediglich einen technischen Leitfaden, um bestimmte Richtlinien zu erfüllen.

4.2 Überblick über verfügbare Tools Web

Da die WCAG in vielen Ländern die legale Definition von Barrierefreiheit im Internet darstellt, prüfen die meisten automatischen Testtools die zu testenden Seiten auf diese Richtlinien. Deswegen werden sie auch häufig "*WCAG-Checker*" genannt [13]. Die W3C bietet einen Überblick über Testtools, von denen im Folgenden einige vorgestellt werden.

4.2.1 WAVE

Die amerikanische Non-Profit Organisation WebAIM (Web Accessibility in Mind) stellt mit WAVE[2] (Web Accessibility Evaluation Tool) Tools zur Evaluation von Web-Accessibility bereit. Neben einem kostenlosen Online-Tool und Browser Plugins für Chrome, Edge und Firefox bietet WAVE eine kostenpflichtige REST-API zur Überprüfung von einzelnen Webseiten. Zudem bietet WAVE einen sogenannten Acessibility IMpact (*AIM*) Report an, bei dem neben automatisierten Tests auch noch menschliche Experten manuelle Tests durchführen und die Zugänglichkeit

[1] List of all ACT Rules
[2] WAVE Web Accessibility Evaluation Tools

der überprüften Webseite bewerten. Der Code, den WAVE verwendet, um die Accessibility zu testen, ist nicht Open Source.

Das Online-Tool und Plugin funktionieren auf ähnliche Weise: Durch die Angabe der zu prüfenden Webseite (beziehungsweise das Ausführen des Plugins auf der Seite) werden verschiedene Regeln überprüft.

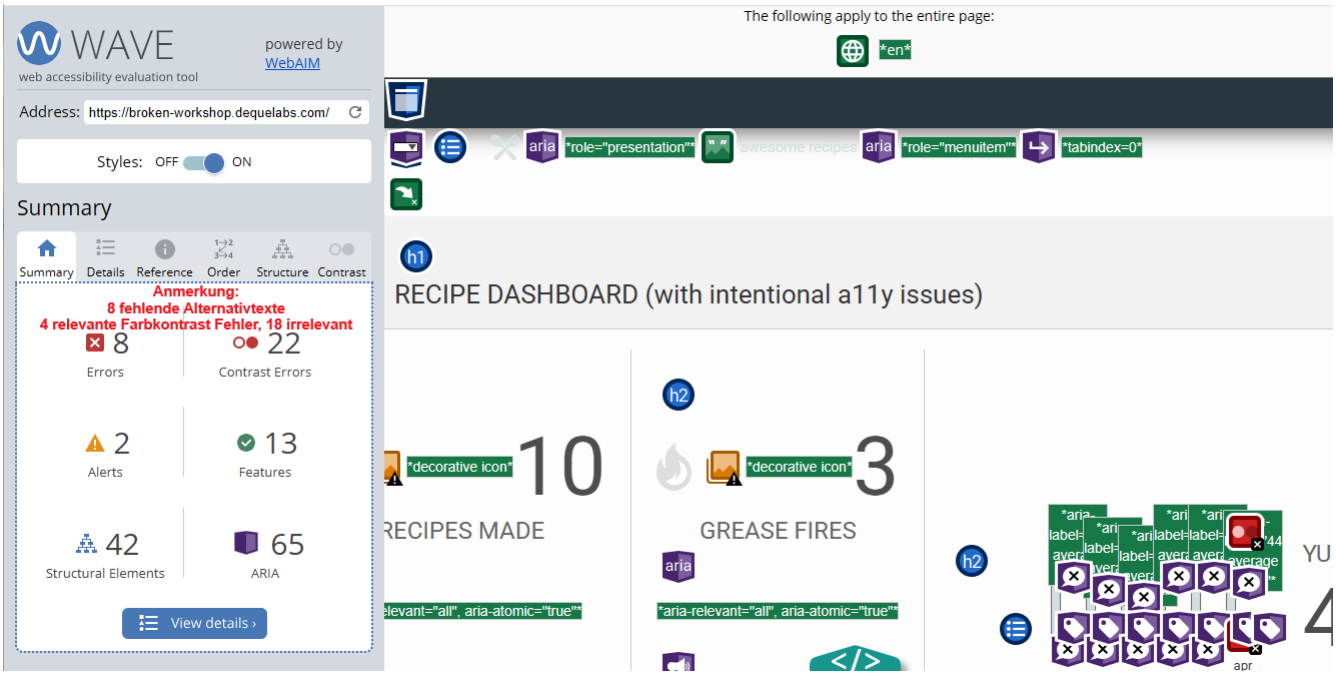


Abbildung 5: Ergebnisse der Überprüfung einer Demoseite mit dem WAVE-Online-Tool

Quelle: WAVE Webaim, Demoseite gestellt von deque

Fehler werden sowohl mit Hilfe von Icons auf der Seite selbst identifiziert, als auch in den Details des Reports gelistet. WAVE sortiert Probleme, die es findet, in 6 Kategorien:

- **Errors** - Allgemeine Fehler in der Barrierefreiheit der Seite, Verstoß gegen WCAG
- **Contrast Errors** - Fehler im Farbkontrast bei Textelementen auf der Seite, Verstoß gegen WCAG
- **Alerts** - Eventuell ein Fehler, aber nicht automatisch determinierbar, manuelle Überprüfung erforderlich
- **Features** - Element mit Vorschlag für die Verbesserung von Accessibility
- **Structural Elements** - Elemente, die für die Struktur der Seite verwendet wurden
- **ARIA** - Zeigt auf, wo ARIA (Accessible Rich Internet Applications) im HTML verwendet wurde, um Elemente besser zugänglich zu machen

In der Referenz werden Fehler genauer erklärt, Lösungsvorschläge gegeben, der Algorithmus zur Prüfung dieses Fehlers zusammengefasst und die relevanten WCAG-Richtlinien verlinkt.
Der Reiter "Order" zeigt die Reihenfolge, in der per Tastatur über die Seite navigiert werden kann. Damit lassen sich manuell Richtlinien, die sich auf die Zugänglichkeit per Tastatur beziehen [45, WCAG §2.1 "Keyboard accessible"], überprüfen.
Neben einer Zusammenfassung der Struktur des Document Object Model (DOM) bietet WAVE auch eine genauere Betrachtung von Farbkontrastfehlern an [45, WCAG §1.4.3 "Contrast (minimum)"]. Hier werden die ermittelten Farben und Kontraste eines überprüften Elementes präsentiert, sowie die Möglichkeit, über eine Farbauswahl eine Farbkombination mit ausreichendem Kontrast zu finden.

Für einen schnellen Überblick über Fehler bietet das WAVE Online-Tool eine gute Oberfläche und leichte Bedienbarkeit. Die Einbindung in automatisierte Testläufe ist jedoch schwieriger. Zwar sind Requests an die WAVE

Subscription API einfach zu bewerkstelligen, aber für das Testen von nicht öffentlich zugänglichen Seiten, beispielsweise Seiten in einer Produktionsumgebung, muss die WAVE Engine als Standalone lokal installiert werden. Der Preis für die Lizenz ist vermutlich in den meisten Anwendungsfällen höher als die Kosten für Requests an die Standard API - ausgenommen bei sehr großen Projekten mit vielen zu überprüfenden Seiten.

Ein weiterer Kritikpunkt bei automatisierten Testläufen per API ist der Nutzen für Debugging. WAVE bietet einen Überblick über die Anzahl der Fehler, aber fehlerhafte Elemente werden in der XML- oder JSON-Antwort nur per XML Path Language (XPath)[1] identifiziert. Dies ist unvorteilhaft, insbesondere für Mitarbeiter in der Qualitätssicherung, die über geringe technische Fachkenntnisse verfügen. Abgesehen davon kann sich ein XPath ändern, wenn sich der Inhalt der Seite ändert oder die Seite seit dem Test weiterentwickelt wurde.
WAVE implementiert keine ACT-Regeln und kann deshalb nicht leicht mit anderen Tools auf Abdeckung verglichen werden.

4.2.2 Alfa

Alfa[2] ist eine Open Source Testengine der Firma Siteimprove, welche als Dienstleister Websites von Kunden betreut und optimiert.

Siteimprove nutzt und entwickelt Alfa für den automatischen Teil ihrer Barrierefreiheitstest und wird dabei von der EU gefördert. Alfa wird in mehreren Produkten, wie der Siteimprove Browser Extension oder dem in CI-Piplelines einbindbaren Accessibility Code Checker[3] verwendet. Zudem lässt sich Alfa auch manuell als Skript ausführen, um eine Auswahl an vordefinierten (oder selbstdefinierten) Regeln zu prüfen.

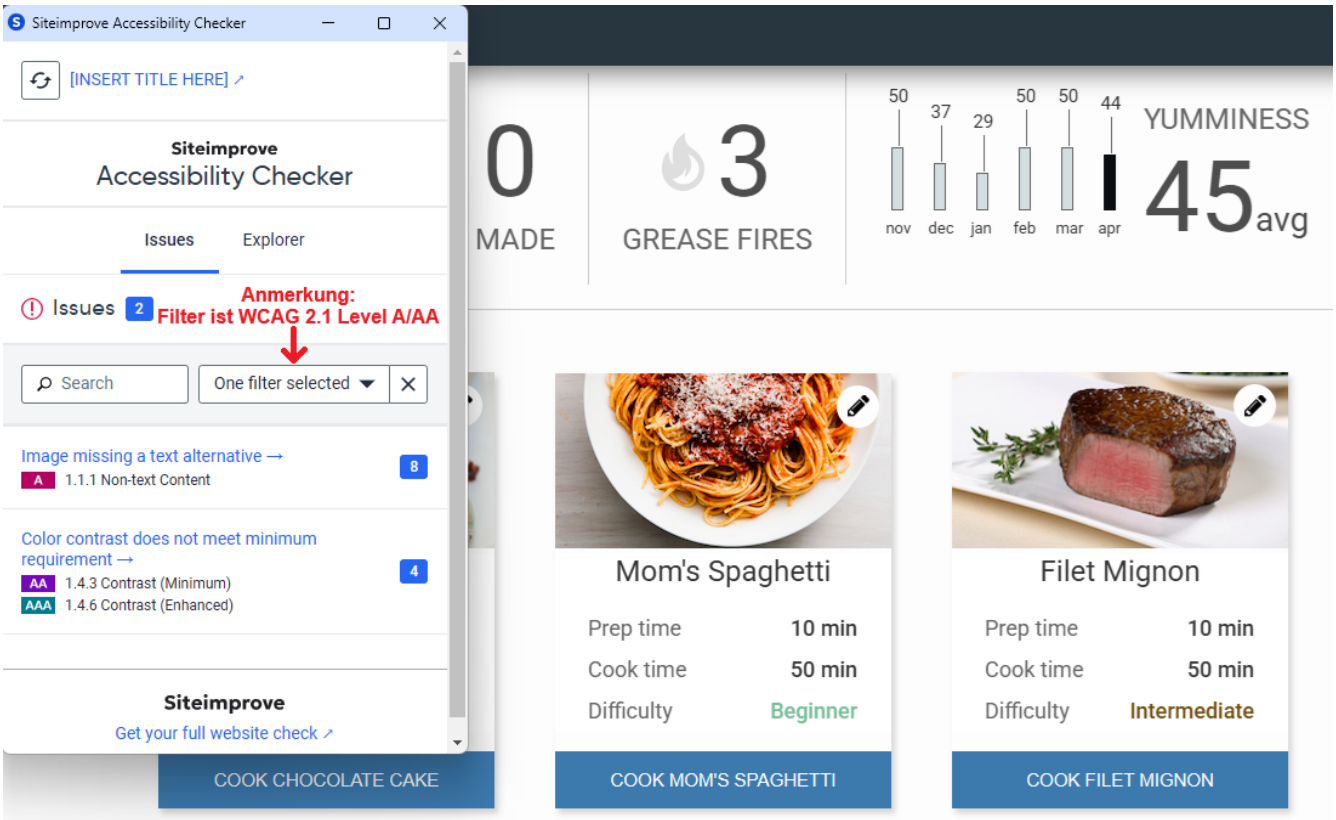


Abbildung 6: Ergebnisse der Überprüfung einer Demoseite mit dem Alfa-basiertem Siteimprove-Browser-Plugin

Quelle: Siteimprove, Demoseite gestellt von deque

[1] Einführung zu XPaths, W3C
[2] Github: Alfa, benannt nach einem Blindenhund
[3] Accessibility Code Checker, kompatibel mit Frameworks wie Cypress, Playwright, Selenium...

Die Browser Extension bietet einen schnellen Überblick über Barrieren auf der aktuellen Seite. Fehler werden mit der zugehörigen verletzten WCAG-Richtlinie und der Alfa-internen Regel angezeigt. Zudem sind zusätzliche Informationen zu den Fehlern zu finden, wie etwa Lösungsvorschläge und eine ausführliche Erklärung. Bei der Auswahl eines einzelnen Fehlers wird das fehlerhafte Element hervorgehoben und praktischerweise für das Debugging die Option gegeben, das Element direkt in den Entwicklertools des Browsers zu betrachten.

Als zusätzliche Funktion unter dem Reiter "Explorer" lässt sich die Seite noch aus der Sicht einer farbenblinden Person betrachten, sogar mit den unterschiedlichen Arten der Farbenblindheit.
Die zusätzlichen Funktionen der Browsererweiterung sind zwar eine willkommene Hilfe bei manuellen Überprüfungen, aber irrelevant bei der Auswahl der Testengine für die Integration in QF-Test. Hier punktet Alfa mit regelmäßigen Updates, keinen Lizenzkosten und großer Abdeckung. Ein Gegenargument aber ist, dass die Verwendung von Alfa viele Abhängigkeiten mit sich bringt und die Umwandlung von Ergebnissen aus Alfa-Testläufen in ein Format, das in QF-Test Protokollen und Reports verwendet werden kann, sehr umständlich ist - unter anderem auch, da Alfa keine deutsche Übersetzung der Ergebnisse liefert, QF-Test aber zweisprachig (Englisch und Deutsch) ausgeliefert wird.

Das Tool definiert mehrere *SIA-Rules* (SiteImprove Accessibility Regeln). Dabei sind einige spezifisch von Siteimprove definiert, aber die meisten direkte Implementationen von ACT-Regeln. Hierbei deckt Alfa in der semi-automatischen Browsererweiterung von Siteimprove 26, in der automatischen Version 24 der 34 akzeptierten ACT-Regeln ab, sowie einige der vorgeschlagenen Regeln. [39, Stand 5. Januar 2025, Version 0.80.0]

4.2.3 axe-core

axe-core[1], kurz axe, ist ein Open-Source-Testtool für Webseiten und andere HTML-basierte Benutzeroberflächen der Firma Deque Systems. Das Tool findet nach eigenen Angaben 57% aller Probleme mit nicht eingehaltenen WCAG-Richtlinien automatisch und produziert keine falsch-positiven Ergebnisse [7].
Die Testengine kann per *<script>*-Element als ausführbarer JavaScript-Code in HTML eingebunden werden, findet aber auch in unterschiedlichen Browsererweiterungen (wie axe-DevTools[2] oder Acessibility Insights[3]) oder für die Barrierefreiheitsbewertung des allgemeinen Website-Testtools Google Lighthouse[4] Verwendung - auch in Kombination mit AI.

Anders als die beiden zuvor vorgestellten Browsererweiterungen findet sich axe DevTools nach der Installation als zusätzlicher Reiter in den Entwicklertools des Browsers wieder.
Das Tool bietet mehrere Funktionen an. Neben der klassischen Überprüfung der Seite gibt es in der kostenpflichtigen Pro-Version auch eine Führung durch manuelle Tests ("Intelligent Guided Tests") oder die Option, die Seite während einer manuell simulierten Nutzerinteraktion in verschiedensten Zuständen zu überprüfen.

Fehlerhafte Elemente, die durch das Tool gefunden werden, können auf der Seite hervorgehoben oder in den Entwicklertools im Seitenquelltext angezeigt werden. Zudem liefert axe-core für diese Elemente einen CSS-Selector, über den die Elemente eindeutig identifiziert werden können.
Dieser Selector kann beispielsweise über Selenium (`driver.findElement(By.cssSelector('<selector>'))`) oder JavaScript (`Document.querySelector('<selector>')`) verwendet werden, um das entsprechende Element zu finden. Er ist robuster gegenüber Änderungen im Vergleich zu anderen Identifikatoren wie XPath.

[1] Github: axe-core
[2] axe DevTools
[3] Accessibility Insights, verwendet axe-core für automatische Checks
[4] Google Lighthouse, verwendet im Code axe-core

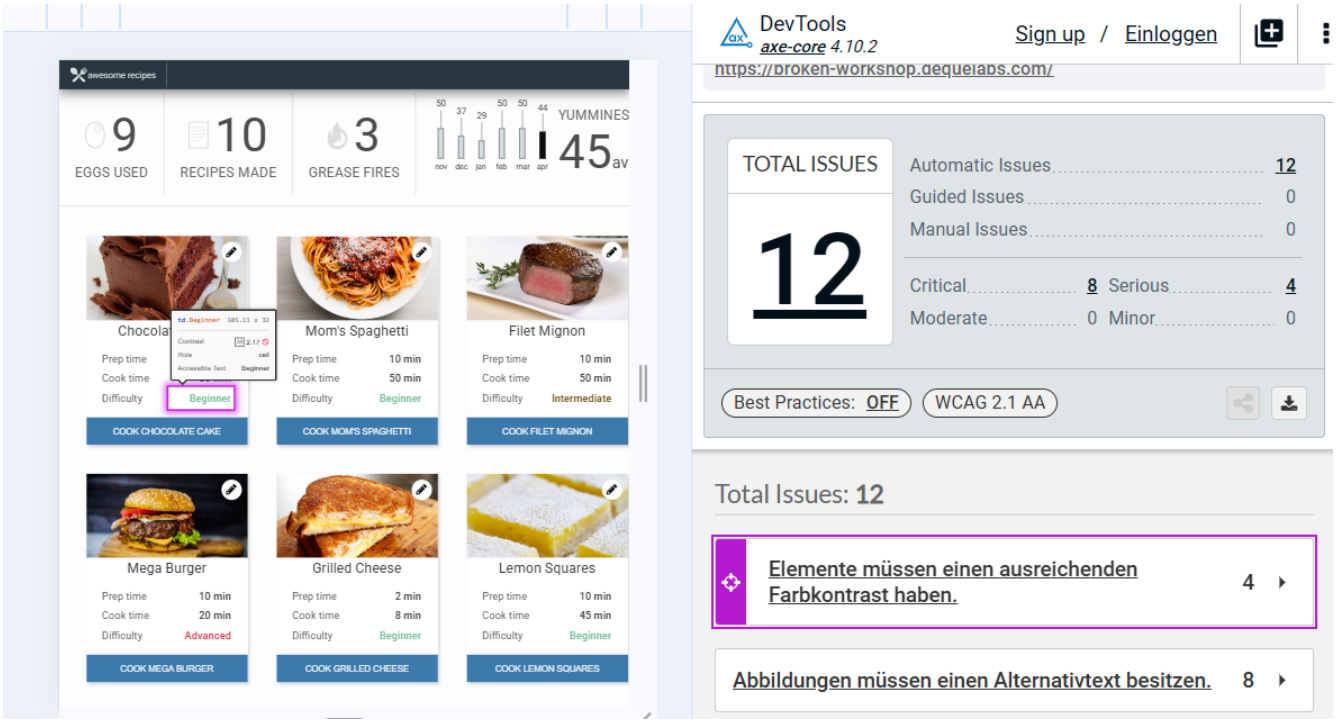


Abbildung 7: Ergebnisse der Überprüfung einer Demoseite mit axe DevTools, nach Installation per Browser-Entwicklertools auszuführen

Quelle: axe DevTools, Demoseite gestellt von deque

Die Ausführung eines axe-core Testlaufs nach Einbindung des Skriptes liefert einen ausführlichen Report über Fehler, nicht anwendbare Tests und bei Bedarf auch bestandene Tests im JSON-Format zurück. Hierbei kann aus einer großen Anzahl an Sprachen ausgewählt werden, unter anderem auch die für QF-Test erforderliche deutsche Sprache. Wie auch die Browsererweiterung liefert das Skript Informationen über das Element sowie über die verletzten Regeln und Vorschläge zur Behebung der Fehler. Bei der Ausführung des Tests lassen sich Parameter definieren, die beispielsweise den getesteten Bereich der Seite oder die überprüften Regeln limitieren, oder Regeln, wie bei Tests mit HTML Iframes[1] interagiert werden soll. [7]

Mitarbeiter der Firma Deque Systems sind auch beteiligt an der Entwicklung von WCAG-Richtlinien [45] und ACT-Regeln [36], weshalb sich axe mit den eigenen "axe-Rules" stark an diesen Standards orientiert. Die offizielle Browser-Extension Axe DevTools (semi-automatisch) deckt 29, die automatische axe-core Engine 26 der 34 akzeptierten ACT-Regeln ab [39, Stand 5. Januar 2025, Version 4.8.3]. Allerdings sind die Angaben auf der ACT-Implementationsseite der W3C zur Zeit des Aufrufs etwas veraltet, da axe-core (Stand Dezember 2024) inzwischen in Version 4.10.2 verfügbar ist.

4.3 Vergleich und Auswahl für die Implementation in QF-Test

Für die Auswahl des Tools, das in die Software QF-Test integriert werden soll, wurden verschiedene Kriterien festgelegt:

- **Kompatibilität & Einbindung** - Das Tool soll möglichst einfach in QF-Test und in darin erstellte Web-Tests eingebunden werden können.
- **Funktionalität & Abdeckung** - Das Tool soll möglichst viele Accessibility-Fehler automatisch korrekt erkennen und Kunden möglichst gut bei der Einhaltung von rechtlichen Rahmenbedingungen unterstützen.
- **Lizensierung & Kosten** - Es ist ein Tool mit möglichst geringen Kosten und unproblematischer Lizensierung zu bevorzugen.

[1] HTML inner frames to display web pages within web pages

- **Flexibilität & Anpassbarkeit** - Das Tool soll möglichst leicht an unterschiedliche Anforderungen in Tests angepasst werden können.
- **Zukunftsfähigkeit & Support** - Das Tool soll vom Anbieter möglichst gut unterstützt und weiterentwickelt werden.

Diese Kriterien führen dazu, dass von den bisher beschriebenen Tools WAVE ausgeschlossen wird. Die für die Einbindung in QF-Test relevante Nutzung von API-Aufrufen (insbesondere die Installation der WAVE Standalone Engine, die für offline-Tests benötigt wird) ist kostenpflichtig (*Lizensierung & Kosten*). WAVE ist nicht Open Source und bietet wenige Möglichkeiten (nur wenige Queryparameter[1]) den Testaufruf anzupassen (*Flexibilität & Anpassbarkeit*).

Werden die übrigen der obig beschriebenen Testengines auf die genannten Kriterien überprüft, ergibt sich folgendes Bild:

Tools	**Alfa**	**axe-core**
Kompatibilität & Einbindung	Integration in Projekt benötigt, damit Tests durchgeführt werden können (ausgelegt für Entwickler)	axe.js in Seite und eventuell vorhandene iframes injecten (keine Voraussetzungen)
Funktionalität & Abdeckung	W3C: 24/34 ACT-Rules abgedeckt Manueller Vergleich Demoseite: 12/12 Accessibility Fehler gefunden nach WCAG 2.1 Level A/AA	W3C: 26/34 ACT-Rules abgedeckt Manueller Vergleich Demoseite: 12/12 Accessibility Fehler gefunden nach WCAG 2.1 Level A/AA
Lizensierung & Kosten	MIT-License (keine Einschränkungen) kostenlos	MPL-License (keine Einschränkungen) verwendete Third-Party Software MIT/ISC-License (keine Einschränkungen) kostenlos
Flexibilität & Anpassbarkeit	Testaufrufe anpassbar nur englische Testergebnisse Open-Source, Mitentwickeln möglich	Testaufrufe anpassbar mehrsprachige Testergebnisse Open-Source, Mitentwickeln möglich
Zukunftsfähigkeit & Support	aktiv weiterentwickelt durch SiteImprove 2024: von Version 0.72.0 zu 0.97.0	aktiv weiterentwickelt durch Deque Systems 2024: von Version zu 4.8.4 zu 4.10.2

Tabelle 2: Die Testengines Alfa und axe im Vergleich auf wichtige Auswahlkriterien

Beide Tools bieten eine gute Testabdeckung, sind kostenlos und werden aktiv weiterentwickelt. Allerdings ist die Einbindung und Bedienung von axe-core einfacher und erfordert nur Zugriff auf das zu testende HTML-Dokument. Insbesondere die mitgelieferte deutsche Übersetzung spart im Entwicklungsprozess viel Zeit, die ansonsten mit umständlicher Übersetzungs- und Schreibarbeit verloren gegangen wäre.

Somit fiel die Entscheidung für dieses Projekt auf axe-core.

Zusätzliche Anmerkung:
Neben den oben beschriebenen Tools gibt es natürlich noch weitere Möglichkeiten, wie etwa Total Validator[2] oder qualweb[3], welche aber aus Gründen der oben genannten Kriterien als Option verworfen wurden.

[1] WAVE API Dokumentation
[2] Total Validator
[3] qualweb

5 Implementierung der automatisierten Tests

5.1 Testtool QF-Test

QF-Test[1] ist eine Software für automatische Benutzeroberflächentests (*GUI-Tests*). In der Software können automatische Tests mit verschiedenen Hilfsmitteln definiert und abgespielt werden. Hierbei kann die Software auch per Kommandozeile im Batchmodus gestartet werden, um Tests auf Testmaschinen ohne interaktive Benutzeroberfläche abzuspielen [24, Kapitel 24.1: Testausführung im Batchmodus]. Nutzerinteraktionen können aufgezeichnet und wieder abgespielt werden, aber dieser "Capture-Replay"-Ansatz ist nicht die einzige Art und Weise, um Tests zu definieren. Manipulationen an dem GUI und weitere Interaktionen mit dem System Under Test (SUT), wie HTTP-Requests, können manuell definiert werden. Testschritte können als Prozeduren zusammengefasst und zur einfachen Wiederverwendung bereitgestellt werden, Vorbedingungen für Tests und Aufräumarbeiten nach Tests können automatisch abgehandelt werden. [24]

Skripte, die bei anderen Testprogrammen häufig die Tests steuern, nehmen in QF-Test eine andere Rolle ein. Tests können komplett ohne Skripte - also effektiv ohne Code - nur über QF-Test Elemente definiert sein. Dieser "Low-Code"[2]-Ansatz für die Testerstellung erlaubt es auch Fachtestern ohne Programmierkenntnisse umfassende automatische Überprüfungen zu definieren. Allerdings können Skripte zu verschiedensten Anwendungsfällen bei Bedarf in die Tests eingebunden werden. [24, Kapitel 11: Skripting]

Ein Beispiel zur Veranschaulichung: Um in QF-Test über verschiedene Webseiten zu iterieren und Tests auszuführen, kann ein QF-Test interner Datentreiber erstellt werden. Dieser beinhaltet beispielsweise eine Datentabelle mit URLs, die zu den zu testenden Seiten führen. Wie in einer Schleife wird jetzt über die Einträge der Datentabelle iteriert und die angegebenen Prozeduren werden ausgeführt. Weitere Ablaufsteuerungen, wie If-Else-Abfragen oder Try-Catch-Blöcke, können auch innerhalb einer Testsuite mit entsprechenden Knoten verwirklicht werden - hierfür sind keine Skripte erforderlich. Allerdings lässt sich die Funktionalität von QF-Test per Skript erweitern. In dem folgenden Screenshot der Beispieltestsuite wird etwa ein Skript verwendet, um bestimmte Testergebnisse zusätzlich zum QF-Test Protokoll auch noch in eine .csv-Datei zu loggen.

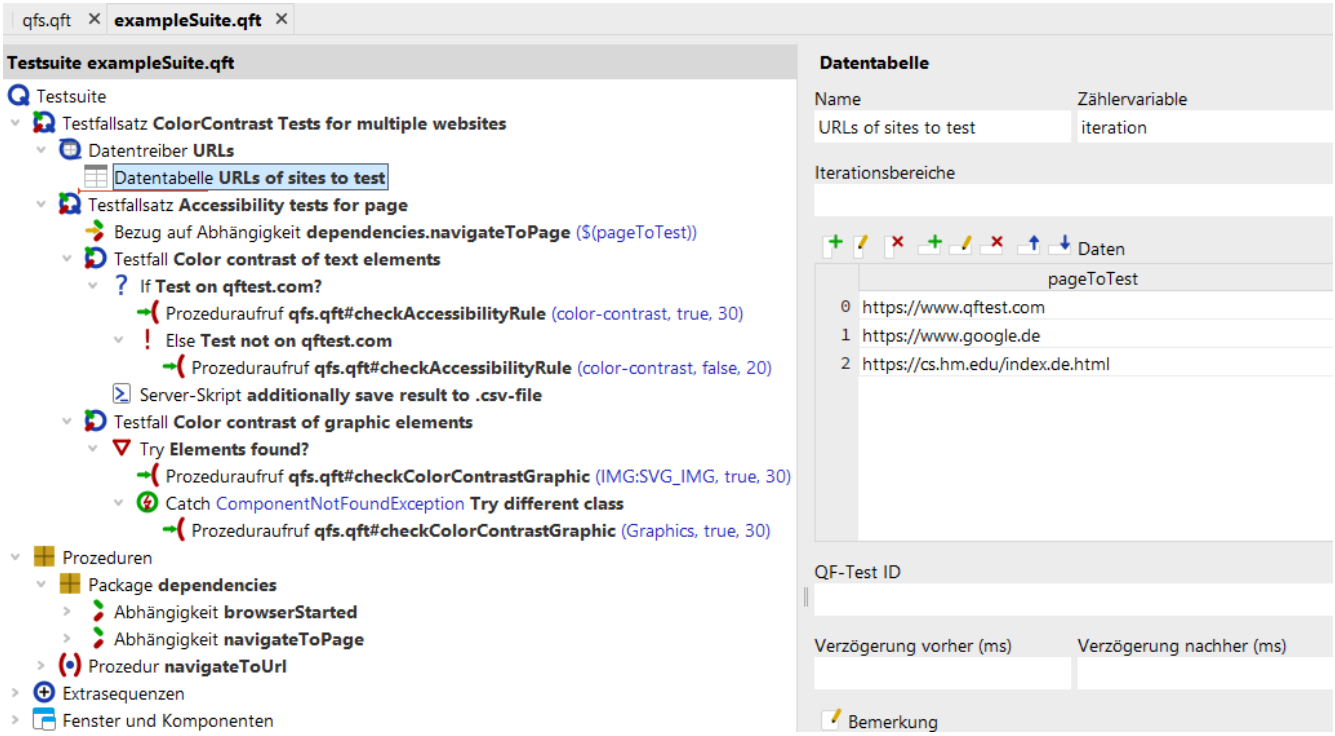


Abbildung 8: Testsuite in QF-Test zum obig beschriebenen Beispiel

[1] QF-Test Homepage
[2] ibm Artikel zu Low-Code

Wichtig für automatisierte Tests ist die Komponenten(wieder)erkennung. Aktionen müssen auf bestimmten Komponenten (Elementen) des UI ausgeführt werden können (beispielsweise ein Mausklick auf einem Button). Außerdem müssen Komponenten auf Inhalte, Aussehen, Vorhandensein und andere Eigenschaften überprüft werden können. Die Komponentenerkennung muss robust sein, denn im Verlauf eines Tests kann es zu Änderungen im SUT und somit auch an den Zielkomponenten kommen [24, Kapitel 5: Komponenten]. Hierfür gibt es in QF-Test zwei übergeordnete Methoden:

- **Explizite Aufnahme der Komponenten**: QF-Test nimmt bei der Aufnahme von Aktionen oder auch explizit die Struktur des SUT auf. Die dabei erkannten Komponenten werden mit nötigen Identifikationsmerkmalen in einer sinnvollen Struktur abgespeichert. Dabei erhält die Komponente eine ID, welche für Aktionen und Überprüfungen dieses Elementes referenziert werden kann.
- **SmartID**: Die Referenzierung einer Komponente kann mittels SmartID auch ohne Aufnahme erfolgen. Wird das Element benötigt, kann es direkt angesprochen werden, indem eindeutige Identifikationsmerkmale selbst in die ID aufgenommen werden. Beispielsweise muss ein Button mit einem gesetzten "name"-Attribut nicht mehr aufgenommen werden, sondern kann mit der SmartID `#Button:name=nameOfTheButton` zuverlässig erkannt werden.

Da die zu implementierenden Barrierefreiheitstests den Nutzern bestmöglich bei der Fehlerbehebung helfen sollen, ist es wichtig, fehlerhafte Komponenten möglichst leicht auffindbar zu machen. Um Komponenten innerhalb QF-Tests ansprechbar zu machen, bietet sich beispielsweise die SmartID an. Auch weitere Informationen, wie Screenshots oder Identifikatoren wie XPath oder CSS-Selektoren, sind hilfreich bei der Fehlerbehebung.

Diese Informationen müssen in das Protokoll von QF-Test aufgenommen werden, in dem sich alle Informationen zur Testausführung finden lassen. Fehler, Warnungen, Bildschirmabbilder und viele Informationen werden dort gelistet. Bei Bedarf können, beispielsweise per Skripting, noch weitere Informationen über den Systemzustand oder das SUT geloggt werden. Für die Verwendung der Testergebnisse außerhalb der QF-Test-Anwendung lassen sich auch noch HTML-Reports erzeugen, in denen vor allem Fehler, aber - wenn entsprechend festgelegt - auch beliebige Informationen über den Testlauf protokolliert werden können. Eine Herausforderung dieser Arbeit ist es, diese Testberichte für den Nutzer mit sinnvollen und ansehnlichen Informationen zu befüllen.

Ursprünglich für Java Swing Oberflächen entwickelt, unterstützt QF-Test heute viele unterschiedliche Technologien (Engines). Desktopapplikationen, die mit Electron, SWT, JavaFX oder Windows-Technologien, wie etwa .NET-Anwendungen, können ebenso überprüft werden wie PDFs, mobile Anwendungen (Android und iOS) und Webanwendungen. [24],[25]

Während langfristig Barrierefreiheitstests auch auf anderen Plattformen geplant sind, ist für diese Arbeit nur die Web-Engine relevant.

5.1.1 QF-Tests Web Engine

QF-Test prüft Webseiten aus Anwendersicht. Das bedeutet, dass die Webseite nicht in reiner Codeform, sondern in einem Browser geladen betrachtet wird. Hierbei unterstützt QF-Test die gängigen Browser, teilweise auch im Headless-Modus[1]. Die Überprüfungen können per Emulation eines mobilen Browsers auch aus der Sicht von Nutzern mobiler Geräte (wie Smartphones, Tablets...) durchgeführt werden. [24],[25]

Die Kommunikation zwischen QF-Test und dem Browser kann auf verschiedenen Wegen durch sogenannten "Driver" erfolgen:

- **QF-Driver**: Der Browser wird in ein Wrapper-Fenster eingebunden und über die Automatisierungsschnittstellen des jeweiligen Browsers kontrolliert.
- **WebDriver**: Der Browser wird über Schnittstellen des WebDriver[2] Standard der W3C (basierend auf dem Selenium WebDriver) gesteuert.
- **CDP-Driver**: Chromium basierte Browser (Chrome, Edge, Opera...) können über die **C**hrome **D**evTools **P**rotocol[3] Schnittstelle, die auch in den Entwicklertools von Chrome genutzt wird, gesteuert werden.

[1] Headless-Mode Beschreibung, hier das Beispiel Google Chrome
[2] W3C WebDriver Standard
[3] Chrome DevTools Protocol

Für die Implementierung verursachen diese unterschiedlichen Driver allerdings weniger Unterschiede. Der CDP-Driver erlaubt einen etwas tieferen Zugriff auf die Seitenstruktur, was sich in leichten Vorteilen bei der Überprüfung von shadow DOMs niederschlägt (siehe nächstes Kapitel). Für die Erzeugung von Bildschirmabbildern werden bestehende Schnittstellen angesteuert, wobei die Genauigkeit bezüglich ermittelter Koordinaten und Geometrie bei dem QF-Driver am niedrigsten ist. Insgesamt ist im Endprodukt die Verwendung des CDP-Drivers zu bevorzugen - oder der WebDriver bei nicht Chromium basierten Browsern wie Safari oder Firefox.

5.2 Einbindung der axe-core Library

Wie in Kapitel 4.3 beschrieben, fiel die Wahl für eine externe Web-Accessibility Bibliothek auf axe-core.

Nach dem Klonen des Repositories kann axe als JavaScript-Datei (`axe.js`) per `npm run build` erzeugt werden. Damit axe-Tests auf einer Website gestartet werden können, muss die erzeugte `axe.js` durch QF-Test in den JavaScript-Kontext eingebunden werden. Hierbei wird bei Bedarf noch die deutsche Übersetzung (`axe-de.json`) des Outputs geladen oder, wenn nötig, entfernt.

An dieser Stelle muss Rücksicht auf spezielle Strukturen der Website genommen werden. Die Testbibliothek muss auch in IFrames (eingebettete Dokumente im aktuellen HTML-Dokument) bereitgestellt werden. Bei inneinander verschachtelten IFrames wird axe rekursiv bis zum niedrigsten IFrame eingebunden.
Eine weitere Herausforderung stellen sogenannte "shadow DOMs"[1] dar, die ähnlich wie IFrames Elemente im DOM einbetten kann. Shadow DOMs eignen sich für benutzerdefinierte Elemente (*custom widgets*), die in unterschiedlichen Seiten eingebunden werden sollen, dabei aber nicht vom Haupt-DOM von Effekten wie Styling betroffen sein sollen. Solche Elemente, beispielsweise ein Video-Player, müssen natürlich auch auf Zugänglichkeit überprüft werden können. Aber aus der Sicht des Haupt-DOMs sind die einzelnen Unterelemente, die einen Video-Player ausmachen (Buttons, Slider etc.) versteckt. Somit muss axe auch in den Root-Knoten eines jeden shadow DOMs injeziert werden, um diese Elemente überprüfen zu können. Hierbei kann QF-Test bei der Verwendung des CDP-Drivers auch eine Methode umgehen, die genutzt von Entwicklerseite genutzt werden kann, um JavaScript den Zugriff auf diese shadow DOMs zu verweigern (`mode: "closed"`).

Nun ist die axe-API verfügbar und kann von QF-Test durch `eval(...)`-Aufrufe angesteuert werden, etwa `axe.configure(...)` zum Modifizierung von Prüfungen oder Einbindung eigener Regeln, oder einen Überblick über die verfügbaren Regeln mit `axe.getRules(...)` [7]. Der wichtigste Aufruf ist natürlich der Befehl zur Testausführung: `axe.run(...)`.

5.2.1 Testausführung mit axe-core

Axe bietet verschiedene Parameter, die Auswirkungen auf die zu überprüfenden Regeln, den zu überprüfenden Bereich (*scope*) der Seite oder auf die zurückgelieferten Resultate haben. Diese sind in der axe-API Dokumentation [7] gelistet. Nicht alle Parameter sind für QF-Test Testläufe relevant - manche, wie etwa spezifische Modifikationen am zurückgelieferten JSON-Report, können sogar die Einbindung in die QF-Test Protokolle und Reports unnötig erschweren. Zusätzlich können bei vielen Werten sinnvolle Default-Werte gesetzt werden (und als Konsequenz unsinnige Werteingaben eines Nutzers vermieden werden). Zusammengenommen führt dies dazu, dass der Nutzer nur die wichtigsten Einstellungen setzen kann. Damit wird unnötige Komplexität vermieden.

In der folgenden Tabelle sind die wichtigsten Parameter einer axe-Testausführung zu finden - sowie default-Werte und die Entscheidung, ob die Parameter im Rahmen von QF-Test für Kunden direkt einstellbar sind:

[1] Shadow DOM

Parameter	Funktion	default	In QF-Test
`context`	Einschränken der zu überprüfenden Elemente	-	`scope, default: gesamte Seite` Angabe einer Komponente per QF-Test ID oder SmartID Checks werden nur innerhalb dieser Komponente ausgeführt
`runOnly`	Einschränken der zu überprüfenden Regeln per Name oder Tag	-	`rules, default: WCAG Regeln` selbe Funktionalität wie in axe
`rules`	Einzelne Regeln können hier extra enabled/disabled werden	-	`rulesToSkip, default: -` ausschalten einzelner Regeln

`resultTypes`	Limitiert Resultate nach Ergebnis
	
	violations: Fehler
	incomplete: Warnung, muss manuell überprüft werden da der Check kein eindeutiges Ergebnis liefern konnte
	inapplicable: Keine Elemente vorhanden auf welche diese Regeln zutreffen
	successful: erfolgreiche Checks
		-	`showImpossibleChecks, default: true`
			listet die Elemente, die manuell überprüft werden müssen, als Warnung im Protokoll/Report auf
			
			`showSuccessfulChecks`
			default: false
			um auch erfolgreiche Checks anzuzeigen

`selectors`	Überprüfte Elemente werden im Ergebnis mit CSS-Selector gelistet	true	wird in QF-Test immer geloggt
`xpath`	Überprüfte Elemente werden im Ergebnis mit XPath gelistet	false	wird in QF-Test immer geloggt
`iframes`	Legt fest, ob axe auch in IFrames läuft	true	wird in QF-Test immer ausgeführt

Tabelle 3: Umsetzung wichtiger axe-Testlaufparameter in QF-Test

Der Aufruf von axe wird somit in eine QF-Test Prozedur übersetzt. Einige Parameter dieser Prozedur legen das Verhalten von axe fest, andere bestimmen das Verhalten der Reportgenerierung.

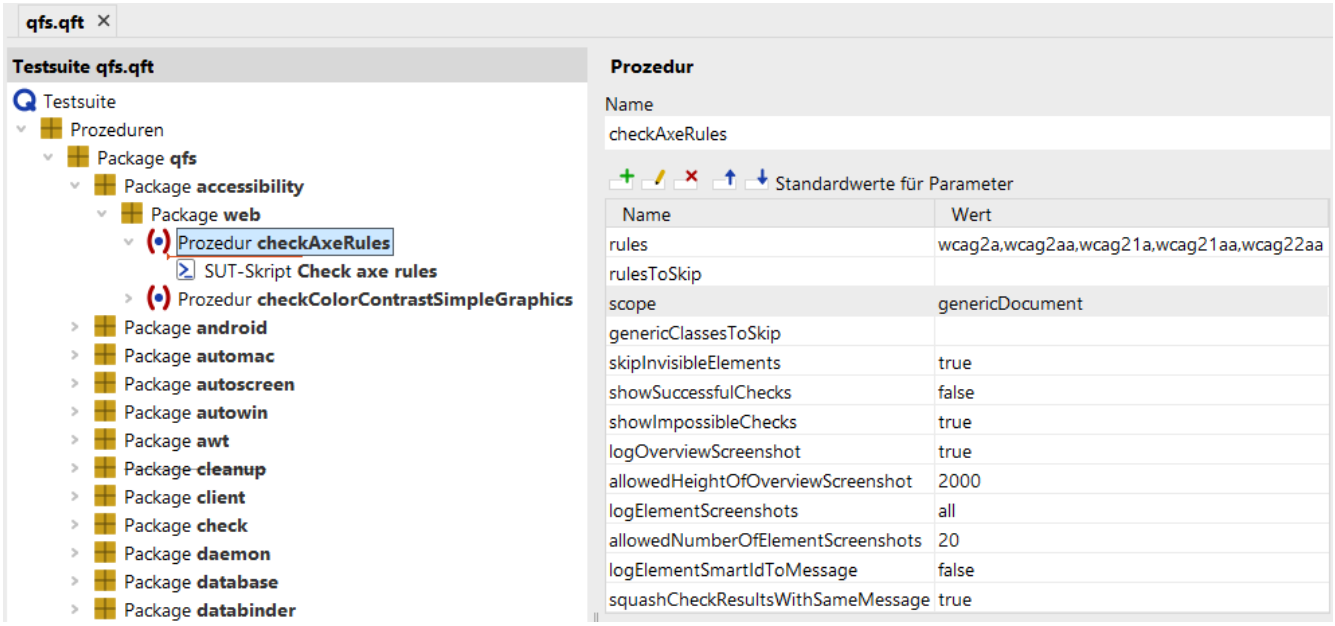


Abbildung 9: Prozedur zur Ausführung eines axe-Testlaufs in QF-Test mit default-Parameterwerten
Die Prozedur ist in der allgemeinen Standardbibliothek `qfs.qft` abgelegt und somit für alle Nutzer verwendbar
Hinweis: Der Wert `'genericDocument'` für den zu überprüfenden Abschnitt `scope` steht für die **gesamte** Seite

5.2.2 Reportgenerierung

Das Ergebnis eines axe-Testlaufs wird im JSON-Format zurückgeliefert. Dieses liefert neben generellen Informationen, wie etwa die verwendete axe-Version, die in obiger Tabelle beschriebenen vier `resultTypes`. Für Nutzer wichtige Informationen finden sich vor allem in `violations` und `incomplete`, die respektiv fehlgeschlagene Checks und mit zugehörigen Elementen und unvollständige, einer manuellen Überprüfung bedürfenden Checks beinhalten.

Diese Informationen sind allerdings mehrfach verschachtelt in der JSON Struktur des Resultates, wie in den folgenden UML-Diagrammen zu sehen. Mit Hilfe eines JSON-Parsers werden die relevanten Attribute und Werte aus dem Resultat in eine Java-Datenstruktur geladen. Diese bildet die `resultTypes` durch die Klasse `AxeResultType` ab, welche die einzelnen Regeln und auch die Daten zu den überprüften Elementen in Unterklassen abspeichert. Für das QF-Test Protokoll können die benötigten Daten nun einfach durch Iterieren über diese Datenstruktur gewonnen werden.

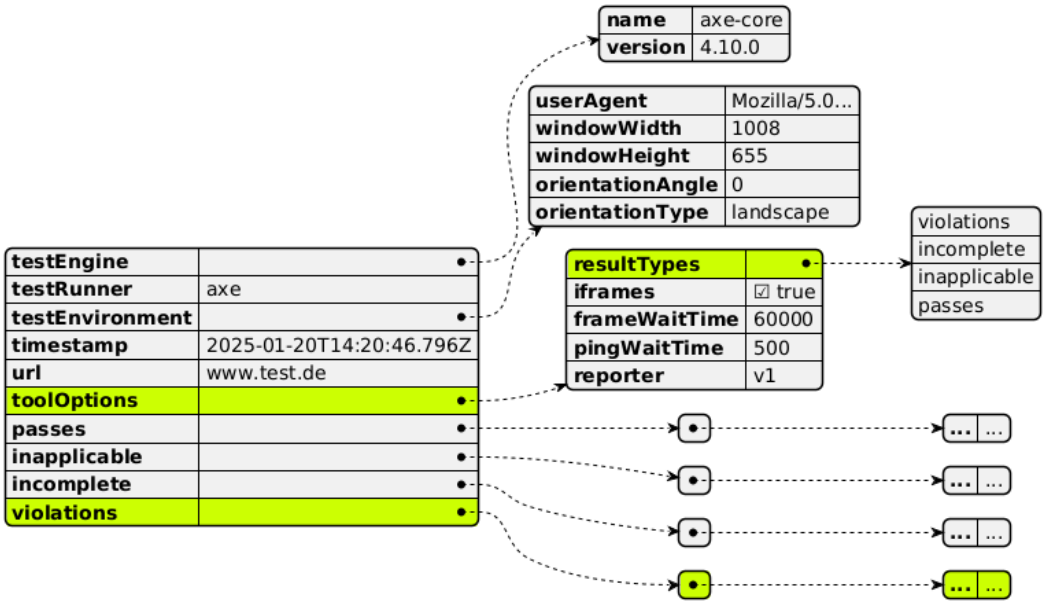


Abbildung 10: Aufbau eines axe-Resultates
In `resultTypes` werden die zurückgegebenen Arten der Resultate (Fehler, Erfolge ...) definiert.
Das `violations`-Array beinhaltet fehlerhafte Checks mit weiteren Informationen, siehe folgende Abbildung.

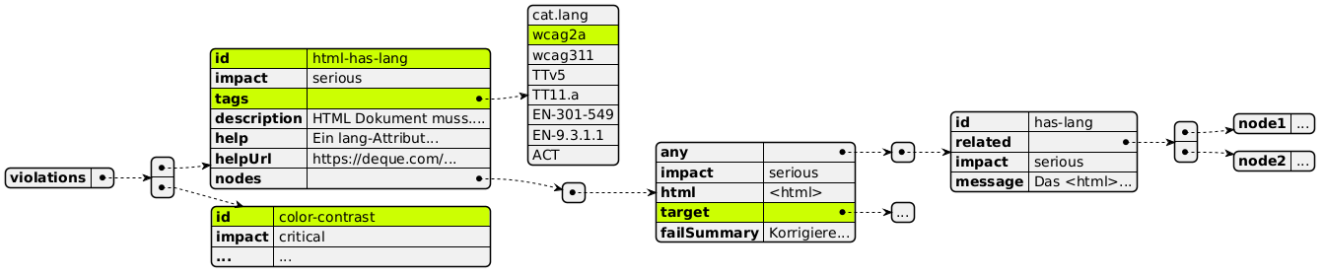


Abbildung 11: Beispiel des Aufbaus des `violations`-Arrays
Jeder Eintrag beschreibt eine Regel, definiert per `id` - die Regel besitzt mehrere Eigenschaften, wie etwa `tags`.
Der `tag wcag2a` bedeutet, dass diese Regel für die Erfüllung der WCAG 2.0 Level A eingehalten werden muss.
Unter `target` findet sich das fehlerhafte Element, je nach Einstellung per CSS-Selektor und/oder XPath aufgeführt.

Insbesondere wichtig für einen praktischen und ansehnlichen Testreport sind Bildschirmabbilder. Um die in der QF-Test Web Engine vorhandenen Mechanismen zum Erstellen von Screenshots einer Web-Komponente ansteuern zu können, muss das Element in ein von QF-Test anerkanntes Objekt umgewandelt werden.
Hierfür wird der von axe gelieferte CSS-Selektor verwendet. Dieser ist robuster gegenüber Veränderungen auf der Seite, die eventuell bei einem Testlauf auftreten können und andere Identifikatoren wie XPath wertlos machen können. Über die Verwendung von JavaScript kann per `querySelector()`[1]-Methode das zum CSS-Selektor zugehörige Element gefunden und in ein von QF-Test lesbares `Node`-Objekt übersetzt werden.

Zusätzlich zu Abbildern einzelner Elemente wird noch ein Überblick über den gesamten getesteten `scope` erstellt. Häufig sind die zu prüfenden Webseiten zu groß, um in nur einem Bildschirmabbild festgehalten werden zu können. In diesen Fällen müssen mehrere Screenshots erzeugt werden, zwischen denen die Webseite entsprechend "heruntergescrollt" werden muss. Danach wird das Gesamtabbild (*Overview-Screenshot*) aus den Teilabbildern zusammengebaut.

Fehlerhafte Elemente werden rot, Elemente, die einer manuellen Überprüfung bedürfen gelb umrandet. Dies ermöglicht es, die einzelnen Probleme schnell zu erkennen und in den Gesamtkontext einzuordnen.

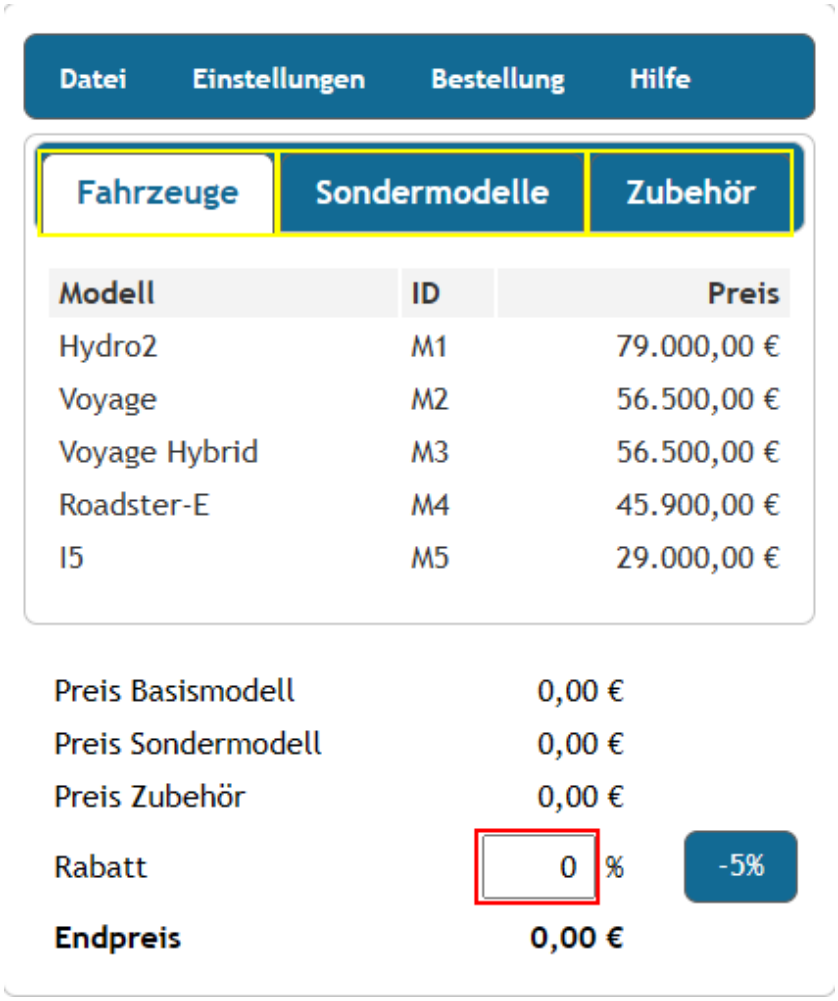


Abbildung 12: Beispiel eines Overview-Screenshots
Fehlerhafte Elemente sind farblich umrandet

Quelle: QF-Test Handbuch [24]

Mehrere Parameter der QF-Test Prozeduren zu Barrierefreiheitstests beziehen sich auf diese Screenshots und regeln deren Größe und Anzahl - und ob diese überhaupt erzeugt werden sollen. In erster Linie existieren diese Einstellungen, um den Speicherverbrauch zu kontrollieren. Bei umfangreichen Tests wird zwangsweise auch ein umfangreiches Protokoll kreiert. Accessibility-Fehler sind gerade am Anfang der Bemühungen, eine Seite barrierefrei zu gestalten, in großer Zahl vorhanden - und Bilddateien benötigen viel Speicher. Sofern der Nutzer nicht den Testumfang einschränken will, ist es empfehlenswert, die erzeugten Bilder über diese Parameter zu limitieren.

5.2.3 Protokoll und HTML-Report eines QF-Test Barrierefreiheitstests

Wie beschrieben erzeugt QF-Test bei Testläufen sowohl ein Protokoll zur Verwendung innerhalb der Software als auch einen HTML-Report, welcher ohne eine Lizenz oder Installation von QF-Test betrachtet werden kann. Diese Berichte sollen beim Auftreten eines Fehlers möglichst viele sinnvolle und hilfreiche Informationen enthalten.

[1] Document.querySelector()

Insbesondere relevant sind:

- Die Art des Fehlers und Wege, diesen zu beheben
- Die fehlerhafte Komponente und ausreichende Möglichkeiten, diese im SUT wiederzufinden

Auf der folgenden Seite sind ein in QF-Test geöffnetes Protokoll eines Barrierefreiheitstests und der in einem Browser geöffnete zugehörige HTML-Report zu sehen.
In beiden ist die Fehlerart an den zugehörigen Meldungen zu erkennen. Im Protokoll befinden sich die Fehlermeldung zu einem fehlerhaften Element über dem zugehörigen Screenshot, so denn gewollt. Die Fehlermeldung findet sich hier unter dem Punkt "A11y-Check fehlgeschlagen: label..." - die zugehörige Fehlermeldung ist dort zu finden und auf der Abbildung des Reports zu lesen:

```
A11y-Check fehlgeschlagen: label
Fehlercode: ERR_AXE-CORE_CHECKS
   impact critical: label
Element:
    URL: https://www.carconfig.de
   XPath: /html/body/div/div/table/tbody/tr[4]/td[2]/input
   SmartID: #INPUT:TEXT:name=DiscountValue_input
   Selector: "#DiscountValue_input"
Korrigiere mindestens einen der folgenden Punkte:
    Das <form>-Element besitzt kein implizites <label>-Element.
   Das <form>-Element besitzt kein explizites <label>.
   Es existiert kein aria-label-Attribut oder das Attribut ist leer.
   ...
```

Diese Fehlermeldung liefert mehrere Informationen. Mit `label` ist bei axe-Barrierefreiheitstests die verletzte axe-Regel gemeint. Diese Regeln sind mitsamt Erklärung, Vorschlägen zur Fehlerbehebung, Tags und mehr Informationen in der offiziellen Dokumentation von axe-core nachzulesen.

Dort findet sich auch die in der Fehlermeldung ebenfalls beschriebene `impact`-Bewertung, die die axe-core Entwickler an die Regeln vergeben haben. Dieser Wert quantifiziert die Auswirkung eines Problems auf einen Benutzer mit Behinderung - von niedriger Priorität (*minor*) bis zu oberster Priorität (*critical*). Allerdings sind diese Werte nur zur Priorisierung bei der Behebung von Barrierefreiheitsproblemen zu nutzen - für die Erfüllung von WCAG-Richtlinien sind letztendlich **alle** Fehler zu beheben. [7],[24, Kapitel 19.2: Axe-Checks mit QF-Test]
Um in QF-Test zwischen den bereitgestellten Barrierefreiheitstests zu unterscheiden, wurde ein QF-Test Fehlercode definiert. Im Falle eines Tests mit axe lautet dieser `ERR_AXE-CORE_CHECKS`, die später beschriebene Farbkontrastüberprüfung besitzt beispielsweise den Fehlercode `ERR_COLOR_CONTRAST_SIMPLE_GRAPHICS`.

Der für die tatsächliche Fehlerbehebung wichtigste Abschnitt der Fehlermeldung beginnt nun mit den Identifikatoren des fehlerhaften Elementes (oder der fehlerhaften Elemente, wenn mehrere Komponenten den gleichen Fehler vorweisen).
Die `URL` der Seite, auf der sich die Komponente befindet, ist besonders bei Tests über mehrere unterschiedliche Seiten wichtig.
`XPath` und CSS-`Selector` bieten eine Möglichkeit, die Komponente mit Hilfe von Browser-Entwicklertools oder anderen Hilfsmitteln, wie etwa Selenium, zu finden.
Die `SmartID` kann optional ebenfalls geloggt werden. Damit lässt sich die Komponente innerhalb von QF-Test adressieren. Dies kann zum Debuggen oder für weitere speziellere Tests verwendet werden.

Zuletzt werden die von axe- oder QF-Test-Entwicklern beschriebenen Lösungsvorschläge genannt. Je nach Test kann es mehrere unterschiedliche Lösungen geben, beispielsweise kann ein fehlendes Label explizit mit einem `for`- oder `aria-label`-Attribut gesetzt werden, oder das beschriebene Element kann implizit von einem Label-Tag umschlossen werden.

Mit dieser Form der Testberichte sind sowohl der Fehler als auch die fehlerhafte Komponente einfach zu erkennen. Selbst ohne Vorwissen in der Arbeit mit QF-Test sollte der Umgang mit den Informationen aus Protokoll und Report möglich sein - bei Fragen gibt das Handbuch [24, Kapitel 19] oder der Kundensupport weitere Auskunft.

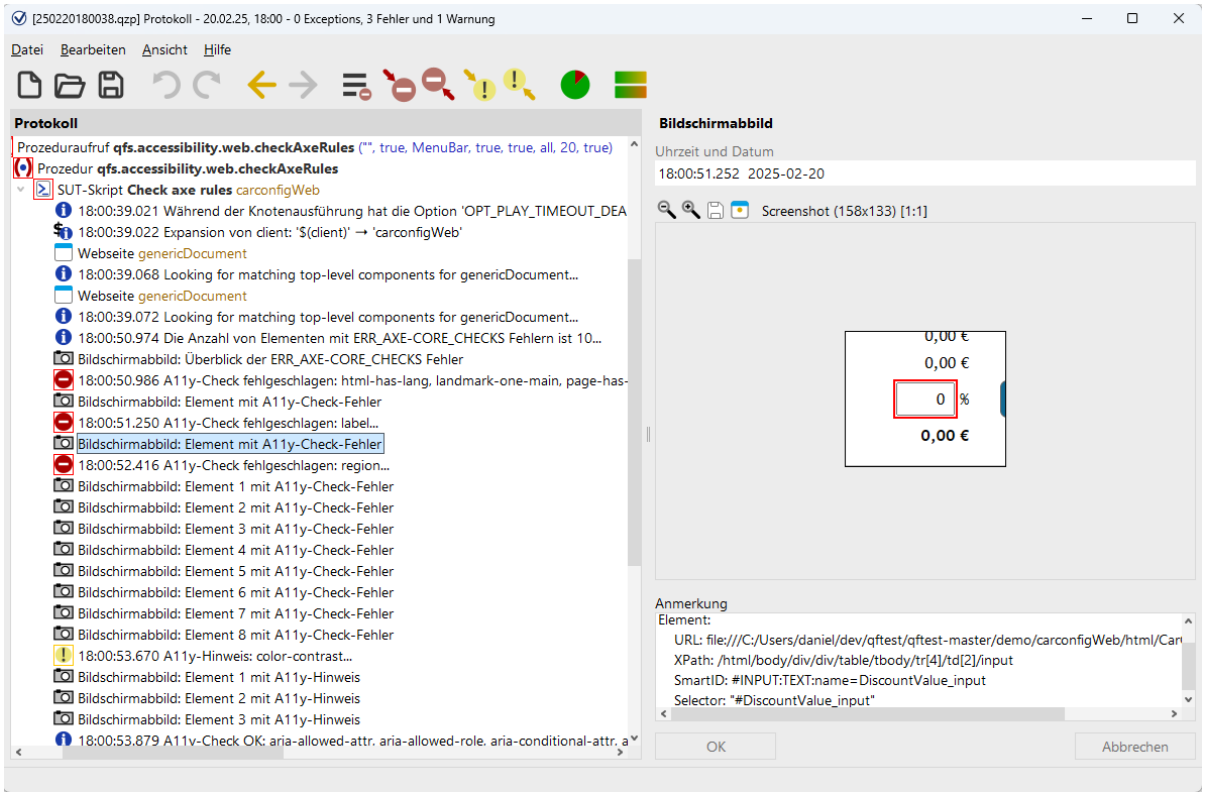


Abbildung 13: Ein QF-Test Protokoll eines Barrierefreiheitstests mit axe

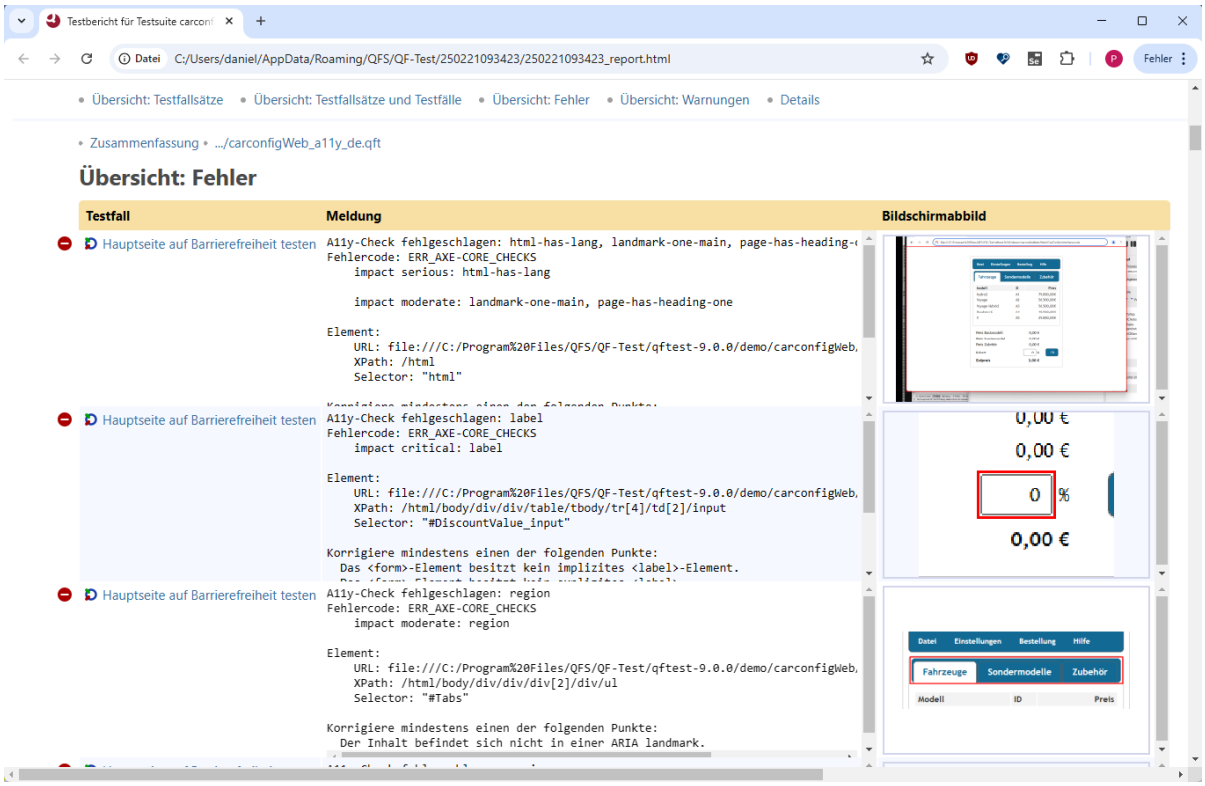


Abbildung 14: Ein QF-Test HTML-Report eines Barrierefreiheitstests mit axe

5.3 Farbkontrastüberprüfungen von Grafiken

Obwohl die Testberichte mit Bildschirmabbildern und der vereinfachte Zugang zur API schon eine signifikante Erweiterung der axe-core Engine darstellen, soll der Umfang der automatisierten Accessibility-Tests von QF-Test noch weiter ausfallen. Als ersten Schritt wurde beschlossen, erweiterte Farbkontrastüberprüfungen anzubieten. Zwar enthält axe-core mit dem `color-contrast`-Check bereits eine solche Überprüfung - diese ist aber nur auf Textelemente anwendbar und ermittelt dort den Kontrast zwischen Text- und Hintergrundfarbe.

Die in QF-Test implementierte Methode `checkColorContrastSimpleGraphics` hingegen kann auch (einfache) Grafikelemente, wie Icons oder Grafiken von Texten prüfen. Die Einschränkung "simple" besteht, da in dieser Methode nur in signifikanter Anzahl vorkommende Farben gegen eine automatisch ermittelte Hintergrundfarbe auf ihren Kontrast überprüft werden können. Ist allerdings auch der Unterschied zwischen diesen Farben zueinander wichtig, wie etwa bei einem Tortendiagramm, bei dem aneinander grenzende Farben ebenfalls kontrastreich sein müssen, kann der Algorithmus dies nicht überprüfen.

Die Funktionsweise des Algorithmus lässt sich so beschreiben: Valide Grafiken, also sichtbare Grafiken, die eine Minimalgröße überschreiten, werden gesammelt und einzeln überprüft. Zuerst wird eine eventuell vorhandene Rahmenfarbe ermittelt und die einzelnen Farbpixel in einem Histogramm gespeichert. Farben, die unter eine definierte Signifikanzschwelle fallen, also zu selten vorkommen, werden aus der Überprüfung ausgeschlossen. An einem beispielhaften Icon ausgeführt sieht das erzeugte Histogramm etwa so aus:

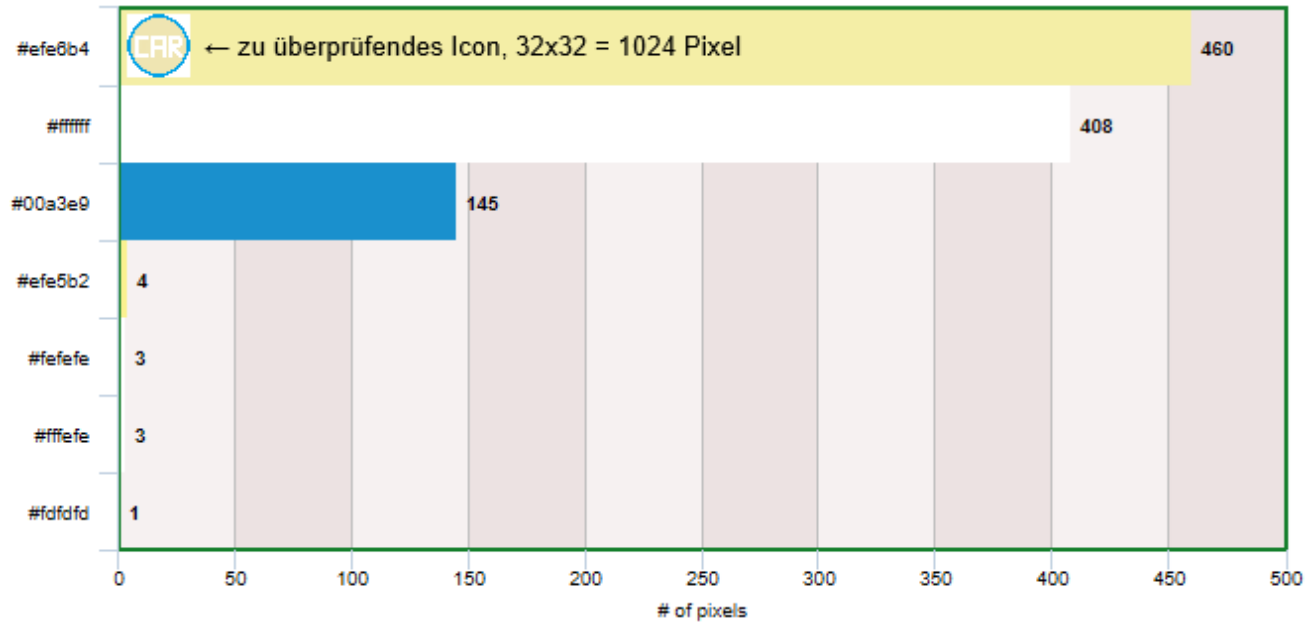


Abbildung 15: Beispiel: Farbhistogram eines 32x32 Pixel Icons mit den Hexcodes der ermittelten Farben
Die Farbe der Umrandung #00a3e9(blau) wird später bei der Kontrastprüfung nicht berücksichtigt.
Farben, die nur in insignifikant kleinen Mengen vorkommen (unter einer definierten Minimalanzahl),
werden bei der Überprüfung ebenfalls vernachlässigt.

Nachdem Randbedingungen kontrolliert wurden - wie etwa, dass keine Farbe als signifikant gewertet werden kann, oder, dass das Bild einfarbig ist - wird die Hintergrundfarbe (die am häufigsten vertretene Farbe, in unserem Beispiel hellgelb #efe6b4) ermittelt. Sollte diese sich von der Rahmenfarbe unterscheiden, wird die Rahmenfarbe aus der Überprüfung ausgeschlossen. Umrandungen sind nicht informationstragend und müssen daher keine Farbkontrastregeln erfüllen [45, WCAG §1.4.3].

Der Farbkontrast der verbleibenden Farben wird nun gemäß folgender Formel auf Farbkontrast gegenüber der Hintergrundfarbe geprüft:

$$Farbkontrast \ = \ \frac{L1 + 0.05}{L2 + 0.05} \tag{1}$$

Hier beschreibt L1 die Luminanz der helleren, L2 die Luminanz der dunkleren Farbe. Die Luminanz **L** is die "relative Helligkeit eines Punktes in einem Farbraum", von 0 (schwarz) bis 1 (weiß) [45, Appendix A] und lässt sich im sRGB-Farbraum [35] wie folgt berechnen:

$$\mathbf{L} = 0.2126 * \mathbf{R} \ + \ 0.7152 * \mathbf{G} \ + \ 0.0722 * \mathbf{B}, wobei \ \mathbf{R}, \ \mathbf{G}, \ \mathbf{B} \ definiert \ sind \ als: \tag{2}$$

$$\mathbf{R} = \begin{cases} \dfrac{R_{sRGB}}{12.92} & wenn \ R_{sRGB} \leq 0.03928, \\ \left(\dfrac{R_{sRGB} \ + \ 0.055}{1.055}\right)^{2.4} & sonst \end{cases} \tag{3}$$

$$\mathbf{G} = \begin{cases} \dfrac{G_{sRGB}}{12.92} & wenn \ G_{sRGB} \leq 0.03928, \\ \left(\dfrac{G_{sRGB} \ + \ 0.055}{1.055}\right)^{2.4} & sonst \end{cases} \tag{4}$$

$$\mathbf{B} = \begin{cases} \dfrac{B_{sRGB}}{12.92} & wenn \ B_{sRGB} \leq 0.03928, \\ \left(\dfrac{B_{sRGB} \ + \ 0.055}{1.055}\right)^{2.4} & sonst \end{cases} \tag{5}$$

Der Farbkontrast kann Werte im Bereich von 1:1 (kein Kontrast, gleiche Farbe) bis zu 21:1 (maximaler Kontrast, schwarz-weiß) annehmen. Im obigen Beispiel beträgt der errechnete Farbkontrast zwischen der Hintergrundfarbe #efe6b4 (hellgelb) und der verglichenen Schriftfarbe#ffffff (weiß) nur 1,26:1. Somit liegt er deutlich unter dem geforderten 4,5:1 Kontrast von Grafiken, die Text beinhalten, aber auch unter dem geforderten 3:1 Kontrast bei Bildern von großem Text oder Komponenten der Benutzeroberfläche [45, §1.4.3, §1.4.11].

Die Fehlermeldung wird im selben Format wie bei Überprüfungen mit der axe-Engine ins Protokoll geschrieben (siehe die Beispielmeldung aus vorherigem Kapitel sowie Protokoll und Report). Nur der Fehlercode unterscheidet sich (`ERR_COLOR_CONTRAST_SIMPLE_GRAPHICS`), und statt der ID einer axe-Regel (wie `label`, `html-has-lang`, `aria-allowed-attr` etc.) wird die eigene Regel-ID `color-contrast-simple-graphics` geloggt. Das Prozedere bezüglich Bildschirmabbilder bleibt zwischen den unterschiedlichen Barrierefreiheitstests gleich.

5.4 Usability und Nutzerinteraktion

Den einfachsten Einstieg in Barrierefreiheitstests bekommen Nutzer von QF-Test über den "Schnellstart-Assistenten" [24, Kapitel 3]. Mit dem Release der Accessibility-Test-Features findet sich hier nun eine Auswahlmöglichkeit für Barrierefreiheitstests. Zum Start kann die URL der getesteten Seite eingegeben werden. Danach können die unterschiedlichen Checks (axe, Farbkontrast) ausgewählt und die überprüften Elemente/Regeln eingestellt werden. Hier finden sich Vorschläge und Voreinstellungen, die Usern bei der Erstellung erster Tests Anregungen und Hinweise geben.

Dort wird, wie an vielen anderen Stellen auch, immer wieder auf das Handbuch verwiesen. Das Thema "Barrierefreiheitstests" erhielt darin ein ganzes Kapitel, in dem die Verwendung der Prozeduren, deren Parameter, der Umgang mit Protokollen und Reports und weitere Informationen zu finden sind.

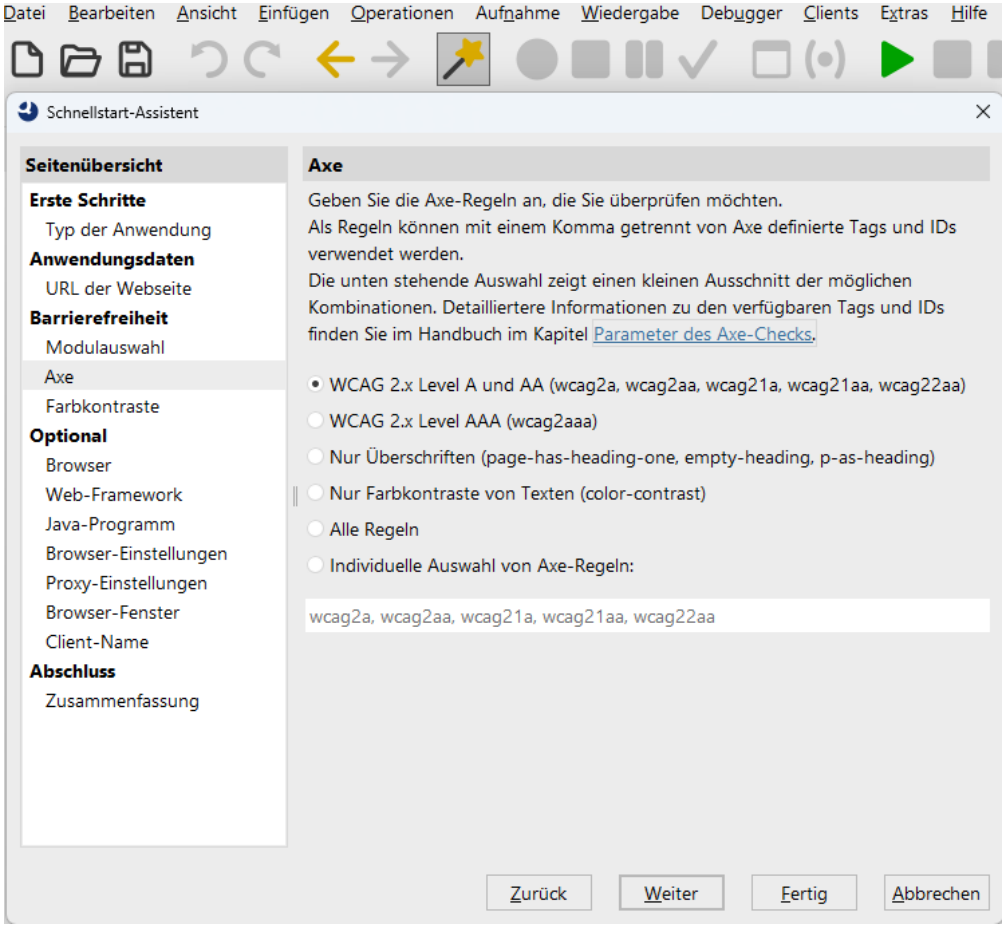


Abbildung 16: Einsatz des Schnellstart-Assistenten zum Erstellen einfacher Accessibility-Tests

Nach der Ausführung des Schnellstart-Assistenten wird, gemäß den ausgewählten Einstellungen, ein Testfall erzeugt. Dieser beinhaltet einen Vorbereitungsknoten. Nach dessen Ausführung wird das SUT gestartet und mit QF-Test verbunden. Nun können die Prozeduren der Barrierefreiheitstests aufgerufen werden und einen ersten Überblick über die Zugänglichkeit der Seite bieten.

Abbildung 17: Testfall mit Vorbereitung (Start der Website als SUT), axe- und Farbkontrast-Prozeduraufruf, automatisch erzeugt durch den Schnellstart-Assistenten

In weiteren explorativen Schritten kann der User selbst die Parameter, mit denen die Funktion aufgerufen wird, abändern. Diese sind nicht nur im Handbuch, sondern auch am Speicherort der Prozedur (der Standardbibliothek `qfs.qft`, über das Kontextmenü oder die Tastenkombination `Strg+P` leicht zu finden) ausführlich dokumentiert. Für die Einbindung in eigene Tests genügt es, den Prozeduraufruf zu kopieren und an die gewünschte Stelle einzufügen.

6 Fallstudie: ZPA der Hochschule München

In der "Erklärung zur Barrierefreiheit" der Hochschule München[1] heißt es: "Somit ist die Hochschule München bemüht, ihre im Einklang mit den nationalen Rechtsvorschriften [...] über den barrierefreien Zugang zu den Websites und mobilen Anwendungen öffentlicher barrierefrei zugänglich zu machen [sic]". Da der Webauftritt der Hochschule München sehr umfangreich ist, erfolgt die "technische Umsetzung der Barrierefreiheit [...] [zurzeit] sukzessive".

Die Verwendung der implementierten automatischen Barrierefreiheitstests soll nun an einem Teil dieses Webauftritts demonstriert werden. Hierfür wurde der Studentenbereich der ZPA-Seite der Fakultät für Mathematik und Informatik der Hochschule München ausgewählt. Die Seite wird von Studenten und Dozenten gleichermaßen genutzt. Das Onlineangebot regelt Kursanmeldungen, verwaltet Stundenpläne und stellt eine Plattform für Benachrichtigungen innerhalb der Fakultät und Informationen zu aktuellen Veranstaltungen bereit.

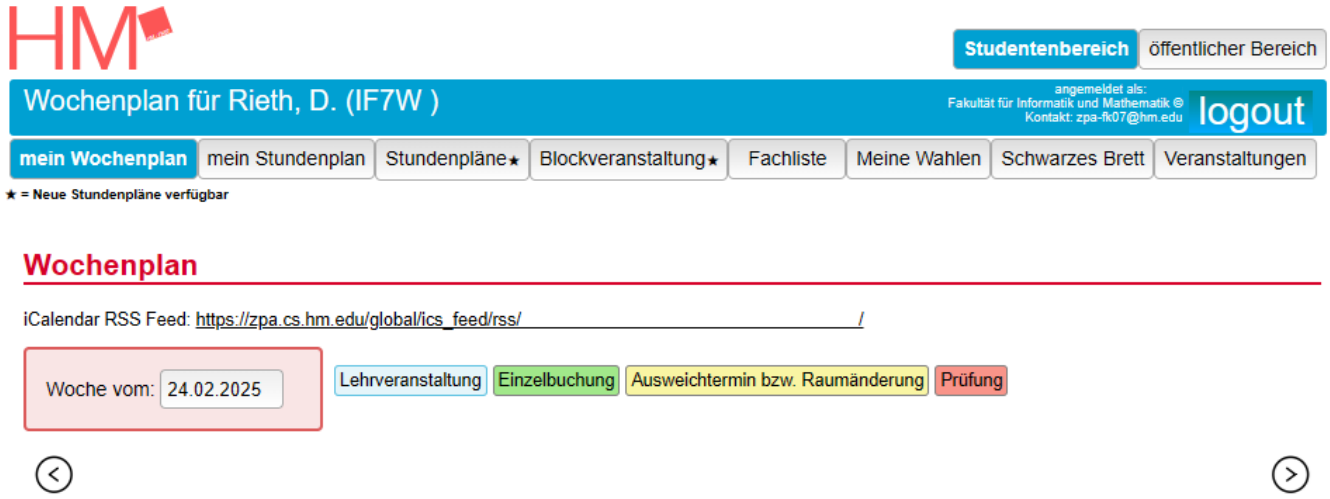


Abbildung 18: Überblick über den Studentenbereich des ZPA

6.1 Testaufbau

Per Schnellstart-Assistent wurde der Grundstein der Tests, also die Vorbereitung des SUT inklusive Start des Browsers, sowie die eigentlichen Prozeduraufrufe der Accessibility-Checks gelegt. Überprüft werden alle WCAG relevanten axe-Regeln sowie der Farbkontrast verwendeter `IMG` Elemente.

Für eine verbesserte Komponentenerkennung wurde der `CustomWebResolver`-Knoten in der Vorbereitung angepasst. Dies ermöglicht eine leichtere und gegenüber Änderungen robustere Navigation über die verschiedenen Reiter (*Tabs*) ("mein Wochenplan", "mein Stundenplan", etc. - siehe Abbildung).
Das Aufrufen der zu testenden Seite kann auf verschiedene Arten geschehen. In diesem Test werden die verfügbaren Tabs per einfachem Skript ermittelt und dann durch einen Mausklick ausgewählt. Beispielsweise könnte die nächste Seite aber auch aus einer Datentabelle [24, Kapitel 23: Datengetriebenes Testen] ausgelesen und per `Browser-Fenster öffnen` Knoten aufgerufen werden. Letztere Vorgehensweise ist bei einem größeren Testumfang, bei welchem die Navigation per Tastatur und Maus nicht mehr ausreichend ist, zu empfehlen.

Der Zugriff auf das ZPA erfordert gültige Zugangsdaten. Für diesen Test wurden die persönlichen Login-Daten des Autors verwendet. Um diese nicht zu veröffentlichen, wurde die Verschlüsselungsfunktion von QF-Test verwendet, sodass die Werte "username" und "password" in der mit dieser Arbeit abgegebene Testsuite nicht ohne den geheimen Schlüssel des Autors verwendbar sind. Um die Testsuite auszuführen benötigt es also eigene Zugangsdaten, die in der Testsuite ergänzt werden müssen. Der genaue Vorgang dazu und eine andere vorzunehmende Anpassung ist in den Kommentarknoten innerhalb der Testsuite beschrieben. Zudem wird QF-Test in der Version 9.0 oder höher sowie eine gültige Lizenz benötigt (dafür genügt eine gratis Testlizenz).

[1] Erklärung zur Barrierefreiheit der Hochschule München, Stand 24.02.2025

Der zusammengefasste Testaufbau:

- Stelle notwendige Vorbedingungen her (Seite aufgerufen, Login durchgeführt)
- Bestimme alle verfügbaren Tabs
- Iteriere über alle verfügbaren Tabs
  - Führe Sondereinstellungen für einzelne Tabs durch, beispielsweise die Angabe eines vergangenen Semesters, um einen gültigen Stundenplan angezeigt zu bekommen
  - Prüfe alle WCAG relevanten axe-Regeln im `content` des Tabs
- Prüfe die Farbkontraste von `IMG`s auf dem Tab "mein Wochenplan"

Der Farbkontrastcheck wird für diesen Test auf den ersten Tab beschränkt, da dort als einziges für Farbkontrastüberprüfungen relevante Bedienelemente zu finden sind - Knöpfe, mit denen die angezeigte Woche des Stundenplans geändert werden kann. Da die Farbkontraste im ZPA vorbildlich sind, wirft der Test an dieser Stelle keine Fehler. Zu Demonstrationszwecken wurde deshalb der Parameter `showSuccessfulChecks` auf `true` gesetzt. Dies sorgt dafür, dass auch diese erfolgreichen Tests im Protokoll und Report mit dem errechneten Farbkontrast sichtbar sind.

6.2 Ergebnisse

Die Überprüfung der 8 Tabs führte zu 12 von axe erkannten Accessibility-Fehlern. Zudem wurden zwei Fehler innerhalb des SUT von QF-Test als Warnungen protokolliert - diese sind allerdings im Kontext der Barrierefreiheit irrelevant.
Die Farbkontrastüberprüfung wurde fehlerfrei bestanden. Die Knöpfe zur Wochenauswahl besitzen den maximal möglichen Farbkontrast von 21:1. Sogar das zusätzlich überprüfte Logo der Hochschule München erreicht einen Wert von 3,2:1 - ist aber in [45, § 1.4.3: Kontrast (Minimum)] explizit von dem Erfolgskriterium der Richtline ausgeschlossen. Wortbildmarken (*Logotype*) müssen keinen minimalen Farbkontrast einhalten, sie benötigen lediglich einen Alternativtext.
Testsuite, Protokoll und Report wurden im Rahmen dieser Arbeit abgegeben.

6.2.1 Fehlende Beschriftungen: `label, select-name` (Anzahl: 4)

An mehreren Stellen auf der Seite finden sich Komponenten (wie Checkboxen, Dropdown-Listen oder Texteingabefelder), die nicht durch eine Beschriftung (*Label*) gekennzeichnet wurden. Dies führt dazu, dass etwa sehbehinderte Nutzer die Funktion eines solchen Elementes nicht bestimmen können [45, § 3.3.2: Labels].

Listing 1: Beispiel eines Elementes zur Datumsauswahl ohne *Label*

```
<input type="text" name="date" value="11.11.2023" onchange="submit_form(this);" id="id_date"
    style="width: 6em;" class="hasDatepicker">
```

Dieses Problem ist auf mehrere unterschiedliche Arten leicht zu beheben.
Mit Hilfe eines zusätzlichen `aria-label`-Attributes kann die ARIA-Schnittstelle genutzt werden, um eine für assistive Technologien nutzbare Beschriftung hinzuzufügen - `aria-label="Datumsauswahl"` wäre ausreichend, um dieses Element barrierefrei zu gestalten.
Ein *Label* kann auch ein sichtbarer Text sein, der das zu beschriftende Element entweder einbettet oder durch andere Attribute (`for, aria-labelledby`) mit dem Element verbunden wird.

6.2.2 Verschachtelte Kontrollelemente: `nested-interactive` (Anzahl: 7)

Listing 2: Ein fehlerhaftes verschachteltes Kontrollelement

```
<li role="tab" tabindex="0" class="ui-tabs-tab ui-corner-top ui-state-default ui-tab
    ui-tabs-active ui-state-active" aria-controls="tabs-1" aria-labelledby="ui-id-2"
    aria-selected="false" aria-expanded="false">
  <a href="#tabs-1" tabindex="-1" class="ui-tabs-anchor" id="ui-id-2">
    PBLV
  </a>
</li>
```

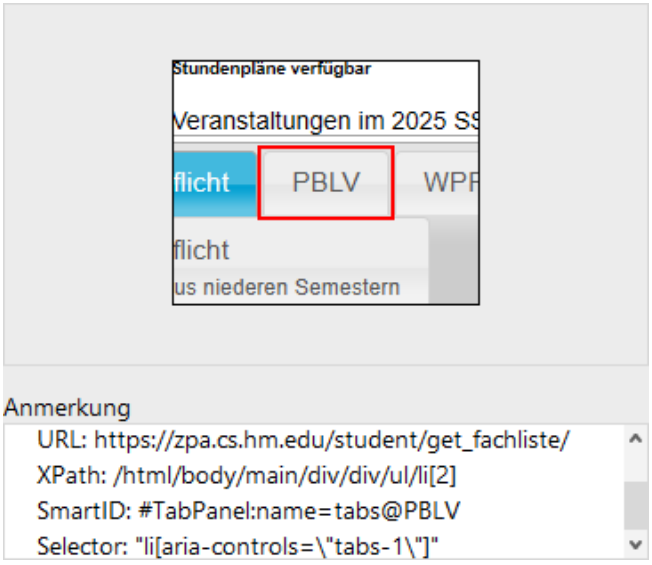


Abbildung 19: Beispiel eines `nested-interactive` Fehlers aus dem Testprotokoll

Die sieben Navigationselemente im "Fachliste"-Tab werfen bei dem Accessibility-Test jeweils den gleichen Fehler: `nested-interactive`, welcher die Verschachtelung interaktiver Elemente verbietet.
Wie bei Betrachtung eines solchen Elementes erkennbar ist, bestehen diese Kontrollelemente aus einem interaktiven `anchor`-Element, welches in ein `list`-Element mit der interaktiven Rolle `tab` eingebettet ist. Für die Navigation per Tastatur stellt dies kein Problem dar, per Tabulator und Pfeiltasten kann jedes Element erreicht werden. Doch Screen Reader können bei der Verwendung der Elemente auf Probleme stoßen.
Bei Verwendung der nativen Windows Screen Reader *Narrator* sind diese Probleme gut zu beobachten.
Zum einen wird, trotz Verwendung von `tabindex="-1"`, das innere `anchor`-Element durch den Screen Reader fokussiert. Somit wird das Element doppelt vorgelesen.
Zum anderen kann der *Narrator* beim Vorlesen einzelner Zeichen den Fokus nicht mehr über die Pfeiltasten aus dem fehlerhaften Element lösen. Dies ist bei einer korrekten Implementierung, wie sie etwa bei der obigen Navigation mittels Tabs zu finden ist, problemlos möglich.

An dieser Stelle soll erwähnt werden, dass nicht jeder Screen Reader Probleme mit dieser HTML-Syntax hat. Mit dem bereits erwähnten Tool NVDA treten etwa bei der Betrachtung derselben Seite keine Fehler auf.

Die WAI bietet auf ihrer Seite ein barrierefreies Beispiel für automatisch aktivierte Tabs an, welches als Lösung der Accessibility-Probleme implementiert werden kann.

6.2.3 Scrollbare Regionen ohne Fokus: `scrollable-region-focusable` (Anzahl: 1)

Der Wochenplan enthält einen personalisierten Stundenplan mit Einträgen zu regelmäßig stattfindenden Kursen, besonderen Veranstaltungen oder Terminänderungen. Allerdings kann es vorkommen, dass der Inhalt eines solchen Eintrags nicht so einfach per Tastatur erreichbar ist.

Listing 3: Zwei scrollbare Elemente, eines davon ohne fokussierbaren Inhalt

```
<!-- Fehlerhaft, enthält kein fokussierbares Element -->
<div class="slot_inner">
    Maschinelles Lernen (Ersatzraum) Wegen HOKO
    <br>
    GN IC IC5 IC7 IF / R3.017 / Spieler, D.
    <br>
</div>

<!-- Korrekt, enthält fokussierbare Elemente (anchor a) -->
<div class="slot_inner">
    <strong>
        <a target="ci_module" href="/public/module/317/">
            Maschinelles Lernen
        </a>
    </strong>
    <br>
    Praktikum, 2. Teilgruppe
    <br>
    GN IC IC5 IC7 IF / R1.005 / <a href="https://cs.hm.edu/kontakte_de/
        phonebook_detailseite_35969.de.html" target="_parent">Spieler, D. (FK07)</a>
    <div class="red_text">
        Findet in R3.017  statt. Wegen HOKO
    </div>
</div>
```

Obiger Auszug aus den entsprechenden Elementen zeigt, dass die fehlerhafte Komponente (Zeile 2-5) nur Text enthält. Anders als die korrekte Komponente (Zeile 8-21), welche mehrere *anchor*-Elemente (`<a>`) beinhaltet, die problemlos per Tastatur angesteuert werden können.
Wird mit der Tabulator-Taste über die Seite gesprungen, erhält das fehlerhafte Feld nie einen Fokus. Dieses Problem lässt sich einfach beheben, indem die Text enthaltenden `<div>`s grundsätzlich auswählbar gemacht werden - beispielsweise mit einem `tabindex`-Attribute mit dem Wert `"0"`.

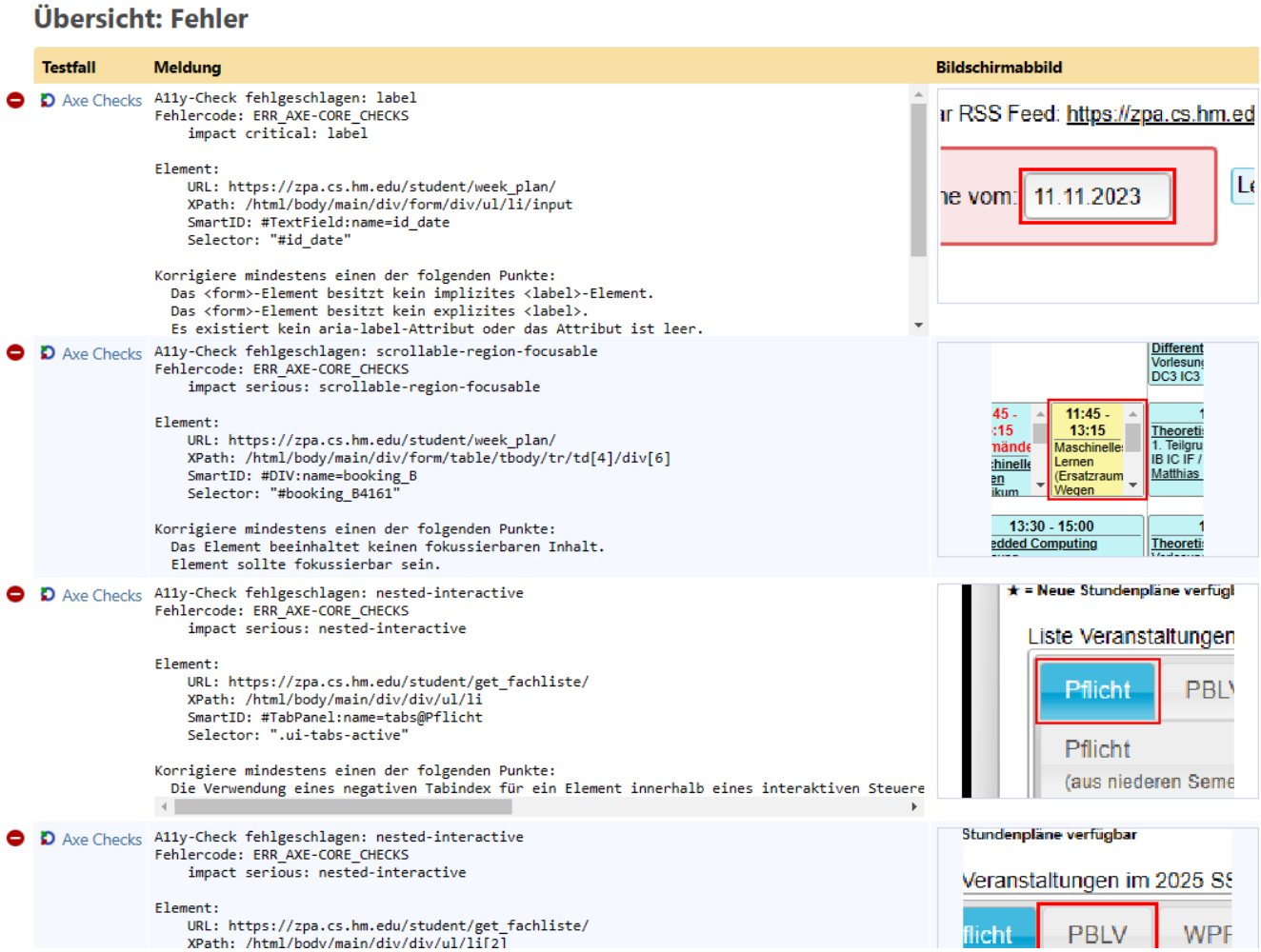


Abbildung 20: Ausschnitt aus dem HTML-Testreport des ZPA-Tests mit Fehlermeldungen, Deskriptoren des fehlerhaften Elementes und Lösungsvorschlägen.
Sowohl Protokoll als auch Report des Testlaufes vom 24.02.2025 wurden mit dieser Arbeit abgegeben.

7 Fazit und Ausblick

In dieser Arbeit wurde die Implementierung von automatisierten Barrierefreiheitstests in ein bestehendes Testtool beschrieben. Dabei wurde sowohl eine bestehende Test-Engine, als auch ein zusätzlicher Test in den Funktionsumfang der Software aufgenommen. Besonderer Wert wurde hier auf anschauliche und hilfreiche Testberichte gelegt, die für Nutzer einen großen Mehrwert bieten.
Im Rahmen der Auswahl des letztendlich eingebundenen automatischen Testtools wurden auch andere Alternativen präsentiert, die vielleicht in Zukunft noch eine Rolle in QF-Test spielen können. Zudem wurde ein Überblick über automatische und manuelle Testmethoden gegeben, sowie deren jeweilige Stärken und Schwächen gelistet. Letztendlich ist festzuhalten, dass die Zugänglichkeit digitaler Inhalte nur durch eine Kombination beider Arten von Tests gewährleistet werden kann.

Zudem wurden noch viele, im nicht-technischen Umfeld von Web-Accessibility wichtige Informationen zusammengetragen.
Insbesondere für Unternehmen sind die Kapitel zur Gesetzgebung bezüglich digitaler Barrierefreiheit wichtig. In der EU wirkende Firmen werden sich bei ihren digitalen Produkten bis Ende Juni 2025 an die Forderungen des EAA, umgesetzt auf Landesebene beispielsweise durch Gesetze wie das BFSG, halten müssen. Ansonsten drohen rechtliche Konsequenzen.
Explizit ist in diesem Gesetz die Einhaltung der WCAG-Richtlinien Pflicht, deren Aufbau und Inhalte dargelegt wurden. Diese Richtlinien stellen auch außerhalb der EU einen wichtigen Standard für Web-Accessibility dar.
Zuletzt - beziehungsweise im Aufbau dieser Arbeit zuerst - wurde auch noch auf die möglichen Einschränkungen eingegangen, die Nutzer im Umgang mit dem Internet oder Software betreffen können. Auch Hilfsmittel, wie etwa Screen Reader, die den Zugang zu digitalen Inhalten für diese Personengruppen erleichtern kann, wurden präsentiert.
Und natürlich wurde der Begriff "Barrierefreiheit" und dessen Bedeutung aufgezeigt. Die Ermöglichung der Teilhabe möglichst vieler an digitalen Inhalten ist nicht nur aus humanistischer, sondern auch aus wirtschaftlicher Sicht positiv zu betrachten - da ein barrierefreier Webauftritt eine breitere Zielgruppe an Menschen erreichen kann. Außerdem kann sich die Verbesserung der Barrierefreiheit auch auf nicht eingeschränkte Nutzer positiv auswirken, beispielsweise durch eine vereinfachte Navigation.

7.1 Planung Barrierefreiheitstestprojekt bei QF-Test

Für die Zukunft automatisierter Barrierefreiheitstests in QF-Test gibt es einige Ideen und Pläne, die nach dem Release der ersten Accessibility-Testmethoden diskutiert und eventuell verwirklicht werden.

Bei internen Accessibility-Tests sind bereits einige Prozeduren im Einsatz, die über den Funktionsumfang der axe-Bibliothek hinausgehen. Mit diesen werden verschiedene Zugänglichkeitskriterien überprüft, wie beispielsweise die Navigation [45, § 2.4.3 Fokus-Reihenfolge], die Sichtbarkeit des Fokus [45, § 2.4.7 Fokus sichtbar] oder die Kenntlichmachung des Zweckes von Links [45, § 2.4.4 Linkzweck]. In zukünftigen Versionen sollen diese Prozeduren auch für Kunden aufrufbar gemacht werden.

Da QF-Test nicht nur Weboberflächentests, sondern auch Tests auf vielen anderen Technologien anbietet, ist es ein offensichtlicher Schritt, Barrierefreiheitstests auch auf diesen Engines umzusetzen. Die Struktur des Codes zur Reportgenerierung sowie der Aufbau der Testberichte und Fehlermeldungen wurde hierfür extra generisch gehalten, um eine einfache Einbindung weiterer Tests und Engines, wie iOS, Android oder Java-Oberflächen zu ermöglichen. Der zusätzlich implementierte Farbkontrasttest benötigt lediglich unverdeckte Sicht auf das zu überprüfende Element, um ein Abbild zu erstellen, kann also ohne großen Aufwand verallgemeinert werden.
Die Anzahl und Qualität der Tests der eingebundenen axe-core-Bibliothek ist allerdings nicht ohne weiteres auf andere Oberflächen auszuweiten - zumindest nicht auf Oberflächen, die nicht HTML-basiert sind. Allerdings gibt es bereits Open-Source Projekte für Accessibility-Tests auf anderen Plattformen, wie etwa die Google-Projekte ATF[1] für Android oder GTXiLib[2] für iOS, die in QF-Test und das erarbeitete Testbericht-Framework eingebettet werden könnten.

[1] ATF - Acessibility Test Framework for Android
[2] GTXiLib - Google Toolbox for Accessibility for the iOS platform

Weitere mögliche Funktionalitäten, die unter anderem bei der Recherche für diese Arbeit in die Diskussion aufgenommen wurden, sind beispielsweise ein Farbenblind-Modus oder ein manueller Farbkontrastprüfer.
Da das zu überprüfende Programm in QF-Test eingehängt wird, kann beispielsweise per Manipulation der berechneten Anzeigefarbe in der `Window`-Schnittstelle eines DOMs oder durch ein Overlay-Fenster über dem SUT die Farbanzeige geändert werden. Somit wäre es möglich, das zu testende System aus der Sicht verschiedener Farbsehstörungen zu präsentieren, was für explorative, aber auch automatisierte Accessibility-Tests innerhalb QF-Tests nützlich sein kann.
Ebenso kann es von Vorteil sein, die Berechnung des Farbkontrastes in einem größeren Umfang als nur einer Prozedur bereitzustellen. Beispielsweise durch einen automatischen Check, der die Farben auf zwei per Koordinaten (innerhalb eines Elementes) definierten Bildpunkten auf ihren Kontrast überprüft - oder durch eine manuelle Auswahl der Farben durch einen Klick oder Eingabe des jeweiligen Farbcodes.

7.2 Ausblick: Künstliche Intelligenz

Zuletzt soll noch auf das aktuell sehr populäre Thema der Verwendung von künstlicher Intelligenz eingegangen werden. Der große Nachteil bei manuellen Tests, welche (wenn korrekt ausgeführt) die bestmögliche Zugänglichkeit von digitalen Inhalten garantieren können, ist der enorme Ressourcenaufwand. Fachtester müssen eine große Anzahl an unterschiedlichen Seiten abdecken, diese auf unterschiedliche Barrieren bei unterschiedlichen Anwendungsfällen überprüfen - und manchmal darüber hinaus noch Testpersonen beaufsichtigen, sowie deren Arbeit dokumentieren und evaluieren.

Eine offensichtliche Lösung für diese großen Kosten ist *Artificial Intelligence* (Künstliche Intelligenz) (AI). Tatsächlich findet künstliche Intelligenz bereits in einigen Testtools für Barrierefreiheit Verwendung [20]. Beispielsweise nutzt axe DevTools laut eigenen Angaben AI um Größen und Seitenverhältnisse zu überprüfen oder Verbesserungsvorschläge für ARIA-Rollen anzubieten. Laut Madu und Nzenwa verwendet auch WAVE AI [20].

Fuglerud et al. entwickelten in [13] mit Hilfe von maschinellem Lernen eine Möglichkeit, mehrere, mit anderen automatischen WCAG-Checkern nicht zuverlässig überprüfbare WCAG-Richtlinien automatisch testbar zu machen. Beispielsweise die Richtline [45, 1.1.1: Nicht-Text-Inhalt], bei der es um das Zutreffen der Beschreibung von Alternativtexten auf Bilder geht - also wie gut ein Alternativtext ein Bild beschreibt. Mit traditionellen Algorithmen ist dies nicht festzustellen, aber der von den Autoren der Studie entwickelte Prototyp konnte Alternativtexten vernünftige Werte zwischen 0 und 100% zuweisen, die das Zutreffen des Textes quantifizieren. [13]

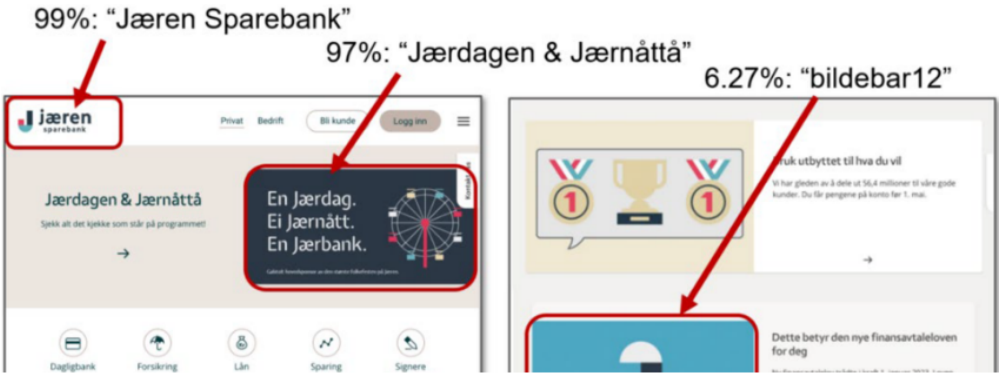


Abbildung 21: Einschätzung des Beschreibungsgrades eines Alternativtextes durch künstliche Intelligenz

Quelle: Fuglerud et al., 2024 [13]

Die Beispiele aus der Forschung von Fuglerud et al. zeigen deutlich, dass die Möglichkeiten von AI im Bereich der Web-Accessibility noch lange nicht voll ausgeschöpft werden. Der Bedarf an manuellen Überprüfungen kann mit diesem mächtigen Hilfsmittel weiter reduziert werden. Allerdings muss festgehalten werden, dass AI noch immer menschlicher Aufsicht und Training bedarf - zur Überwachung der Ergebnisse und zur Verbesserung der Genauigkeit und des Funktionsumfangs [20].

Insgesamt stellt die Integration von AI jedoch eine weitere viel versprechende Innovation dar, um Web-Inhalte für alle Nutzer zugänglicher zu machen.

8 Abkürzungen

A11y ..... Accessibility (*Barrierefreiheit*)

ACT ..... Accessibility Conformance Testing

ADA ..... Americans with Disabilities Act

AI ..... *Artificial Intelligence* (Künstliche Intelligenz)

API ..... Application Programming Interface (*Programmierschnittstelle*)

ARIA ..... Accessible Rich Internet Applications

BFSG ..... Barrierefreiheitsstärkungsgesetz

BGG ..... Behindertengleichstellungsgesetz

CSS ..... Cascading Style Sheets

DOM ..... Document Object Model

EAA ..... European Accessibility Act

EU ..... Europäische Union

GUI ..... Graphical User Interface

HTML ..... Hypertext Markup Language

ID ..... Identification

JSON ..... JavaScript Object Notation

QA ..... Quality assurance (*Qualitätssicherung*)

UI ..... User Interface (*Benutzeroberfläche*)

W3C ..... World Wide Web Consortium

WAI ..... Web Accessibility Initiative

WAI-ARIA ..... Web Accessibility Initiative - Accessible Rich Internet Applications

WCAG ..... Web Content Accessibility Guidelines

WCAG-EM ..... Website Accessibility Conformance Evaluation Methodology

SUT ..... System Under Test

XML ..... Extensible Markup Language

XPath ..... XML Path Language

9 Abbildungsverzeichnis

10 Tabellenverzeichnis

11 Codeverzeichnis

Codebeispiele und Anhang

12 Literaturverzeichnis

[1] Ahmad Anshari. "A SYSTEMATIC STUDY OF MULTIMEDIA IMPLEMENTATION AND ITS IMPACT TOWARDS USER ENGAGEMENT." In: *Journal of Educators Online* 21.2 (2024). DOI: 10.9743/jeo.2024.21.2.2. URL: https://www.thejeo.com/archive/archive/2024_212/24cirt0001_jeo_march_22_ansharipdf.

[2] BGG. *Behindertengleichstellungsgesetz - BGG*. Zuletzt geändert: Art. 7 G vom 23. Mai 2022. Mai 2002. URL: https://www.gesetze-im-internet.de/bgg/.

[3] Giorgio Brajnik. *The Barrier Walkthrough method*. Letzter Zugriff: 15.01.2025. 2011. URL: https://people.uniud.it/node/3465.

[4] Giorgio Brajnik, Yeliz Yesilada und Simon Harper. "Is accessibility conformance an elusive property? A study of validity and reliability of WCAG 2.0". In: *ACM Transactions on Accessible Computing (TACCESS)* 4.2 (2012), S. 1–28. URL: https://dl.acm.org/doi/abs/10.1145/2141943.2141946.

[5] Council of the European Union. *DIRECTIVE (EU) 2016/2102 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL*. Letzter Zugriff: 10.02.2025. Okt. 2016. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex:32016L2102.

[6] Council of the European Union. *DIRECTIVE (EU) 2019/882 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL*. Letzter Zugriff: 10.02.2025. Apr. 2019. URL: https://eur-lex.europa.eu/legal-content/DE/TXT/?uri=CELEX:32019L0882.

[7] Deque Systems. *axe API Documentation*. Letzter Zugriff: 25.02.2025. URL: https://www.deque.com/axe/core-documentation/api-documentation/.

[8] Deutscher Blinden- und Sehbehindertenverband e.V. (DBSV). *European Accessibility Act (EAA) - die Entstehung*. Letzter Zugriff: 12.12.2024. 2019. URL: https://www.dbsv.org/european-accessibility-act-entstehung.html.

[9] Deutscher Blinden- und Sehbehindertenverband e.V. (DBSV). *Zahlen und Fakten*. https://www.dbsv.org/zahlen-fakten.html. Letzter Zugriff: 12.12.2024. URL: https://www.dbsv.org/zahlen-fakten.html.

[10] Deutscher Bundestag. *Gesetz zur Verlängerung befristeter Regelungen im Arbeitsförderungsrecht und zur Umsetzung der Richtlinie (EU) 2016/2102 über den barrierefreien Zugang zu den Websites und mobilen Anwendungen öffentlicher Stellen*. Letzter Zugriff: 29.12.2024. Juli 2018. URL: https://dip.bundestag.de/vorgang/gesetz-zur-verlÃ¤ngerung-befristeter-regelungen-im-arbeitsfÃ¶rderungsrecht-und-zur-umsetzung/233374.

[11] EU Directorate General of Employment, Social Affairs and Inclusion. *Europäischer Rechtsakt zur Barrierefreiheit*. Letzter Zugriff: 10.02.2025. URL: https://employment-social-affairs.ec.europa.eu/policies-and-activities/social-protection-social-inclusion/persons-disabilities/union-equality-strategy-rights-persons-disabilities-2021-2030/european-accessibility-act_de.

[12] European Union. *European Union directives*. Artikel 288. Letzter Zugriff: 10.02.2025. URL: https://eur-lex.europa.eu/DE/legal-content/summary/european-union-directives.html.

[13] Kristin Skeide Fuglerud u. a. "Exploring the Use of AI for Enhanced Accessibility Testing of Web Solutions". In: *Universal Design 2024: Shaping a Sustainable, Equitable and Resilient Future for All. Proceedings of the Seventh International Conference on Universal Design (UD2024), Oslo, Norway, 20-22 November 2024*. 2024. URL: https://nr.brage.unit.no/nr-xmlui/bitstream/handle/11250/3167500/SHTI-320-SHTI241041.pdf?sequence=1.

[14] Renee Garett u. a. "A literature review: website design and user engagement". In: *Online journal of communication and media technologies* 6.3 (2016), S. 1. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC4974011/.

[15] Informations Technik Zentrum Bund. *Barrierefreie-Informationstechnik-Verordnung (BITV 2.0)*. Letzter Zugriff: 19.01.2025. URL: https://www.barrierefreiheit-dienstekonsolidierung.bund.de/Webs/PB/DE/gesetze-und-richtlinien/bitv2-0/bitv2-0-node.html.

[16] Informations Technik Zentrum Bund. *Barrierefreiheitsstärkungsgesetz (BFSG)*. Letzter Zugriff: 19.01.2025. Juni 2022. URL: https://www.barrierefreiheit-dienstekonsolidierung.bund.de/Webs/PB/DE/gesetze-und-richtlinien/barrierefreiheitsstaerkungsgesetz/barrierefreiheitsstaerkungsgesetz-node.html.

[17] Informations Technik Zentrum Bund. *Harmonisierte Europäische Norm (EN) 301 549*. Letzter Zugriff: 19.01.2025. URL: https://www.barrierefreiheit-dienstekonsolidierung.bund.de/Webs/PB/DE/gesetze-und-richtlinien/en301549/en301549-node.html.

[18] Informations Technik Zentrum Bund. *Web Content Accessibility Guidelines 2.1 (WCAG 2.1)*. Letzter Zugriff: 19.01.2025. URL: https://www.barrierefreiheit-dienstekonsolidierung.bund.de/Webs/PB/DE/gesetze-und-richtlinien/barrierefreiheitsstaerkungsgesetz/barrierefreiheitsstaerkungsgesetz-node.html.

[19] Georgia James. *Alles, was Sie zu manuellen, automatisierten und hybriden Barrierefreiheits-Tests wissen müssen*. Letzter Zugriff: 27.01.2025. Jan. 2021. URL: https://www.siteimprove.com/de/blog/alles-was-sie-zu-manuellen-automatisierten-und-hybriden-barrierefreiheits-tests-wissen-mussen/.

[20] Ifeanyi Madu und Chidiebere Nzenwa. "Ai in Web Development: Enhancing Accessibility". In: (2024). URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4949516.

[21] Delvani Antônio Mateus u. a. "A systematic mapping of accessibility problems encountered on websites and mobile apps: A comparison between automated tests, manual inspections and user evaluations". In: *Journal on Interactive Systems* 12.1 (2021), S. 145–171. URL: https://journals-sol.sbc.org.br/index.php/jis/article/view/1778/1848.

[22] Fausto O Medola u. a. "Experiences, problems and solutions in computer usage by subjects with tetraplegia". In: *Design, User Experience, and Usability: Users and Interactions: 4th International Conference, DUXU 2015, Held as Part of HCI International 2015, Los Angeles, CA, USA, August 2-7, 2015, Proceedings, Part II 4*. Springer. 2015, S. 131–137.

[23] NETPROFIT. *BFSG Barrierefreiheitsstärkungsgesetz*. https://bfsg-gesetz.de/. Letzter Zugriff: 12.11.2024. Anwaltlich geprüft zuletzt am 11.11.2024. 2024. URL: https://bfsg-gesetz.de/.

[24] Quality First Software GmbH. *QF-Test - Handbuch*. Letzter Zugriff: 20.02.2025. URL: https://www.qftest.com/qf-test-handbuch/lc/manual-de-manual.html.

[25] Quality First Software GmbH. *QF-Test - Produktbroschüre*. Letzter Zugriff: 20.02.2025. URL: https://www.qftest.com/fileadmin/Webdata/pdf/qf-test-produktbroschuere.pdf.

[26] Sven Schmutz, Andreas Sonderegger und Juergen Sauer. "Easy-to-read language in disability-friendly web sites: Effects on nondisabled users". In: *Applied Ergonomics* 74 (2019), S. 97–106. ISSN: 0003-6870. DOI: https://doi.org/10.1016/j.apergo.2018.08.013. URL: https://www.sciencedirect.com/science/article/pii/S0003687018302862.

[27] Sven Schmutz, Andreas Sonderegger und Juergen Sauer. "Implementing Recommendations From Web Accessibility Guidelines: A Comparative Study of Nondisabled Users and Users With Visual Impairments". In: *Human Factors* 59.6 (2017). PMID: 28467134, S. 956–972. DOI: 10.1177/0018720817708397. eprint: https://doi.org/10.1177/0018720817708397. URL: https://doi.org/10.1177/0018720817708397.

[28] Sven Schmutz, Andreas Sonderegger und Juergen Sauer. "Implementing Recommendations From Web Accessibility Guidelines: Would They Also Provide Benefits to Nondisabled Users". In: *Human Factors* 58.4 (2016). PMID: 27044605, S. 611–629. DOI: 10.1177/0018720816640962. eprint: https://doi.org/10.1177/0018720816640962. URL: https://doi.org/10.1177/0018720816640962.

[29] Statistisches Bundesamt - Destatis. *Internetnutzer/-innen und Online-Einkäufer/-innen 2024 nach Geschlecht und Alter*. Letzter Zugriff: 29.12.2024. 2024. URL: https://www.destatis.de/DE/Themen/Gesellschaft-Umwelt/Einkommen-Konsum-Lebensbedingungen/IT-Nutzung/Tabellen/nutzung-internet-onlinekaeufe-geschlecht-alter-mz-ikt.html.

[30] Statistisches Bundesamt - Destatis. *Statistik der schwerbehinderten Menschen 2021*. https://www.destatis.de/DE/Themen/Umwelt/Gesundheit/Behinderte-Menschen/_inhalt.html#sprg233848. Letzter Zugriff: 29.12.2024. Sep. 2022. URL: https://www.destatis.de/DE/Themen/Gesellschaft-Umwelt/Gesundheit/Behinderte-Menschen/_inhalt.html\#sprg233848.

[31] Amanda Swearngin u. a. "Towards automated accessibility report generation for mobile apps". In: *ACM Transactions on Computer-Human Interaction* 31.4 (2024), S. 1–44. URL: https://dl.acm.org/doi/full/10.1145/3674967.

[32] U.S. Department of JusticeCivil Rights Division. *Guidance on Web Accessibility and the ADA*. Letzter Zugriff: 20.11.2024. März 2022. URL: https://www.ada.gov/resources/web-guidance/.

[33] Markel Vigo, Justin Brown und Vivienne Conway. "Benchmarking web accessibility evaluation tools: measuring the harm of sole reliance on automated tests". In: *Proceedings of the 10th international cross-disciplinary conference on web accessibility*. 2013, S. 1–10. URL: https://dl.acm.org/doi/abs/10.1145/2461121.2461124.

[34] Beat Vollenwyder u. a. "How compliance with web accessibility standards shapes the experiences of users with and without disabilities". In: *International Journal of Human-Computer Studies* 170 (2023), S. 102956. URL: https://www.sciencedirect.com/science/article/pii/S1071581922001756.

[35] W3C, Michael Stokes, Mathew Anderson, Srinivasan Chandrasekar, Ricardo Motta. *A Standard Default Color Space for the Internet - sRGB*. Nov. 1996. URL: https://www.w3.org/Graphics/Color/sRGB.html.

[36] W3C, WAI. *Accessibility Conformance Testing (ACT) Overview*. Letzter Zugriff: 02.02.2025. URL: https://www.w3.org/WAI/standards-guidelines/act/.

[37] W3C, WAI. *Accessibility Conformance Testing (ACT) Rules Format 1.1*. Letzter Zugriff: 02.02.2025. Juni 2024. URL: https://www.w3.org/TR/act-rules-format-1.1/.

[38] W3C, WAI. *Accessible Rich Internet Applications (WAI-ARIA) 1.3*. Letzter Zugriff: 02.02.2025. Jan. 2024. URL: https://www.w3.org/TR/wai-aria-1.3/.

[39] W3C, WAI. *ACT Rules Implementation in Test Tools and Methodologies*. Letzter Zugriff: 02.02.2025. URL: https://www.w3.org/WAI/standards-guidelines/act/implementations/.

[40] W3C, WAI. *Understanding ACT Rules for WCAG Success Criteria*. Letzter Zugriff: 02.02.2025. URL: https://w3c.github.io/wcag-act/understanding-act-rules.html.

[41] W3C, WAI. *WAI-ARIA Overview*. Letzter Zugriff: 02.02.2025. URL: https://www.w3.org/WAI/standards-guidelines/aria/.

[42] W3C, WAI, Shadi Abou-Zahra, Judy Brewer. *Diverse Abilities and Barriers*. https://www.w3.org/WAI/people-use-web/abilities-barriers/. Letzter Zugriff: 10.11.2024, 1999. URL: https://www.w3.org/WAI/people-use-web/abilities-barriers/.

[43] W3C, WAI, Shawn Lawton Henry. *Introduction to Web Accessibility*. https://www.w3.org/WAI/fundamentals/accessibility-intro/. Letzter Zugriff: 10.11.2024. Feb. 2005. URL: https://www.w3.org/WAI/fundamentals/accessibility-intro/.

[44] W3C, WAI, Shawn Lawton Henry, Shadi Abou-Zahra. *WCAG-EM Overview: Website Accessibility Conformance Evaluation Methodology*. Letzter Zugriff: 07.02.2025. Apr. 2020. URL: https://www.w3.org/WAI/test-evaluate/conformance/wcag-em/.

[45] Web Accessibility Initiative WAI. *Web Content Accessibility Guidelines (WCAG) 2.2*. https://www.w3.org/TR/WCAG22/. Letzter Zugriff: 25.02.2025. Okt. 2023. URL: https://www.w3.org/TR/WCAG22/.

[46] web.dev. *Manuelle Tests der Barrierefreiheit*. Letzter Zugriff: 02.01.2025. 2023. URL: https://web.dev/learn/accessibility/test-manual?hl=de.

[47] wiki.selfhtml. *WAI-ARIA*. Letzter Zugriff: 02.02.2025. 2024. URL: https://wiki.selfhtml.org/wiki/WAI-ARIA.

[48] Bernd Wissinger, Susanne Kohl und Molekulargenetisches Labor. "Genetische Ursachen der Farbenblindheit". In: *BIOspektrum* 11.1 (2005), S. 29–33. URL: http://www.apickert.ch/farbenblindheit.pdf.

[49] Yeliz Yesilada und Simon Harper. "Web Accessibility". In: *Human–Computer Interaction Series. Springer London* (2019).

13 Anhang

Im Rahmen der Fallstudie wurde eine Testsuite geschrieben, welche das ZPA der Hochschule München automatisch auf Barrierefreiheit überprüft. Die Ergebnisse eines Testlaufs vom 24.02.2025 wurden als QF-Test Protokoll und ein QF-Test Report erzeugt. Diese wurden als einzelne Dateien zusätzlich zu dieser Arbeit abgegeben.
Die Testsuite ist hier im folgenden in ihrer XML-Rohform beigefügt. Im korrekten Format abgespeichert (`.qft`), ist sie mit einer QF-Test Version von 9.0.0 oder höher ausführbar.

13.1 Testsuite zpa11y.qft

Listing 4: Testsuite für Accessibility-Tests des ZPA, `zpa11y.qft`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE RootStep>
<RootStep id="_0" indentation="0" linelength="160" name="root" version="9.0.0">
<include>qfs.qft</include>
<CommentStep id="_2Jfd">
<heading>&lt;i color='red'&gt;WICHTIG: zum Ausführen ist ein valider Login in das ZPA erforderlich – Infos dazu in der Bemerkung dieses Kommentarknotens&lt;/i&gt;</heading>
<comment>Unter Prozeduren/Abhängigkeiten/loginZPA/Login muss der User und das Passwort gesetzt werden.
Per Rechtsklick –&gt; "Text verschlüsseln" können diese Werte verschlüsselt werden, damit sie nicht im Protokoll/Report auftauchen.

Genauere Informationen dazu finden sich im Handbuch (https://www.qftest.com/doc/manual/de/opt_play.html#opt_passwordsalt) oder in diesem Blog: https://www.qftest.com/blog/article/passwortverschluesselung.html</comment>
</CommentStep>
<CommentStep id="_2Jfo">
<heading>&lt;i color='red'&gt;Zudem müssen eventuell die Sondereinstellungen für einzelne Tabs geändert werden – Infos dazu in der Bemerkung dieses Kommentarknotens&lt;/i&gt;</heading>
<comment>Um zusätzliche Inhalte zu überprüfen (etwa meine alten Stundenpläne), bearbeite ich unter "Prozeduren/Seitenbearbeitung/Sondereinstellungen für einzelne Tabs" die aktuelle Seite – beispielsweise durch die Auswahl eines alten Semesters.

Diese Werte sind individuell und unterscheiden sich je nach eingeloggtem User, müssen also eventuell angepasst oder ausgeschaltet/entfernt werden.</comment>
</CommentStep>
<CommentStep id="_2JbW">
<heading>Die Tests des Headers sind aktuell ausgeschaltet, da dort keine Accessibility-Fehler auftreten</heading>
<comment>können über das Kontextmenü oder Strg+Shift+D wieder angeschaltet werden</comment>
</CommentStep>
<TestSet disabled="true" id="_2Jb8" name="Axe-Accessibility Tests des ZPA Headers (Navigationsleiste etc)">
<CommentStep id="_2Jb9">
<heading>Die Abhängigkeit kümmert sich vollautomatisch um Vorbereitung/Aufräumen</heading>
<comment>insbesondere modifiziert sie die Komponentenerkennung mittels des "CustomWebResolver"
dieser sorgt dafür, dass QF-Test z.B. die Navigationsleiste als TabPanel erkennt
und ermöglicht somit einen einfacheren und lesbareren Zugriff auf die einzelnen Tabs und den Seiteninhalt</comment>
</CommentStep>
<DependencyReference id="_2JbA" reference="Abhängigkeiten.loginZPA"/>
<TestCase id="_2JbC" name="Axe-Checks">
<ProcedureCall id="_2JbH" procedure="qfs.accessibility.web.checkAxeRules">
<variable name="rules">wcag2a, wcag2aa, wcag21a, wcag21aa, wcag22aa</variable>
<variable name="rulesToSkip"/>
<variable name="scope">#HEADER</variable>
<variable name="genericClassesToSkip"/>
<variable name="skipInvisibleElements">true</variable>
<variable name="showSuccessfulChecks">false</variable>
<variable name="showImpossibleChecks">true</variable>
<variable name="logOverviewScreenshot">true</variable>
<variable name="allowedHeightOfOverviewScreenshot">2000</variable>
<variable name="logElementScreenshots">all</variable>
<variable name="allowedNumberOfElementScreenshots">20</variable>
<variable name="logElementSmartIdToMessage">false</variable>
```

```
<variable name="squashCheckResultsWithSameMessage">false</variable>
</ProcedureCall>
</TestCase>
</TestSet>
<TestSet id="_6" name="Accessibility Tests des ZPA Inhalts">
<CommentStep id="_2Jb6">
<heading>Die Abhängigkeit kümmert sich vollautomatisch um Vorbereitung/Aufräumen</heading>
<comment>Insbesondere modifiziert sie die Komponentenerkennung mittels des "CustomWebResolver".
Dieser sorgt dafür, dass QF-Test z.B. die Navigationsleiste als TabPanel erkennt und ermöglicht somit einen einfacheren und lesbareren Zugriff auf die einzelnen Tabs und den Seiteninhalt.
Ändert sich der Aufbau der Seite, muss eventuell der CustomWebResolver angepasst werden.</comment>
</CommentStep>
<DependencyReference id="_2Jb5" reference="Abhängigkeiten.loginZPA"/>
<TestCase id="_O" name="Axe Checks">
<ComponentWaiter client="$(client)" component="#TabPanel:" id="_2JdN" local="true" timeout="5000"/>
<ClientScriptStep client="$(client)" id="_2JdL" interpreter="groovy" name="Bestimme Anzahl der zu überprüfenden Tabs">
<code>def navbar = rc.getComponent("#Window:label=zpa.cs.hm.edu@#TabPanel:")

//the following prints all available methods for the navbar
//qf.println(navbar.class.methods.collect { it.name })

//the following requires the CustomWebResolver to identify TabPanelItems correctly
def navbarEntries = navbar.getElementsByClassName("Item:TabPanelItem")

//set number of tabs for use in loop
rc.setLocal("numberOfTabs", navbarEntries.length)

def listOfTabNames = []
for (entry: navbarEntries){
    //qf.println(entry.getAllAttributes())     //href
    //qf.println(entry.getVisibleText())       //text of the TabPanelItem
    listOfTabNames.add(entry.getVisibleText())
}

rc.setLocal("tabNames", listOfTabNames)</code>
</ClientScriptStep>
<CommentStep id="_2JdM">
<heading>Die Schleife iteriert über die Tabs, deren Anzahl im obigen SUT-Skript festgestellt wird</heading>
<comment>Das sorgt dafür, dass weitere Tabs hinzugefügt werden können und  ohne Anpassung der Tests überprüft werden.</comment>
</CommentStep>
<RepeatSequence count="$(numberOfTabs)" id="_2JdK" name="Tab $_(iteration)" var="iteration">
<ClientScriptStep client="$(client)" id="_2Jdc" interpreter="groovy" name="Name des aktuellen Tabs">
<code>def tabNames =  rc.getObj("tabNames")
rc.setLocal("currentTabName", tabNames[rc.getInt("iteration")])</code>
</ClientScriptStep>
<TestStep id="_2Jba" name="Navigiere zu entsprechendem Tab">
<ProcedureCall id="_2JfJ" procedure="Seitenbearbeitung.Navigiere zu Tab">
<variable name="tab">$(currentTabName)</variable>
<variable name="fallbackIndex">$(iteration)</variable>
</ProcedureCall>
</TestStep>
<CommentStep id="_2Jf8">
<heading>Wähle bestimmte Elemente auf der Seite aus, um z.B. Stundenpläne anzuzeigen</heading>
<comment>Je nach eingeloggtem Account unterschiedlich.</comment>
</CommentStep>
<CommentStep id="_2Jfz">
<heading>&lt;i color='red'&gt;Muss je nach User angepasst oder übersprungen (Strg+Shift+D) werden&lt;/i&gt;</heading>
</CommentStep>
<TestStep id="_2JdZ" name="Sondereinstellungen auf bestimmten Tabs">
<ProcedureCall id="_2Jf7" procedure="Seitenbearbeitung.Sondereinstellungen für einzelne Tabs">
<variable name="currentTab">$(currentTabName)</variable>
</ProcedureCall>
```

```
105 </TestStep>
106 <TestStep id="_2JbX" name="Axe-Checks">
107 <CommentStep id="_2JfM">
108 <heading>Der "scope", also Bereich der Überprüfung, ist auf den Inhalt der Seite (#Panel:name=
        content, effektiv HTML/BODY/MAIN) beschränkt </heading>
109 </CommentStep>
110 <ProcedureCall id="_P" procedure="qfs.accessibility.web.checkAxeRules">
111 <variable name="rules">wcag2a, wcag2aa, wcag21a, wcag21aa, wcag22aa</variable>
112 <variable name="rulesToSkip"/>
113 <variable name="scope">#Panel:name=content</variable>
114 <variable name="genericClassesToSkip"/>
115 <variable name="skipInvisibleElements">true</variable>
116 <variable name="showSuccessfulChecks">false</variable>
117 <variable name="showImpossibleChecks">true</variable>
118 <variable name="logOverviewScreenshot">true</variable>
119 <variable name="allowedHeightOfOverviewScreenshot">2000</variable>
120 <variable name="logElementScreenshots">all</variable>
121 <variable name="allowedNumberOfElementScreenshots">20</variable>
122 <variable name="logElementSmartIdToMessage">true</variable>
123 <variable name="squashCheckResultsWithSameMessage">false</variable>
124 </ProcedureCall>
125 </TestStep>
126 </RepeatSequence>
127 </TestCase>
128 <TestCase id="_2Jf9" name="Farbkontrast Checks">
129 <TestStep id="_2JbY" name="Farbkontrast-Checks">
130 <CommentStep id="_2JfO">
131 <heading>Dort finden sich die meisten Grafikelemente, da die Knöpfe zur Wochenauswahl als IMGs
        hinterlegt sind</heading>
132 </CommentStep>
133 <ProcedureCall id="_2JfK" procedure="Seitenbearbeitung.Navigiere zu Tab">
134 <variable name="tab">mein Wochenplan</variable>
135 </ProcedureCall>
136 <CommentStep id="_2JfN">
137 <heading>Überprüft alle IMGs auf der gesamten Seite, effektiv das HM-Logo und zwei Knöpfe</
        heading>
138 <comment>showSuccessfulChecks -&amp;gt; true, da alle Elemente auf der Seite den Farbkontrast erfü
        llen, aber trotzdem etwas im Log zu sehen sein soll</comment>
139 </CommentStep>
140 <ProcedureCall id="_R" procedure="qfs.accessibility.web.checkColorContrastSimpleGraphics">
141 <variable name="genericClasses">IMG</variable>
142 <variable name="scope">genericDocument</variable>
143 <variable name="genericClassesToSkip"/>
144 <variable name="skipInvisibleElements">true</variable>
145 <variable name="showSuccessfulChecks">true</variable>
146 <variable name="showImpossibleChecks">true</variable>
147 <variable name="logOverviewScreenshot">true</variable>
148 <variable name="allowedHeightOfOverviewScreenshot">2000</variable>
149 <variable name="logElementScreenshots">all</variable>
150 <variable name="allowedNumberOfElementScreenshots">20</variable>
151 <variable name="logElementSmartIdToMessage">true</variable>
152 <variable name="squashCheckResultsWithSameMessage">false</variable>
153 </ProcedureCall>
154 </TestStep>
155 </TestCase>
156 </TestSet>
157 <PackageRoot id="_3">
158 <Package id="_S" name="Abhängigkeiten">
159 <Dependency id="_T" name="starteZPA">
160 <SetupSequence id="_7" name="Starte zpa.cs.hm.edu">
161 <BasicSequence id="_8" name="Globale Variablen setzen">
162 <SetGlobalStep id="_9" varname="client">
163 <default>zpa</default>
164 </SetGlobalStep>
165 <SetGlobalStep id="_A" varname="browser">
166 <default>chrome</default>
167 </SetGlobalStep>
168 <SetGlobalStep id="_B" varname="browserdir"/>
169 </BasicSequence>
170 <ClientWaiter client="$(client)" engine="web" id="_C" local="true" raise="false" resvarname="
        isSUTRunning" timeout="0">
```

```
171 <comment>Dieser Knoten prüft, ob das SUT bereits läuft. Das Ergebnis der Prüfung wird in der
        Variable isSUTRunning gespeichert. Diese Variable enthält entweder true, wenn das SUT lä
        uft oder false, wenn das SUT nicht läuft. Im folgenden IF-Knoten wird diese Variable
        ausgewertet.</comment>
172 </ClientWaiter>
173 <IfSequence id="_D" name="SUT starten, wenn notwendig" test="not $(isSUTRunning)">
174 <BasicSequence id="_E" name="Web-Engine starten">
175 <BrowserClientStarter browser="$(browser)" client="$(client)" connectionmode="Prefer CDP-
        Driver" executable="${qftest:java}" id="_F" mozhome="$(browserdir)"
176                         openwindow="false"/>
177 <ClientWaiter client="$(client)" engine="web" id="_G"/>
178 </BasicSequence>
179 <BasicSequence id="_H" name="Einstellungen für Browser setzen">
180 <ProcedureCall id="_I" procedure="qfs.web.browser.settings.doStartupSettings">
181 <variable name="client">$(client)</variable>
182 <variable name="browser">$(browser)</variable>
183 <variable name="mozhome">$(browserdir)</variable>
184 <variable name="emptyCache">true</variable>
185 <variable name="enableCookies">true</variable>
186 <variable name="deleteCookies">true</variable>
187 <variable name="locale">de</variable>
188 <variable name="enableProxy">false</variable>
189 <variable name="proxyAddress"/>
190 <variable name="proxyPort"/>
191 <variable name="proxyAutoconfigurl"/>
192 <variable name="enableProxyBypass">false</variable>
193 <variable name="proxyBypass"/>
194 <variable name="mozprofile"/>
195 <variable name="compatibilitymode">no</variable>
196 </ProcedureCall>
197 </BasicSequence>
198 <BasicSequence id="_J" name="Browser-Fenster öffnen">
199 <BrowserClientStarter browser="${default:browser:chrome}" client="$(client)" executable="${
        qftest:java}" id="_K" openwindow="true"
200                         url="https://zpa.cs.hm.edu/"/>
201 <DocumentWaiter client="$(client)" component="genericDocument" id="_L" timeout="60000"/>
202 </BasicSequence>
203 <BasicSequence id="_M" name="CustomWebResolver konfigurieren">
204 <CommentStep id="_2Jf+">
205 <heading>Wichtig für die Komponentenerkennung</heading>
206 </CommentStep>
207 <InstallCWRStep client="$(client)" engine="web" id="_N">
208 <code>base: jqueryui.v1.1.0-r0
209
210 genericClasses:
211 - TabPanel:
212     attribute: class
213     attributeValue: navigation
214     tag: DIV
215 - Item:TabPanelItem:
216     attribute: href
217     attributeValue:
218         value: https://zpa.cs.hm.edu/.*
219         regex: true
220     ancestor:
221         level: 3
222         className: TabPanel
223 - Panel:TabPanelContent:
224     attribute: id
225     attributeValue: content
226     tag: DIV
227
228 ignoreTags:
229 - &lt;DIV&gt;
230 - &lt;SPAN&gt;
231 </code>
232 <comment>Mit einem CustomWebResolver übersetzen Sie die Spezifika der Komponenten Ihrer Web-
        Anwendung in generische Klassen, die QF-Test versteht.
233 Dadurch werden die Komponentenerkennung drastisch verbessert und spezifischere Checks ermö
        glicht.
234
```

```
235 Die wichtigste Kategorie ist "genericClasses". Hier können Sie HTML-Elemente generischen
        Klassen zuweisen anhand deren HTML-Tags, CSS-Klassen oder HTML-Attributen.
236
237 Für weitere Informationen und Beispiele wählen Sie "Was ist das?" im Kontextmenü des Knotens.<
        /comment>
238 </InstallCWRStep>
239 </BasicSequence>
240 </IfSequence>
241 </SetupSequence>
242 <CleanupSequence id="_2JQo" name="beendeZPA">
243 <TryStep id="_3dl" name="bereits beendet? ">
244 <ProcessWaiter client="$(client)" id="_3dk" timeout="1"/>
245 <CatchSequence exception="ClientNotTerminatedException" id="_3dm" maxerror="0" name="noch
        nicht beendet">
246 <TryStep id="_3dz" name="beende ZPA">
247 <WindowEventStep client="$(client)" component="#Window:label=zpa.cs.hm.edu" event="
        WINDOW_CLOSING" id="_3do"/>
248 <ProcessWaiter client="$(client)" id="_3dp"/>
249 <CatchSequence exception="TestException" id="_3d+" maxerror="0" name="Browserfenster konnte
        nicht geschlossen werden -&gt; Prozess beenden">
250 <ClientStopper client="$(client)" id="_3e0"/>
251 <ProcessWaiter client="$(client)" id="_3d-"/>
252 </CatchSequence>
253 </TryStep>
254 </CatchSequence>
255 </TryStep>
256 </CleanupSequence>
257 </Dependency>
258 <Dependency id="_3Cw" name="loginZPA">
259 <comment>Stellt sicher, dass SUT gestartet und in den Ausgangszustand gesetzt wurde.</comment>
260 <DependencyReference id="_1pGA" reference="Abhängigkeiten.starteZPA">
261 <variable name="user"/>
262 <variable name="password"/>
263 </DependencyReference>
264 <SetupSequence id="_3Aj" name="Login">
265 <CommentStep id="_2Jat">
266 <heading>&lt;i color='red'&gt;User/Passwort HIER setzen und verschlüsseln&lt;/i&gt;</heading>
267 </CommentStep>
268 <CommentStep id="_2Jfi">
269 <heading>&lt;i color='red'&gt;Die aktuell eingetragenen Werte sind für meinen persönlichen
        Account und lassen sich ohne mein passwordsalt nicht entschlüsseln&lt;/i&gt;</heading>
270 </CommentStep>
271 <SetGlobalStep id="_2JRG" local="true" varname="userencrypt">
272 <default>${decrypt:A94BB5236119F146626D2377E2090826}</default>
273 </SetGlobalStep>
274 <SetGlobalStep id="_2Jas" local="true" varname="passwordencrypt">
275 <default>${decrypt:DF88A5222FD5F8F204151B63D41D6771}</default>
276 </SetGlobalStep>
277 <ProcedureCall id="_2JRA" procedure="Authentifikation.login">
278 <variable name="user">$(userencrypt)</variable>
279 <variable name="password">$(passwordencrypt)</variable>
280 </ProcedureCall>
281 </SetupSequence>
282 <CleanupSequence id="_1pGB" name="Logout">
283 <ProcedureCall id="_2JRH" procedure="Authentifikation.logout"/>
284 </CleanupSequence>
285 </Dependency>
286 </Package>
287 <Package id="_2JQ+" name="Authentifikation">
288 <Procedure id="_2JQ-" name="login">
289 <variable name="user"/>
290 <variable name="password"/>
291 <ComponentWaiter client="$(client)" component="#Link:label=login" id="_2JdS" local="true"
        raise="false" resvarname="notLoggedInAlready" timeout="5000"/>
292 <IfSequence id="_2JdT" name="Noch nicht eingeloggt" test="$(notLoggedInAlready)">
293 <MouseEventStep clicks="1" client="$(client)" component="#Link:label=login" event="MOUSE_MPRC"
         id="_2JR1" modifiers="16"/>
294 <DocumentWaiter client="$(client)" component="genericDocument" id="_2JR2" timeout="60000"/>
295 <ComponentWaiter client="$(client)" component="#Link:label=SHIBBOLETH..." id="_2JR3" local="
        true" raise="false"
296                  resvarname="notAutomaticallyLoggedInViaShibboleth" timeout="5000"/>
```

```
297 <IfSequence id="_2JR4" name="Nicht automatisch per Shibboleth eingeloggt" test="$(
        notAutomaticallyLoggedInViaShibboleth)">
298 <MouseEventStep clicks="1" client="$(client)" component="#Link:label=SHIBBOLETH..." event="
        MOUSE_MPRC" id="_2JR5" modifiers="16"/>
299 <DocumentWaiter client="$(client)" component="genericDocument" id="_2JR6" timeout="60000"/>
300 <TextInputStep client="$(client)" component="#TextField:name=username" id="_2JR7">
301 <text>$(user)</text>
302 </TextInputStep>
303 <TextInputStep client="$(client)" component="#PasswordField:name=password" id="_2JR8">
304 <text>$(password)</text>
305 </TextInputStep>
306 <KeyEventStep client="$(client)" component="#PasswordField:name=password" event="KEY_PTR" id="
        _2JR9" keychar="10" keycode="10" modifiers="0"/>
307 </IfSequence>
308 </IfSequence>
309 <ComponentWaiter client="$(client)" component="#Link:label=logout" id="_2Jfj" raise="false"
        timeout="5000"/>
310 </Procedure>
311 <Procedure id="_2JR0" name="logout">
312 <TryStep id="_2JRC" name="logout möglich?">
313 <MouseEventStep clicks="1" client="$(client)" component="#Link:label=logout" event="MOUSE_MPRC
        " id="_2JRE" modifiers="16"/>
314 <CatchSequence exception="ComponentNotFoundException" id="_2JRD" maxerror="0">
315 <LogMessageStep clientInfo="false" clientScreenshots="-1" dontcompact="false" errorlevel="1"
        id="_2JRF" report="false" screenshots="-1">
316 <text>"logout" wurde nicht gefunden, vermutlich bereits ausgeloggt</text>
317 </LogMessageStep>
318 </CatchSequence>
319 </TryStep>
320 </Procedure>
321 </Package>
322 <Package id="_2Jeq" name="Seitenbearbeitung">
323 <Procedure id="_2JfG" name="Navigiere zu Tab">
324 <variable name="tab"/>
325 <variable name="fallbackIndex"/>
326 <TryStep id="_2Jf-" name="Tab über Namen findbar‘?">
327 <MouseEventStep clicks="1" client="$(client)" component="#Item:TabPanelItem:$(tab)" event="
        MOUSE_MPRC" id="_2JfH" modifiers="16"/>
328 <CatchSequence exception="ComponentNotFoundException" id="_2Jg0" maxerror="0" name="Über Namen
         nicht auffindbar, verwende Index als Fallback-Value">
329 <MouseEventStep clicks="1" client="$(client)" component="#Item:TabPanelItem:&lt;${
        default:fallbackIndex:0}&gt;" event="MOUSE_MPRC" id="_2Jg1" modifiers="16">
330 <comment>Falls kein Index gesetzt -&gt; 0</comment>
331 </MouseEventStep>
332 </CatchSequence>
333 </TryStep>
334 <ComponentWaiter client="$(client)" component="#Panel:name=content" id="_2JfI" local="true"
        timeout="5000"/>
335 </Procedure>
336 <CommentStep id="_2Jfx">
337 <heading>&lt;i color='red'&gt;Mit Rechtsklick-&gt;"Referenzen finden" können alle Verwendungen
         dieser Prozedur gefunden werden&lt;/i&gt;</heading>
338 </CommentStep>
339 <Procedure id="_2Jf6" name="Sondereinstellungen für einzelne Tabs">
340 <variable name="currentTab"/>
341 <CommentStep id="_2Jfy">
342 <heading>&lt;i color='red'&gt;Hier müssen bei anderen Usern entweder die Werte angepasst
        werden, oder eventuell muss die Prozedur übersprungen werden (Strg+Shift+D)&lt;/i&gt;</
        heading>
343 </CommentStep>
344 <IfSequence id="_2Jdd" interpreter="groovy" test="rc.getStr(&#34;currentTab&#34;).startsWith
        (&#34;mein Wochenplan&#34;)">
345 <TextInputStep clear="true" client="$(client)" component="#TextField:name=id_date" id="_2JeH">
346 <text>11.11.2023</text>
347 </TextInputStep>
348 <KeyEventStep client="$(client)" component="#TextField:name=id_date" event="KEY_PTR" id="_2JeI
        " keychar="10" keycode="10" modifiers="0"/>
349 <MouseEventStep clicks="1" client="$(client)" component="#FORM:name=form_week_plan" event="
        MOUSE_MPRC" id="_2JeX" modifiers="16"/>
350 <DocumentWaiter client="$(client)" component="genericDocument" id="_2JeW" timeout="60000"/>
```

```
351 <ElseifSequence id="_2JeJ" interpreter="groovy" test="rc.getStr(&#34;currentTab&#34;).
        startsWith(&#34;mein Stundenplan&#34;)">
352 <SelectionEventStep client="$(client)" component="#ComboBox:name=id_semester@2024 SS" detail="
        0" event="SELECTION" id="_2JeU"/>
353 <DocumentWaiter client="$(client)" component="genericDocument" id="_2JeV" timeout="60000"/>
354 </ElseifSequence>
355 <ElseifSequence id="_2JeY" interpreter="groovy" test="rc.getStr(&#34;currentTab&#34;).
        startsWith(&#34;Stundenpläne&#34;)">
356 <SelectionEventStep client="$(client)" component="#ComboBox:name=id_group@IF7" detail="0"
        event="SELECTION" id="_2JeZ"/>
357 <DocumentWaiter client="$(client)" component="genericDocument" id="_2Jea" timeout="60000"/>
358 </ElseifSequence>
359 <ElseifSequence id="_2Jen" interpreter="groovy" test="rc.getStr(&#34;currentTab&#34;).
        startsWith(&#34;Meine Wahlen&#34;)">
360 <SelectionEventStep client="$(client)" component="#ComboBox:name=id_semester@2024 SS" detail="
        0" event="SELECTION" id="_2Jeo"/>
361 <DocumentWaiter client="$(client)" component="genericDocument" id="_2Jep" timeout="60000"/>
362 </ElseifSequence>
363 </IfSequence>
364 </Procedure>
365 </Package>
366 </PackageRoot>
367 <ExtraSequence id="_4"/>
368 <WindowList id="_5"/>
369 </RootStep>
```