THM | CAMPUS GIESSEN | MNI
TECHNISCHE HOCHSCHULE MITTELHESSEN
Mathematik, Naturwissenschaften und Informatik

Bachelor Thesis

Barrierefreie Webentwicklung: Analyse der Herausforderungen und Entwicklung eines Testkonzepts

zur Erlangung des akademischen Grades

Bachelor of Science

eingereicht im Fachbereich Mathematik, Naturwissenschaften und Informatik an der Technischen Hochschule Mittelhessen

von

Maximilian Firla

31. Dezember 2024

Referent: Prof. Dr. Dennis Priefer

Korreferent: M. Sc. Kevin Linne

Erklärung zur Verwendung von generativer KI

In Übereinstimmung mit den Empfehlungen der Deutschen Forschungsgemeinschaft (DFG)[1] und denen der Zeitschrift Theoretical Computer Science[2] erkläre ich (der Autor/die Autorin) hiermit den Einsatz von generativer KI.

Bei der Vorbereitung dieser Arbeit habe ich ChatGPT 4 verwendet, um ausschließlich die Lesbarkeit und Sprache zu verbessern. Nach der Verwendung von ChatGPT 4 habe ich den Inhalt überprüft und nach Bedarf bearbeitet und übernehme die volle Verantwortung für den Inhalt dieser Arbeit.

Erklärung der Selbstständigkeit

Hiermit versichere ich, die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die Zitate deutlich kenntlich gemacht zu haben.

Gießen, den 31. Dezember 2024 Maximilian Firla

1 DFG formuliert Richtlinien für den Umgang mit generativen Modellen für Text- und Bild: https://www.dfg.de/en/news/news-topics/announcements-proposals/2023/info-wissenschaft-23-72

2 Erklärung zur Verwendung von generativer KI in wissenschaftlichen Arbeiten: https://www.sciencedirect.com/journal/theoretical-computer-science/publish/guide-for-authors

Gender-Disclaimer

In dieser Arbeit wird aus Gründen der besseren Lesbarkeit das generische Maskulinum verwendet. Weibliche und anderweitige Geschlechteridentitäten werden dabei ausdrücklich mitgemeint, soweit es für die Aussage erforderlich ist.

Sperrvermerk

Diese Bachelorarbeit basiert auf internen und vertraulichen Daten der Firma Sylphen GmbH & Co. KG. Diese Arbeit darf Dritten, mit Ausnahme der betreuenden Dozenten und befugten Personen des Prüfungsausschusses ohne ausdrückliche Zustimmung des Unternehmens und des Verfassers nicht zugänglich gemacht werden. Eine Vervielfältigung und Veröffentlichung ohne ausdrückliche Genehmigung - auch in Auszügen - ist nicht erlaubt.

Diese Bachelorarbeit untersucht die technischen und organisatorischen Herausforderungen bei der Umsetzung von Barrierefreiheit in modernen Webanwendungen. Ausgangspunkt ist die wachsende Bedeutung digitaler Teilhabe sowie rechtliche Vorgaben wie das Barrierefreiheitsstärkungsgesetz, welches ab 2025 verbindliche Standards für digitale Angebote einführt. Ziel dieser Arbeit ist die Entwicklung eines Testkonzepts, das Barrierefreiheitskriterien frühzeitig und automatisiert im Entwicklungsprozess prüft und fortlaufend überwacht. Hierfür werden relevante Richtlinien und Standards analysiert sowie bestehende Prüfmethoden bewertet. Darauf aufbauend wird eine Vorgehensweise erarbeitet, die automatisierte Tests in bestehende CI/CD-Pipelines integriert. Mithilfe des Frameworks *jest-axe* wird eine *AccessibilityTester*-Klasse entwickelt, die Tests standardisiert, Redundanzen reduziert und die Integration erleichtert. Die Praxistauglichkeit wird am Kundenportal der Sylphen GmbH & Co. KG demonstriert.

Gleichzeitig betont diese Arbeit die Grenzen rein automatisierter Verfahren und empfiehlt zusätzlich nutzerzentrierte Teststrategien, um individuelle Barrieren gezielt zu adressieren. Damit entsteht ein praxisnaher Leitfaden, der die Bedeutung digitaler Inklusion unterstreicht.

This bachelor thesis examines the technical and organizational challenges in implementing accessibility in modern web applications. The starting point is the growing importance of digital inclusion and legal requirements such as the Accessibility Strengthening Act (Barrierefreiheitsstärkungsgesetz), which will introduce binding standards for digital services in Germany starting in 2025. The aim of this thesis is to develop a testing concept that evaluates accessibility criteria early and automatically in the development process and continuously monitors them. To achieve this, relevant guidelines and standards are analyzed, and existing testing methods are assessed. Building on this, an approach is developed that integrates automated tests into existing CI/CD pipelines. Using the framework *jest-axe*, an *AccessibilityTester* class is developed to standardize tests, reduce redundancies, and facilitate integration. The practicality of this approach is demonstrated using the customer portal of Sylphen GmbH & Co. KG.

At the same time, this thesis highlights the limitations of purely automated methods and recommends user-centered testing strategies to address individual barriers effectively. This results in a practical guide that emphasizes the importance of digital inclusion.

Inhaltsverzeichnis

1 Einführung

> „Internet IS for everyone – but it won't be unless WE make it so."
> – *Vinton Cerf*

Das Internet hat sich zu einem unverzichtbaren Bestandteil des täglichen Lebens entwickelt und spielt eine entscheidende Rolle in der modernen Gesellschaft. Es ist von besonderer Bedeutung, den Zugang zum Internet inklusiv zu gestalten, um die gleichberechtigte Teilhabe aller Menschen, einschließlich Personen mit Beeinträchtigungen oder Behinderungen, sicherzustellen. Derzeit nutzen 94 % der deutschen Bevölkerung das Internet zumindest gelegentlich (vgl. [Ini23]). Bereits 80 % der Bevölkerung nutzen das Internet täglich (vgl. [Bei]). In Deutschland leben etwa 7,9 Millionen Menschen mit einer schweren Behinderung, definiert durch einen Grad der Behinderung von über 50 % (vgl. [Sta24]). Bezogen auf eine Gesamtbevölkerung von rund 85 Millionen Menschen (vgl. [deu]) entspricht dies einem Anteil von 9,5 %. Zusätzlich existiert eine oft vernachlässigte Gruppe von Menschen mit temporären Einschränkungen, wie beispielsweise einem gebrochenen Arm (vgl. [Bunb]), der es den Betroffenen erschwert oder unmöglich macht, auf herkömmliche Weise Geräte wie Tastaturen, Mäuse oder Touchscreens zu bedienen. Auch diese Personen sind auf barrierefreie digitale Angebote angewiesen, um uneingeschränkt am gesellschaftlichen und digitalen Leben partizipieren zu können.

1.1 Motivation

Mit der zunehmenden Verbreitung von Single Page Applications SPA[1] (vgl. [Stab]), gemessen an der Popularität von Technologien wie Node.js und React (vgl. [staa]) entstehen zusätzliche Herausforderungen für die Barrierefreiheit. SPAs ermöglichen dynamische und interaktive Benutzerinteraktionen, ohne dass die Seite vollständig neu geladen werden muss (vgl. [Ado]). Dies bringt jedoch spezifische Probleme mit sich, wie das Management des Fokus bei dynamisch geladenen Inhalten und die korrekte Kommunikation dieser Inhalte durch assistive Technologien wie Screenreader (vgl. [Swe]).

1 Eine Single Page Application (SPA) ist eine Webanwendung, die beim Navigieren keine vollständigen Seitenladevorgänge erfordert. Inhalte werden dynamisch aktualisiert, wodurch ein schnelleres und flüssigeres Nutzererlebnis entsteht.

Die komplexe Natur von SPAs erfordert daher eine gezielte Implementierung von Barrierefreiheitsmaßnahmen, um sicherzustellen, dass diese für alle zugänglich sind. Im Rahmen dieser Entwicklungen wurde das Barrierefreiheitsstärkungsgesetz (BFSG) verabschiedet, das am 28. Juni 2025 in Kraft tritt. Es legt verbindliche Anforderungen an die Barrierefreiheit von Produkten und Dienstleistungen, einschließlich digitaler Angebote, fest. Das Gesetz zielt darauf ab, den Zugang für Menschen mit Behinderungen zu verbessern und stellt sicher, dass alle gleichberechtigt an der digitalen Gesellschaft teilhaben können. Es unterstreicht die Dringlichkeit, das Internet barrierefrei zu gestalten und setzt klare Standards, die von Unternehmen und öffentlichen Stellen erfüllt werden müssen (vgl. [Buna, BG]).

Aber die Barrierefreiheit ist nicht nur für Menschen mit Behinderungen relevant. Auch 13 % der Befragten ohne Behinderungen geben an, dass schlecht nutzbare oder unübersichtliche Webseiten im Alltag als Hindernis empfunden werden (vgl. [Akt21]). Dazu gehören beispielsweise unzureichende Kontrastwerte (siehe Abbildung 1.1) bei der Nutzung mobiler Endgeräte in zu hellen oder zu dunklen Umgebungen.

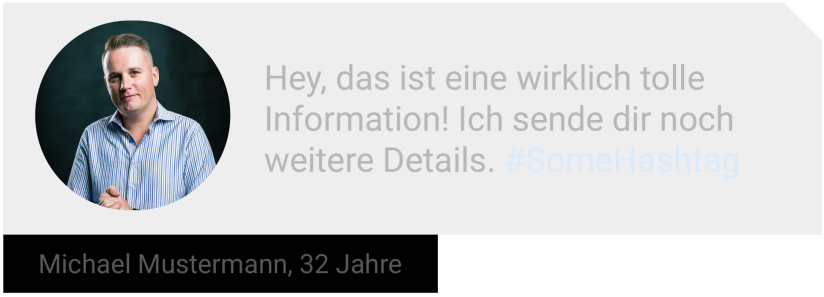

Abbildung 1.1: Unzureichende Kontrastwerte

Die Bereitstellung von Untertiteln für Videos und Podcasts erweist sich nicht nur für Personen mit Hörbeeinträchtigungen als nützlich, sondern ist auch in lauten Umgebungen, wie beispielsweise in der U-Bahn, sowie in Situationen in denen das Abspielen von Ton nicht möglich ist - etwa in Bibliotheken oder Wartezimmern - von großer Bedeutung. Untertitel ermöglichen es, die Inhalte auch unter diesen Bedingungen vollständig zu erfassen (vgl. [Wor24a]).

Zusätzlich bietet die Barrierefreiheit im Web auch für SEO[1] Vorteile. Sie sorgt für eine klarere Struktur der Inhalte. Wesentliche SEO-Aspekte wie Alternativ-Texte, die korrekte Nutzung von Überschriften und die Lesbarkeit von URLs[2] sind eng mit den Prinzipien der Barrierefreiheit verknüpft. Eine barrierefreie Website ermöglicht es

1 SEO (Search Engine Optimization) bezeichnet die Optimierung von Websites, um deren Sichtbarkeit in den Ergebnissen von Suchmaschinen zu verbessern.
2 URL (Uniform Resource Locator) ist die eindeutige Adresse einer Ressource im Internet.

Suchmaschinen, die Inhalte besser zu interpretieren und dadurch besser zu ranken, was zu einer potenziell höheren Suchmaschinenplatzierung führen kann (vgl. [Eve, Yam]).

Die Einführung des Barrierefreiheitsstärkungsgesetzes (BFSG) stellt Unternehmen vor die Herausforderung, moderne Webanwendungen barrierefrei zu gestalten. Insbesondere bei dynamischen Anwendungen ergeben sich technische und organisatorische Schwierigkeiten, die einer systematischen Analyse bedürfen. Diese Bachelorarbeit zielt darauf ab, die Herausforderungen bei der Implementierung von Barrierefreiheit in Webanwendungen zu identifizieren und ein Testkonzept zu entwickeln, welches den Anforderungen des BFSG gerecht wird und sich in den Entwicklungsprozess integrieren lässt um eine hohe Benutzerfreundlichkeit sicherzustellen.

1.2 Ziele dieser Arbeit

Das Ziel dieser Arbeit besteht darin, die Herausforderungen bei der Implementierung der Barrierefreiheit in modernen Webanwendungen zu identifizieren, analysieren und kritisch zu beurteilen. Dabei soll ein Verständnis dafür entwickelt werden, welche technischen, organisatorischen oder praktischen Hürden im Entwicklungsprozess auftreten und welche Auswirkungen diese auf die Barrierefreiheit von Webanwendungen haben.

Ein weiteres Ziel ist die Entwicklung eines Testkonzepts für die Überprüfung der Barrierefreiheit von Webanwendungen. Dieses Testkonzept soll den Anforderungen des Barrierefreiheitsstärkungsgesetzes (BFSG) gerecht werden und sich in bestehende Entwicklungsprozesse integrieren lassen, damit eine kontinuierliche Überprüfung der Barrierefreiheit ermöglicht wird.

Durch die Ergebnisse dieser Arbeit soll ein Beitrag dazu geleistet werden, dass Webanwendungen für alle Nutzergruppen, unabhängig von ihren individuellen Bedürfnissen, besser zugänglich werden.

Im Fokus dieser Arbeit stehen die folgenden Forschungsfragen:

1. **Welche Herausforderungen existieren bei der Implementierung von Barrierefreiheit in Webanwendungen?**
2. **Wie sieht ein Test-Konzept für die Barrierefreiheit von Webanwendungen aus und wie kann es in den Entwicklungsprozess integriert werden?**

1.3 Methodik

Zur Beantwortung der Forschungsfragen dieser Arbeit wird eine Kombination aus qualitativen und quantitativen Forschungsansätzen angewendet. Die erste Forschungsfrage: *Welche Herausforderungen existieren bei der Implementierung von Barrierefreiheit in Webanwendungen?* wird durch eine qualitative Literaturrecherche beantwortet. Die Datenbasis umfasst wissenschaftliche Publikationen, Gesetzestexte, Studien sowie Fachartikel, die sich mit der Barrierefreiheit von Webanwendungen beschäftigen. Anhand der Analyse dieser Quellen werden technische, organisatorische und rechtliche Herausforderungen identifiziert, die bei der Umsetzung der Barrierefreiheit auftreten können.

Für die Bearbeitung der zweiten Forschungsfrage: *Wie sieht ein Test-Konzept für die Barrierefreiheit von Webanwendungen aus und wie kann es in den Entwicklungsprozess integriert werden?* wird ein qualitativer, sowie ein experimenteller Ansatz, verfolgt. Zunächst erfolgt eine Recherche bestehender Literatur und Methodiken, um aktuelle Testverfahren und Software, zur Überprüfung der Barrierefreiheit, zu identifizieren und deren Anwendungsbereiche, sowie Vor- und Nachteile zu evaluieren. Die Auswahl der Testmethoden basiert auf den Kriterien Aktualität, Eignung für CI/CD[1]-Prozesse und der Fähigkeit, dynamische Inhalte zu überprüfen. Ergänzt wird das Testkonzept durch manuelle Überprüfungen in Bereichen, die durch automatisierte Verfahren nicht abgedeckt werden. Das entwickelte Testkonzept wird prototypisch in einen CI/CD-Workflow integriert, um zu evaluieren, wie sich die kontinuierliche Überprüfung der Barrierefreiheit auf den Entwicklungsprozess auswirkt. Hierzu wird das Testkonzept in die Build-Pipeline einer Webanwendung implementiert, sodass bei jedem neuen Build automatische Barrierefreiheits-Tests durchgeführt werden. Die Ergebnisse der Tests werden ausgewertet und führen gegebenenfalls zu Optimierungen im Entwicklungsprozess. Die Ergebnisse des Testkonzepts werden qualitativ analysiert, indem die ermittelten Barrieren kategorisiert und die Effektivität der Tests hinsichtlich ihrer Praktikabilität und Genauigkeit bewertet wird. Ausschlusskriterien bei der Entwicklung dieses Testkonzepts waren Verfahren, welche nicht in CI/CD-Prozesse integrierbar sind oder nur zur Überprüfung statischer HTML[2]-Seiten geeignet sind.

1 CI/CD (Continuous Integration/Continuous Deployment) beschreibt automatisierte Prozesse um Codeänderungen kontinuierlich zu testen, zu integrieren und bereitzustellen, um Entwicklungszyklen zu beschleunigen und die Qualität zu sichern.

2 HTML (HyperText Markup Language) ist die standardisierte Auszeichnungssprache des Internets, mit der die Struktur und Inhalte von Webseiten definiert werden.

1.4 Abgrenzung

In dieser Bachelorarbeit wird die Barrierefreiheit von modernen Webanwendungen untersucht. Der Fokus liegt dabei unter anderem auf Single Page Applications (SPAs), deren dynamisches Rendern spezifische Herausforderungen für die Barrierefreiheit mit sich bringt, die bei der Evaluation berücksichtigt werden müssen. Darum wird gezielt auf die Rolle von ARIA[1]-Attributen und deren Implementierung zur Verbesserung der Barrierefreiheit in SPAs eingegangen. Andere Aspekte der Webentwicklung, wie UX[2]/UI[3]-Prinzipien oder die Performance-Optimierung, werden nur dann thematisiert, wenn sie im direkten Zusammenhang mit der Barrierefreiheit stehen. Im Rahmen dieser Arbeit kommen automatisierte Testverfahren zur Bewertung der Barrierefreiheit zum Einsatz. Diese Verfahren konzentrieren sich ausschließlich auf Barrierefreiheitsaspekte, während andere potenziell prüfbare Bereiche wie Performance oder Sicherheitsaspekte, die ebenfalls von den Tools erfasst werden könnten, bewusst nicht berücksichtigt werden.

Das entwickelte Testkonzept zielt darauf ab, Entwicklern und Unternehmen einen praktischen Leitfaden an die Hand zu geben, um die Anforderungen der Web Content Accessibility Guidelines (WCAG) und des Barrierefreiheitsstärkungsgesetzes (BFSG) in Webanwendungen zu testen. Allerdings wird kein Anspruch auf Vollständigkeit erhoben. Es wird darauf hingewiesen, dass zukünftige Weiterentwicklungen in rechtlichen Rahmenbedingungen oder technischen Standards dazu führen können, dass die in dieser Arbeit vorgestellten Ergebnisse an Aktualität verlieren.

1.5 Struktur der Arbeit

Das erste Kapitel führt in das Thema der Barrierefreiheit ein und erläutert die Relevanz im Bezug auf Webanwendungen. Hier werden die Forschungsfragen formuliert und die Zielsetzung der Arbeit dargelegt. Im zweiten Kapitel werden die Grundlagen erläutert. Es werden die theoretischen Konzepte, welche für das Verständnis der Barrierefreiheit in Webanwendungen wesentlich sind, dargestellt. Im dritten Kapitel wird die theoretische Betrachtung der Thematik vollzogen. Zunächst erfolgt eine Analyse, bei der die Anforderungen an barrierefreie Webanwendungen ermittelt werden. Im Anschluss werden die

1 ARIA (Accessible Rich Internet Applications) ist ein Standard, der die Barrierefreiheit für dynamische Webinhalte verbessert. Es werden zusätzliche Attribute verwendet, um Screenreadern und assistiven Technologien die Bedeutung und Funktion von Elementen verständlich zu machen.

2 UX (User Experience) beschreibt die Gesamterfahrung und Wahrnehmung eines Nutzers bei der Interaktion mit einem Produkt, einer Webseite oder einem Service, einschließlich Bedienbarkeit, Design und Emotionen.

3 UI (User Interface) bezieht sich auf die visuelle Gestaltung und Funktionalität der Benutzeroberfläche, wie Layout, Buttons und Navigationselemente, welche die Interaktion ermöglichen.

Herausforderungen und Implikationen bei der Implementierung von Barrierefreiheit diskutiert. Es werden typische Probleme und Hürden aufgezeigt, welche bei der Umsetzung der Barrierefreiheit auftreten können. Daran anknüpfend wird das Technologiekonzept entwickelt, welches die Testverfahren und Werkzeuge zum Testen der Barrierefreiheit beinhaltet. Das vierte Kapitel beschreibt die praktische Umsetzung des in Kapitel 3 entwickelten Konzepts. Die Implementierung und Durchführung des Testkonzepts wird beschrieben und die Integration in eine real existierende Anwendung demonstriert. Das fünfte Kapitel behandelt die Zusammenfassung der Ergebnisse, das Fazit der Arbeit und die wichtigsten Erkenntnisse.

2 Hintergrund

Dieser Abschnitt bietet eine Einführung in die Definition und Bedeutung von Barrierefreiheit im Bezug auf digitale Produkte. Zunächst werden die rechtlichen Grundlagen näher beleuchtet, welche Barrierefreiheit in Webanwendungen vorschreiben und fördern. Anschließend werden grundlegende Technologien vorgestellt, die eingesetzt werden, um die Barrierefreiheit umzusetzen. Die Betrachtung dieser Aspekte sorgt für den benötigten Hintergrund, um im weiteren Verlauf der Arbeit aufzuzeigen, wie digitale Angebote entwickelt und getestet werden.

2.1 Definition und Bedeutung von Barrierefreiheit

Das Bundesamt für Justiz der Bundesrepublik Deutschland definiert den Begriff *barrierefrei* wie folgt:

> "Barrierefrei sind bauliche und sonstige Anlagen, Verkehrsmittel, technische Gebrauchsgegenstände, Systeme der Informationsverarbeitung, akustische und visuelle Informationsquellen und Kommunikationseinrichtungen sowie andere gestaltete Lebensbereiche, wenn sie für Menschen mit Behinderungen in der allgemein üblichen Weise, ohne besondere Erschwernis und grundsätzlich ohne fremde Hilfe auffindbar, zugänglich und nutzbar sind. Hierbei ist die Nutzung behinderungsbedingt notwendiger Hilfsmittel zulässig"[BD24].

Barrierefreie Produkte sind so gestaltet, dass sie von Menschen jeden Alters und mit verschiedenen Fähigkeiten weitgehend gleichberechtigt und ohne fremde Hilfe genutzt werden können. Dies schließt insbesondere Menschen mit sensorischen, motorischen oder kognitiven Einschränkungen ein (vgl. [Vie15]). Um die Bedeutung der Barrierefreiheit in der digitalen Welt zu verstehen, ist es wichtig, die Entwicklung der entsprechenden Standards und Richtlinien nachzuvollziehen. Die Inhalte dieser Arbeit beziehen sich dabei auf die Barrierefreiheit von Webanwendungen. In diesem Kontext werden die Begriffe *Barrierefreiheit* und *Zugänglichkeit* synonym verwendet.

Im deutschsprachigen Raum hat sich der Begriff *barrierefrei* erst seit den frühen 2000er Jahren durch die Einführung in DIN-Normen und gesetzliche Regelungen etabliert. Seit-

dem wird er zunehmend auch im allgemeinen Sprachgebrauch verwendet (vgl. [Cro24]). Eine erste Definition von Barrierefreiheitsstandards wurde mit den Web Content Accessibility Guidelines, kurz WCAG geschaffen, die von der Web Accessibility Initiative WAI[1] des World Wide Web Consortiums W3C[2] erarbeitet und 1999 als Empfehlung veröffentlicht wurden. Diese Guidelines stellen anwendbare Kriterien für die Gestaltung von Webangeboten bereit. Sie sorgen dafür, dass Webinhalte für Menschen mit unterschiedlichen Einschränkungen uneingeschränkt verwendbar sind. Die Barrierefreiheit wird durch die optimale Nutzung unterstützender Technologien, in Kombination mit einem geeigneten Design, erreicht (vgl. [Cro24]).

2.2 Rechtliche und normative Rahmenbedingungen

Die rechtlichen und normativen Rahmenbedingungen für Barrierefreiheit im Web sind in Deutschland durch verschiedene Gesetze, Verordnungen und Normen geregelt, die sicherstellen, dass digitale Angebote für alle Menschen zugänglich sind. Eine zentrale Rolle spielt das Behindertengleichstellungsgesetz (BGG), welches die Gleichstellung von Menschen mit Behinderungen in Deutschland fördert. Im § 4 des BGG wird Barrierefreiheit definiert als die Auffindbarkeit, Zugänglichkeit und Nutzbarkeit von Anlagen, Verkehrsmitteln, technischen Gebrauchsgegenständen sowie Informationsverarbeitungssystemen, ohne besondere Erschwernis und grundsätzlich ohne fremde Hilfe (vgl. [BD24]). Diese Definition unterstreicht, dass barrierefreie Webanwendungen so gestaltet sein müssen, dass sie für Menschen mit verschiedenen Einschränkungen zugänglich und nutzbar sind.

Barrierefreiheitsstärkungsgesetz

Das Barrierefreiheitsstärkungsgesetz (BFSG), welches im Juni 2025 in Kraft tritt, setzt die Barrierefreiheitsanforderungen für digitale Produkte und Dienstleistungen in deutsches Recht um. Es verpflichtet deren Anbieter diese barrierefrei zu gestalten, um den Zugang für Menschen mit Behinderungen zu gewährleisten. Das BFSG stellt somit einen entscheidenden Schritt zur Verbesserung der Barrierefreiheit im digitalen Raum dar, indem es verbindliche Standards für die Gestaltung von Webseiten, Software und anderen digitalen Angeboten festlegt. Aktuell wird dabei die Version 2.1 der Web Content Accessibility Guidelines herangezogen, zukünftig jedoch auf die neuere Version

1 WAI (Web Accessibility Initiative) ist ein Programm des W3C, das Richtlinien, Standards und Ressourcen zur Verbesserung der Barrierefreiheit im Web entwickelt.

2 W3C (World Wide Web Consortium) ist die internationale Organisation, die Standards für das Web entwickelt, um dessen langfristige Entwicklung und Interoperabilität sicherzustellen.

2.2 umgestellt, welche sechs zusätzliche Anforderungen enthält. Damit macht das BFSG die Einhaltung der WCAG auf den Konformitätsstufen A & AA verbindlich (vgl. [BG]).

Barrierefreie-Informationstechnik-Verordnung

Die Barrierefreie-Informationstechnik-Verordnung (BITV) 2.0 setzt die Anforderungen des BGG für die Barrierefreiheit von Internetangeboten öffentlicher Stellen um. Sie verpflichtet öffentliche Einrichtungen, ihre Websites und mobilen Anwendungen barrierefrei zu gestalten und orientiert sich dabei an den Web Content Accessibility Guidelines 2.2, sowie an zusätzlichen Anforderungen der EN 301 549. Diese Verordnung stellt sicher, dass die rechtlichen Vorgaben in die Praxis umgesetzt werden und ist direkt auf die Entwicklung barrierefreier Webanwendungen anwendbar (vgl. [Bun24], [DIA24]). Darüber hinaus schreibt die BITV 2.0 Erläuterungen in leichter Sprache und Gebärdensprache, sowie eine Erklärung zur Barrierefreiheit vor, die auf den Ergebnissen eines BITV-Tests basieren kann (vgl. [DIA24]).

DIN EN 301 549

Eine zentrale technische Norm für die Barrierefreiheit in Deutschland ist die DIN EN 301 549. Sie legt umfassende Anforderungen für die Barrierefreiheit von Informations- und Kommunikationstechnologien (IKT) fest (vgl. [Bund]). Die Norm integriert wesentliche Vorgaben der Web Content Accessibility Guidelines 2.1 und wird von öffentlichen Stellen herangezogen, um sicherzustellen, dass die digitale Barrierefreiheit den Barrierefreiheitsanforderungen entsprechen. Die BITV 2.0 verweist explizit auf diese Norm, wodurch sie als verbindlicher Standard für Barrierefreiheit im öffentlichen Sektor gilt (vgl. [Bund]).

Web Content Accessibility Guidelines

International wird die barrierefreie Gestaltung von Webinhalten durch die Web Content Accessibility Guidelines (WCAG) geregelt, welche vom World Wide Web Consortium entwickelt werden. Die WCAG 2.1, welche in Deutschland als Maßstab dient, basiert auf den vier Prinzipien der Wahrnehmbarkeit, Bedienbarkeit, Verständlichkeit und Robustheit (Perceivable, Operable, Understandable, Robust – POUR) (vgl. [Wor]). Diese Prinzipien stellen sicher, dass Webinhalte für alle Nutzergruppen zugänglich und nutzbar sind - unabhängig von deren individuellen Einschränkungen.

EU-Richtlinie 2016/2102

Auf europäischer Ebene ist die EU-Richtlinie 2016/2102 zur Barrierefreiheit von Websites und mobilen Anwendungen öffentlicher Stellen von Bedeutung. Diese Richtlinie verpflichtet alle öffentlichen Stellen in den EU-Mitgliedstaaten, ihre digitalen Angebote barrierefrei zu gestalten (vgl. [Eur16]).

Diese rechtlichen und normativen Rahmenbedingungen bilden den Kern der Anforderungen an die Barrierefreiheit im Web in Deutschland. Sie stellen sicher, dass digitale Angebote für alle Menschen zugänglich sind und setzen verbindliche Standards für Entwickler, Anbieter und öffentliche Stellen. Die konsequente Anwendung dieser Vorgaben ist entscheidend, um die digitale Teilhabe aller Bürger zu gewährleisten und Diskriminierung aufgrund von Behinderungen im digitalen Raum zu verhindern.

2.3 Grundlegende Technologien für Barrierefreiheit

Die Umsetzung barrierefreier Webanwendungen erfordert den gezielten Einsatz von Technologien, um die Zugänglichkeit für alle Nutzergruppen sicherstellen. Dabei spielen semantisches HTML und die WAI-ARIA-Suite eine zentrale Rolle. Semantisches HTML sorgt durch den korrekten Einsatz von strukturellen und beschreibenden Elementen für eine klare und maschinenlesbare Darstellung von Inhalten. Ergänzend ermöglicht WAI-ARIA die semantische Anreicherung von HTML-Elementen, um dynamische und komplexe Inhalte besser zugänglich zu machen.

2.3.1 Semantisches HTML

HTML bildet die Grundlage für den Inhalt und die Struktur einer Webseite und sorgt bei korrekter Umsetzung dafür, dass die Semantik des Inhalts maschinenlesbar ist. Dies ist wesentlich für die Zugänglichkeit, die Suchmaschinenoptimierung und die optimale Nutzung der browserseitigen Funktionen (vgl. [Moz24]). Entwickler neigen dazu * *-Elemente als Block-Container zu verwenden. Im Hinblick auf die Barrierefreiheit und Suchmaschinenoptimierung der Webseite ist dies jedoch nicht ratsam. Besser ist es die verfügbaren strukturellen, semantischen Elemente, wie beispielsweise *<main>*, *<section>*, *<article>*, *<header>*, *<nav>* und *<footer>* zu verwenden (siehe Listing 2.1) um die Barrierefreiheit der Webseite zu gewährleisten (vgl. [Moz24]).

```html
<body>
  <header>
    <nav>
      <a href="#section1">Abschnitt 1</a>
      <a href="#section2">Abschnitt 2</a>
    </nav>
    <h1>Beispielseite</h1>
  </header>

  <main>
    <section id="section1">
      <header>
        <h2>Einfuehrung</h2>
      </header>
      <p>Einleitende Informationen zum Thema.</p>
    </section>

    <section id="section2">
      <header>
        <h2>Hauptinhalt</h2>
      </header>
      <article>
        <header>
          <h3>Details zum Thema</h3>
        </header>
        <p>Vertiefte Informationen und Analysen.</p>
      </article>
    </section>
  </main>

  <footer>
    <p>&copy; 2024 Beispielseite</p>
  </footer>
</body>
```

Listing 2.1: Seitenaufbau mit semantischen HTML-Elementen

Die Nutzung dieser semantischen Elemente unterstützt und optimiert die strukturelle Navigation und macht diese auch für ATs[1] zugänglich (vgl. [Vie15]).

2.3.2 WAI-ARIA

Die WAI-ARIA Suite stellt ein Framework bereit, welches es ermöglicht, Attribute zur Kennzeichnung von Benutzerinteraktionen hinzuzufügen, deren wechselseitige Beziehungen zu definieren, sowie ihren aktuellen Zustand zu beschreiben (vgl. [Nur24]). Dies geschieht durch das Erweitern der HTML-Attribute mit Rollen-, Eigenschafts- und Zustandsinformationen. Widgets, Strukturen und Verhaltensweisen einer Webanwendung können durch das Hinzufügen der ARIA-Attribute aus semantischer Sicht beschrieben werden (vgl. [Fog14]). Die Rollen und Eigenschaften erlauben es den ATs die verschiedene Zustände von Widgets (deaktiviert, ausgewählt, markiert, etc.) zu erkennen und an deren Nutzer zu kommunizieren. Eine Rich Internet Application (RIA) wird durch den Einsatz von WAI-ARIA leichter bedienbar, da diese wichtige strukturelle Informationen bereitstellen, welche eine vollständige und effektive Tastaturnavigation ermöglichen. Zudem verbessert die Zuordnung von Rollen die Verständlichkeit der Webseite, da assistive

1 AT (Assistive Technologien) sind Hilfsmittel wie Screenreader, die Menschen mit Behinderungen den Zugang zu Webseiten erleichtern. Durch die Nutzung semantischer HTML-Elemente wird die Navigation für ATs optimiert und die Inhalte maschinenlesbar gemacht.

Technologien die einzelnen Elemente, basierend auf ihrer Funktion sowie deren Rollen, Zustände und Eigenschaften interpretieren und verarbeiten können (vgl. [Fog14]).

Zusätzlich erlaubt der Einsatz von ARIA-Rollen, dass vorhandene Rollen des Markups überschrieben werden. Dies ermöglicht es Elemente, welche aus gestalterischen Gründen nicht korrekt ausgezeichnet worden sind, semantisch korrekt zu beschreiben ohne dabei das Layout der Webanwendung zu modifizieren (vgl. [Vie15]). Ein Beispiel aus der Praxis ist der Seitenaufbau ohne semantische HTML-Elemente, der stattdessen auf ARIA-Attribute setzt. Wie in Listing 2.2 zu sehen ist, erhält das -Element mit der Klasse *header* die Rolle *banner*, um den Kopfbereich der Seite für Assistenztechnologien als solchen zu kennzeichnen. Innerhalb dieses Bereichs befindet sich ein weiteres -Element mit der Rolle *navigation* und dem Attribut *aria-label="Hauptnavigation"*. Das Attribut *aria-label* gibt eine beschreibende Kennzeichnung für den Navigationsbereich an, was Screenreadern den Einstieg in die Navigation erleichtert. Die Hauptüberschrift <h1> wird durch die Angabe der Rolle *heading* und des Attributs *aria-level="1"* als Überschrift ersten Grades definiert. Diese Angabe stellt sicher, dass die Bedeutung und Hierarchie der Überschrift korrekt erkannt wird, auch ohne die explizite Verwendung eines <h1>-Tags.

```
<body>
  <div role="banner">
    <div role="navigation" aria-label="Hauptnavigation">
      <a href="#section1">Abschnitt 1</a>
      <a href="#section2">Abschnitt 2</a>
    </div>
    <h1>Beispielseite</h1>
  </div>

  <div role="main">
    <div role="region" aria-labelledby="section1-heading">
      <h2 id="section1-heading">Einfuehrung</h2>
      <p>Einleitende Informationen zum Thema.</p>
    </div>

    <div role="region" aria-labelledby="section2-heading">
      <h2 id="section2-heading">Hauptinhalt</h2>
      <div role="article" aria-labelledby="article-heading">
        <h3 id="article-heading">Details zum Thema</h3>
        <p>Vertiefte Informationen und Analysen.</p>
      </div>
    </div>
  </div>

  <div role="contentinfo">
    <p>&copy; 2024 Beispielseite</p>
  </div>
</body>
```

Listing 2.2: Seitenaufbau ohne semantische HTML-Elementen, aber mit der Verwendung von ARIA-Attributen

Der Hauptinhalt der Seite wird durch ein -Element mit der Rolle *main* gekennzeichnet, welches den zentralen Inhalt umschließt. Innerhalb dieses Hauptbereichs befinden sich weitere -Elemente mit der Rolle *region*, welche durch das *aria-labelledby*-Attribut jeweils mit den IDs der zugehörigen Überschriften (<h2>) verknüpft sind. Diese Verknüpfung gibt den *region*-Elementen eine thematische Kennzeichnung und erleichtert so die Strukturierung der Seite für AT-Nutzer.

Der Fußbereich wird durch ein -Element mit der Rolle *contentinfo* definiert. Diese Rolle zeigt an, dass der Inhalt dieses Bereichs zusätzliche Informationen über die Seite enthält, wie z.B. Urheberrechtsinformationen.

Durch die Verwendung dieser ARIA-Attribute wird der strukturelle Aufbau der Seite für Screenreader und andere assistive Technologien nachvollziehbar, auch wenn semantische HTML-Elemente nicht verwendet werden. Diese Methode stellt sicher, dass alle Bereiche klar erkennbar sind und auch ohne die Verwendung von <header>, <nav>, <main>, <section>, <article> und <footer> eine zugängliche Struktur geschaffen wird.

2.4 Testmethoden zur Bewertung der Barrierefreiheit

Die Autoren *Graap et al.* stellen in ihrer Arbeit fest, dass der Grad der Barrierefreiheit für Menschen ohne Beeinträchtigungen, sowohl in der Entwicklungsphase als auch bei der Nutzung einer Webseite, meist schwer erkennbar ist. Da Tests nicht immer von, oder mit betroffene Personen durchgeführt werden können, verweisen die Autoren auf die Notwendigkeit systematischer Methoden und Werkzeuge zur Überprüfung der Barrierefreiheit (vgl. [Gra18]).
Tests zur Barrierefreiheit lassen sich in manuelle und automatisierte Verfahren unterteilen. Manuelle Tests erfordern eine direkte Überprüfung durch Testpersonen, oder Entwickler. Automatisierte Tests hingegen nutzen spezielle Tools, die Webseiten auf gängige Barrierefreiheitsrichtlinien prüfen und Fehler automatisch erkennen. Im Folgenden werden die beiden Ansätze näher beschrieben.

2.4.1 Automatisierte Prüfmethoden

Bereits während der Entwicklung kann der Quellcode auf grundlegende Verstöße, oder Fehler hinsichtlich der Zugänglichkeit geprüft werden. Dies geschieht durch sogenannte Validatoren. Validatoren ermöglichen eine Überprüfung des Quellcodes auf die Einhaltung der jeweiligen Spezifikationen und tragen zur Entwicklung korrekter (Web)-Anwendungen bei, damit diese reibungslos mit assistiven Technologien funktionieren (vgl. [Dow19]). Ein Beispiel hierfür ist der *axe Accessibility Linter*[1], der für die Integration in Entwicklungsumgebungen, wie bspw. Visual Studio Code, entwickelt wurde. Er analysiert den Quellcode auf Verstöße gegen die WCAG-Standards (WCAG 2.0 und 2.1) und kann in Entwicklungs- und CI/CD-Umgebungen integriert werden, wodurch

1 `https://marketplace.visualstudio.com/items?itemName=deque-systems.vscode-axe-linter`

Zugänglichkeitsprobleme bereits während des Entwicklungsprozesses erkannt werden können (vgl. [Deq]).

Zusätzlich zu Lintern[1] gibt es weitere automatisierte Software und Online-Dienste, die prüfen, ob Webinhalte barrierefrei sind. Meist werden die zu prüfenden Inhalte gegen die WCAG-Standards geprüft, die von den meisten Werkzeugen unterstützt werden (vgl. [Whi23]). Die Überprüfung geschieht dabei entweder durch die Eingabe einer URL, das Einfügen des Quellcode oder das Hochladen der HTML-Datei. Die meisten Werkzeuge testen dabei einzelne Seiten. Das umfassendere Testen ganzer Anwendungen, mittels Web Crawling[2], bleibt meistens kommerziellen Tools vorbehalten. Die Testergebnisse zeigen die vorliegenden Fehler, welche konkrete Verstöße gegen die Barrierefreiheit darstellen und Warnungen, bei denen ein möglicher Verstoß vermutet wird (vgl. [Aba19]).

Automatisierte Tests können jedoch nicht alle Aspekte der Barrierefreiheit prüfen. Manchmal liefern sie ungenaue oder irreführende Ergebnisse. Sie bestimmen nicht die tatsächliche Barrierefreiheit, sondern dienen lediglich als Hilfsmittel (vgl. [Whi23]).
Eine Liste dieser Werkzeuge wird vom W3C unter https://www.w3.org/WAI/test-evaluate/tools/list/ zur Verfügung gestellt.

2.4.2 Manuelle Prüfmethoden

Manuelle Prüfmethoden können in zwei Kategorien unterteilt werden. Dazu zählen Tests durch Experten und das Testen durch eingeschränkte, oder behinderte Personen selbst - die sog. *User-Tests*. Beide Ansätze können Schwachstellen offen legen, welche die automatisierten Tests nicht finden können (vgl. [Als20]). Manuelle Tests können laut den Autoren *Graap et al.* durch vier Testmethoden abgebildet werden:

- **Bedienbarkeit mit der Tastatur**
  Hier wird geprüft, ob die Webseite sich problemlos mit der Tastatur und ohne Maus bedienen lässt. Eine gute Tastaturbedienbarkeit bietet zudem eine gute Grundlage für die Verwendung von Screenreadern (vgl. [Gra18]).

- **Skalierbarkeit der Webseite**
  Die Skalierbarkeit bezieht sich auf die Darstellung der Webseite. Genauer gesagt darauf, wie sich Texte, Bilder und das Layout verhalten, wenn diese durch Verwendung des Zoom größer dargestellt werden (vgl. [Gra18]).

1 Ein Linter ist ein Werkzeug, welches den Quellcode automatisch auf potenzielle Fehler, Verstöße oder Best Practices überprüft.
2 Web-Crawling bezeichnet das automatisierte Durchsuchen und Erfassen von Webseiten durch Programme.

- **Anzeige ohne Bilder, JavaScript und CSS**
  Mit dieser Prüfung wird herausgefunden, ob die Webseite auch ohne die Verwendung dieser Technologien noch voll funktionsfähig verwendet werden kann (vgl. [Gra18]).

- **Konformitätsprüfung mit Checkliste**
  Diese Tests prüfen anhand der Kriterien von bereitgestellten Checklisten (wie WCAG 2.2, oder BITV 2.0), ob eine Webseite barrierefrei ist (vgl. [Gra18]).

Betrachtet man den Umfang sämtlicher Erfolgskriterien gemäß WCAG 2.2[1], so wird schnell deutlich, dass das manuelle Testen um ein Vielfaches zeitintensiver ausfällt, als das Testen mittels automatisierter Verfahren (vgl. [Egg24a]). Die Unterschiede zwischen manuellen und automatisierten Prüfmethoden zeigen sich nicht nur im Zeitaufwand, sondern auch in Aspekten wie Aussagekraft, Konsistenz und Kosten. Die folgende Tabelle gibt einen strukturierten Überblick über die jeweiligen Vor- und Nachteile beider Ansätze und verdeutlicht deren Stärken und Schwächen im direkten Vergleich.

Kriterium	**Manuell**	**Automatisiert**
Abdeckung der Erfolgskriterien	Hoch	Mäßig
Aussagekraft	Hoch	Begrenzt
Benötigtes Vorwissen	Hoch	Gering
Konsistenz der Ergebnisse	Variabel	Hoch
Skalierbarkeit	Gering	Hoch
Erkennung subjektiver Probleme	Möglich	Nicht möglich
Zeitaufwand	Hoch	Gering
Kosten	Hoch	Niedrig bis moderat

Tabelle 2.1: Vergleich von manuellen und automatisierten Prüfmethoden.

1 https://www.w3.org/WAI/WCAG22/quickref/?showtechniques=pageinfo

3 Konzept

Das vorliegende Kapitel beschreibt die konzeptionellen Grundlagen, die zur Entwicklung eines Testkonzepts für barrierefreie Webanwendungen notwendig sind. Ziel ist es, ein Verständnis der Anforderungen und Herausforderungen bei der Umsetzung von Barrierefreiheit zu schaffen, um darauf basierend ein Testkonzept zu entwickeln. Hierbei werden sowohl bestehende Standards als auch technische und organisatorische Aspekte berücksichtigt. Im Folgenden werden zunächst die Anforderungen an barrierefreie Webseiten definiert, bevor betriebliche und technische Herausforderungen sowie die spezifischen Anforderungen an das Testkonzept näher beleuchtet werden.

3.1 Anforderungen barrierefreier Webseiten

Die Anforderungen an die Barrierefreiheit von Webanwendungen basieren auf den vier zentralen Prinzipien der Wahrnehmbarkeit, Bedienbarkeit, Verständlichkeit und Robustheit (Perceivable, Operable, Understandable, Robust - POUR-Prinzipien), welche im Rahmen der WCAG definiert sind. Die Einhaltung dieser Prinzipien ist entscheidend um sicherzustellen, dass alle Nutzer - einschließlich Menschen mit Behinderungen - die digitalen Inhalte barrierefrei nutzen können (vgl. [Wor]). Die spezifischen Anforderungen in diesen Bereichen werden im Folgenden dargelegt.

3.1.1 Wahrnehmbarkeit

Die Darstellung von Informationen und Benutzeroberflächen muss so gestaltet sein, dass sie von den Nutzenden wahrgenommen werden kann (vgl. [Wor]). Dabei ist es wichtig, dass für Nicht-Text-Inhalte, wie Bilder oder Grafiken, textliche Alternativen bereitgestellt werden. Diese Alternativtexte ermöglichen, dass visuelle Inhalte von Screenreadern oder Braille-Geräten[1] korrekt verarbeitet und ausgegeben werden können (vgl. [Wor24b]).

1 https://de.wikipedia.org/wiki/Braillezeile

Als konkretes Beispiel hierfür führt *Abou-Zahra* an, dass „Textalternativen den Zweck eines Bildes oder einer Funktion vermitteln, um eine äquivalente Nutzererfahrung zu gewährleisten. So wäre eine geeignete Textalternative für einen Suchbutton *Suche* anstatt *Lupe*"([Wor24b]).

Darüber hinaus ist es notwendig, dass die Webseite responsive gestaltet ist. Das dient nicht nur der besseren Zugänglichkeit mit verschiedenen Endgeräten, sondern erlaubt vor allem die individuelle Anpassung an die jeweiligen Bedürfnisse der Nutzer. Dies setzt eine korrekte strukturelle Auszeichnung von Überschriften, Listen, Tabellen und Eingabefeldern voraus. Eine solche Auszeichnung ermöglicht Anpassungen der Darstellung durch Vergrößerungen, alternative Farbkontraste oder Veränderungen der Textgröße. Die korrekte und vollständige Wiedergabe des Inhaltes muss selbst bei einer Vergrößerungen des Textes auf bis zu 400 %, sowie der Anpassung des Text- und Zeilenabstandes gegeben sein (vgl. [Wor24b]).

Zusätzlich ist bei der Einbindung von Audio- und Videoinhalten darauf zu achten, dass Alternativen in Form von Untertiteln, Audiobeschreibungen oder Zeichensprache bereitgestellt werden (vgl. [Wor24b]).

3.1.2 Bedienbarkeit

Um die Bedienbarkeit einer Webseite sicherzustellen muss diese, insbesondere für Nutzer, die keine Maus verwenden, vollständig mit der Tastatur bedienbar sein. Das bedeutet, dass alle Funktionen, welche mit einer Maus verfügbar sind, ebenso auf der Tastatur zur Verfügung stehen. Hierdurch wird sowohl die Steuerung mittels Spracherkennung als auch die Nutzung alternativer Tastaturen, wie Bildschirmtastaturen, unterstützt. Die Bedienbarkeit wird zusätzlich durch eine gut durchdachte Seitenstruktur erleichtert. Zudem sollte die Tastatursteuerung durch eine sichtbare Fokus-Anzeige begleitet werden (vgl. [Wor24b]).

Zusätzliche Erfolgskriterien zur Bedienbarkeit verlangen, dass den Nutzenden eine angemessene Reaktionszeit zur Verfügung steht. Dies wird durch Optionen gewährleistet, die es ermöglichen, Zeitlimits zu stoppen, zu verlängern oder anzupassen, sodass die Inhalte in einem individuellen Tempo genutzt werden können (vgl. [Wor24b]). Darüber hinaus adressieren weitere Kriterien der Bedienbarkeit die Vermeidung von Anfällen und anderen körperlichen Reaktionen, die durch bestimmte visuelle Effekte ausgelöst werden könnten (vgl. [Wor24b]).

3.1.3 Verständlichkeit

Inhalten und Informationen müssen so gestaltet sein, dass sie für ein breites Spektrum an Nutzern lesbar und verständlich sind und durch Text-to-Speech-Technologien korrekt wiedergegeben werden. Darüber hinaus sollte die Benutzeroberfläche konsistent und vorhersehbar gestaltet sein, um die Orientierung und Bedienung zu erleichtern. Ebenso ist es bei komplexeren Interaktionen, insbesondere bei Formularen, notwendig, dass Mechanismen zur Fehlervermeidung und -korrektur vorgesehen sind (vgl. [Wor24b]).

3.1.4 Robustheit

Die Anwendung muss so gestaltet sein, dass sie von aktuellen und zukünftigen Technologien, Browsern und ATs zuverlässig interpretiert werden kann. Dies wird durch eine korrekte Verwendung der Auszeichnungssprache gewährleistet, wodurch strukturelle und semantische Elemente zuverlässig interpretiert werden können. Zusätzlich sollen bei nicht standardisierten Komponenten eindeutige Namen und Rolle definiert werden, um ihre Funktionalität klar zu kennzeichnen (siehe Abschnitt 2.3.2) (vgl. [Wor24b]).

Diese Anforderungen gewährleisten, dass die Webanwendung nicht nur den Vorgaben der Web Content Accessibility Guidelines entspricht, sondern auch eine möglichst hohe Zugänglichkeit und Bedienfreundlichkeit für eine breite Nutzergruppe sicherstellt. Sie bilden die Grundlage für die weitere Implementierung und Prüfung der Barrierefreiheit in dieser Arbeit.

Eine umfassende Übersicht aller Erfolgskriterien ist unter folgendem Link verfügbar: `https://www.w3.org/WAI/WCAG22/quickref`.

3.2 Betriebliche und technische Herausforderungen bei der Implementierung von Barrierefreiheit

Trotz der Verbesserung von Entwicklungstools und der Einführung zahlreicher Richtlinien, Gesetze und Standards bleiben viele Webanwendungen weiterhin nur teilweise barrierefrei. Ein wesentlicher Grund dafür sind die hohen Kosten, die bei der Implementierung anfallen. Nach Ansicht der Autoren *Puzis et al.* liegt das Problem darin, dass ARIA nicht als ganzheitliche Lösung konzipiert wurde, sondern lediglich als Ergänzung zu den bestehenden Möglichkeiten in HTML und JavaScript. Dies führt zu einem komplexen und oft ineffizienten Entwicklungs- und Wartungszyklus, zusätzlich erschwert von dem kontinuierlichen Wachstum des Internets (vgl. [Puz15]).

Die Komplexität wird deutlich bei der Betrachtung der Arbeitsschritte, die notwendig sind, um ein einfaches Login-Dialogfenster barrierefrei zu gestalten. Als erstes muss der Nutzer darüber informiert werden, dass ein Dialogfenster geöffnet wurde und der Fokus muss auf das relevante Eingabefeld, wie bspw. *E-Mail* gesetzt werden. Dieses Feld sollte als bearbeitbar gekennzeichnet sein. Alle Eingabefelder und Schaltflächen, wie etwa *E-Mail*, *Passwort*, *Anmelden* und *Abbrechen*, benötigen aussagekräftige Labels. Die Reihenfolge der Navigation, welche über die Tabulator-Taste gesteuert wird, muss eine sinnvolle, kreisförmige Navigation erlauben. Zusätzlich muss verhindert werden, dass der virtuelle Cursor aus dem Dialogfenster heraus bewegt wird, um einen Orientierungsverlust zu vermeiden. Die Formularfelder sollten mit Sprachausgabe überprüfbar sein, damit Eingabefehler unmittelbar bemerkt werden. Um sicherzustellen, dass der Benutzer nach dem Schließen des Dialogfensters nicht die Orientierung verliert, sollte der Cursor (bzw. der Fokus) an die Ausgangsposition zurückkehren (vgl. [Puz15]). Daher argumentieren die Autoren *Puzis et al.*, dass die Implementierung der Barrierefreiheit bei benutzerdefinierten Widgets erheblich höher ausfallen könnte. Für die Senkung dieser Kosten schlagen die Autoren einerseits vor, dass das Konzept von ARIA so überarbeitet wird, dass es sowohl für Webentwickler als auch für die Entwickler von Screenreadern und Browsern praktikabel ist. Zweitens soll ARIA um neue Kategorien erweitert werden, welche nicht nur für Menschen mit bleibenden Beeinträchtigungen, sondern auch für kurzfristig eingeschränkte Nutzer hilfreich sind (vgl. [Puz15]).

Durch die Nutzung von Standards und geräteunabhängigen Technologien lässt sich zudem eine kostengünstigere Bereitstellung und Wartung erreichen. Bei der Gestaltung neuer Webanwendungen ist es ratsam, zunächst eine komplett barrierefreie Basisversion zu entwickeln und erst im Anschluss daran zusätzliche Interface-Elemente zu ergänzen, welche die Zugänglichkeit potenziell einschränken könnten. Die Webseite bleibt dadurch auch ohne diese Technologien im Wesentlichen nutzbar, selbst wenn einige komfortable Funktionen oder ästhetische Elemente fehlen. Frühere Bedenken, dass die Barrierefreiheit die Gestaltungsfreiheit einschränke, sind laut *Thesmann* heute widerlegt (vgl. [The16]). *Thesmann* zieht einen Vergleich zwischen den Kosten für die barrierefreie Gestaltung von Webanwendungen und den Aufwendungen für den barrierefreien Umbau eines Gebäudes. Er stellt fest, dass das günstigste Verhältnis zwischen Aufwand und Kosten bereits in der frühen Konzeptionsphase erreicht wird (vgl. [The16]). Die Kosten steigen vor allem dann deutlich, wenn eine bestehende Anwendung nachträglich angepasst werden muss. Während zusätzliche Aufwendungen nur in bestimmten Bereichen erforderlich sind, können sie, etwa für die Bereitstellung von Untertiteln, Gebärdensprache oder Hörfassungen bei Videos, beträchtlich ausfallen (vgl. [The16]).

Ein weiterer, nicht zu vernachlässigender, Kostenpunkt entsteht durch das Testen und ggfs. Zertifizieren der Barrierefreiheit einer (Web-)Anwendung. Das Testen durch Experten ist nicht nur zeitintensiv, sondern auch kostenintensiv. Die Preise variieren

je nach Komplexität der Webseite. Das Testen einer einfachen HTML-Seite, durch Experten, kostet beispielsweise ca. 420–450 Euro, während umfangreiche oder komplexe Seiten mit dynamischen Elementen zwischen 840 und 900 Euro liegen können (vgl. [DIA]). Die Schweizer Experten der Stiftung *Zugang für alle*[1] stützen ihr Überprüfungsverfahren auf die Richtlinien der WCAG 2.1. Die Tests werden von ihren Accessibility-Spezialisten durchgeführt, welche selbst mit einer Behinderung leben. Für die Zertifizierung einer komplexen Webanwendung fallen hier 8.500 CHF an, was etwa 9.065 EUR entspricht (vgl. [Zug24]). Andere Quellen nennen ähnliche, teilweise sogar höhere Preisspannen für komplexere Websites. Laut einer Einschätzung von *Aktion Mensch* können sich die Kosten für ein umfassendes Barrierefreiheits-Audit komplexer Webseiten oder Online-Shops auf 5.000 bis 10.000 Euro belaufen (vgl. [Akt]). Das Unternehmen *Barrierefreiheit Umsetzen Malekzadeh & Franzke GbR* spricht bei sehr anspruchsvollen oder umfangreichen Shops sogar von 8.000 bis 20.000 Euro (vgl. [Bar]). Die Angaben für einfachere Webseiten unterschieden sich nur geringfügig und liegen in der Regel unter 5.000 Euro. (vgl. [Bar], [Akt], [Zug24]). Um diese Kosten in einen personellen Kontext zu setzen, ist das durchschnittliche Bruttojahresgehalt eines Webentwicklers in Deutschland zu betrachten. Dieses liegt bei etwa 41.800 Euro (Median) (vgl. [The]), was umgerechnet rund 3.483 Euro pro Monat (bei 12 Monatsgehältern) oder ca. 160 Euro pro Arbeitstag (bei rund 21,7 Arbeitstagen/Monat) entspricht. Tests durch externe Experten kosten die Unternehmen also im Vergleich etwa soviel, wie ein angestellter Webentwickler für einen ganzen Monat oder, je nach Komplexität, sogar mehr. Ob diese Kosten als hoch oder niedrig empfunden werden, hängt somit stark vom Projektumfang, der Unternehmensgröße und der Häufigkeit solcher Prüfungen ab.

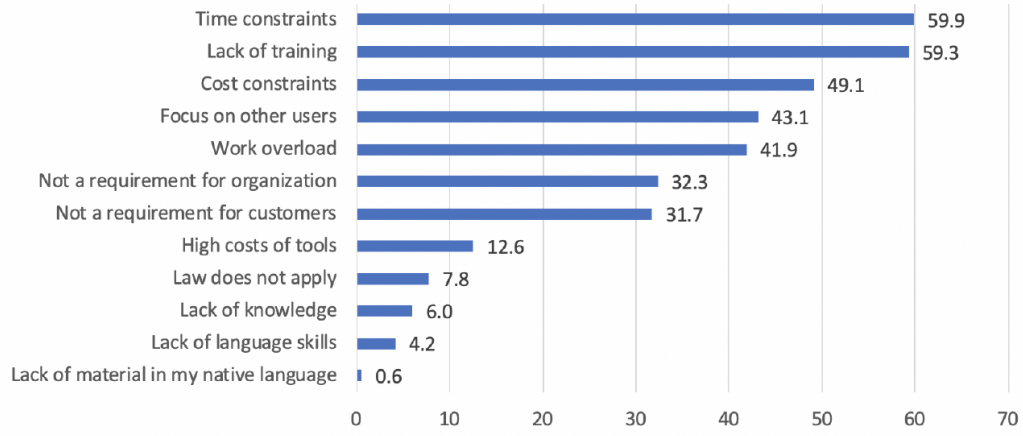

Abbildung 3.1: Herausforderungen bei der Implementierung eines barrierefreien Systems (in Prozent) [Ina20]

In einer Umfrage, welche an User-Experience-Experten in nordischen Ländern gerichtet war, fanden die Autoren *Inal et al.* heraus, dass vor allem zeitliche Einschränkungen,

1 https://access-for-all.ch/

fehlende Schulungsmöglichkeiten und finanzielle Begrenzungen als größte Herausforderungen bei der Entwicklung barrierefreier Systeme bestehen (siehe 3.1). Weitere häufig genannte Hindernisse umfassen die Fokussierung auf andere Nutzergruppen, eine hohe Arbeitsbelastung sowie das Fehlen entsprechender Anforderungen seitens der Organisationen und Kunden (vgl. [Ina20]).

Es lässt sich feststellen, dass die Implementierung von Barrierefreiheit in Webanwendungen sowohl betriebliche als auch technische Herausforderungen mit sich bringt. Trotz zahlreicher Standards und verbesserter Entwicklungstools bleibt die Barrierefreiheit vieler Anwendungen unvollständig, vor allem aufgrund der hohen Implementierungskosten und Komplexität. Eine kosten- und nutzerfreundlichere Lösung könnte durch eine Überarbeitung der ARIA-Spezifikationen, sowie durch den Einsatz geräteunabhängiger Technologien erarbeitet werden. Zusätzlich erweisen sich die Kosten für Tests und Zertifizierungen als hoch, was den Prozess weiter erschwert. Zeitliche und finanzielle Beschränkungen sowie ein Mangel an Schulungsangeboten und Anforderungen seitens der Organisationen stellen dabei weitere Hindernisse dar, welche die Implementierung der Barrierefreiheit einschränken.

Die Schulungssituation zeigt, dass offizielle Schulungsangebote rar sind. Die *Nestler UUX Consulting GmbH* bietet ein 40-stündiges Online-Training, für 490 €, zur WCAG 2.2 an. Dieses umfasst 13 Lektionen und zielt auf ein grundlegendes Verständnis der Richtlinien ab (vgl. [Nes]). Öffentliche Stellen, wie die *Hessische Zentrale für Datenverarbeitung* (HZD) sowie *ekom21* bieten ebenfalls Schulungen an, jedoch sind spezifische, umfangreich zertifizierte Programme für Webentwickler schwer zu finden. Stattdessen existieren kürzere Trainings, wie etwa ein dreistündiger Kurs zum Verständnis der EN 301 549, der für rund 200 € pro Teilnehmer angeboten wird (vgl. [eko]). Dies entspricht in etwa dem Tagessatz eines Webentwicklers, bietet jedoch nur ein komprimiertes Schulungsangebot mit begrenztem thematischem Tiefgang.

3.3 Anforderungen an das Test-Konzept

Anforderungen beschreiben, welche Eigenschaften, Funktionen und Rahmenbedingungen ein System oder Konzept erfüllen muss. Akzeptanzkriterien definieren messbare und überprüfbare Indikatoren, an denen erkennbar wird, ob eine Anforderung vollständig und korrekt umgesetzt wird. Die hier dargestellten Anforderungen an das zu entwickelnde Test-Konzept werden aus den erarbeiteten Informationen der vorangegangenen Kapitel abgeleitet. Die genannten Anforderungen und deren Akzeptanzkriterien dienen als Leitfaden, um nach der Umsetzung des Test-Konzepts überprüfen zu können, ob die angestrebten Ziele erreicht und die identifizierten Hindernisse (siehe Abschnitt 3.2) erfolgreich adressiert werden.

Zur besseren Übersicht sind die funktionalen Anforderungen mit dem Buchstaben 'A' und die nicht-funktionalen Anforderungen mit dem Buchstaben 'B' gekennzeichnet. Die Nummerierung dient der eindeutigen Referenzierung.

3.3.1 Funktionale Anforderungen

A1: Kontinuierliche Prüfung der Barrierefreiheit im Entwicklungsprozess
Das Test-Konzept soll sicherstellen, dass Barrierefreiheitsprüfungen kontinuierlich erfolgen, um frühzeitig Probleme zu erkennen.

- **A1.1:** Bei jedem neuen Build (z. B. Push in das Repository) werden automatisierte Barrierefreiheits-Tests in der CI/CD-Pipeline ausgeführt.
- **A1.2:** Entwickler erhalten nach jeder Testausführung ein automatisiertes Feedback, welches potenzielle Verstöße gegen die WCAG-Richtlinien aufzeigt.
- **A1.3:** Der Test schlägt fehl, wenn Verstöße erkannt werden, um deren Behebung zu erzwingen.

A2: Nahtlose Integration in bestehende Entwicklungsumgebungen
Das Test-Konzept muss sich nahtlos in bereits vorhandene Build-, Test- und Entwicklungstools integrieren lassen, um zusätzlichen Aufwand und Kosten zu minimieren.

- **A2.1:** Das Testkonzept ist mit den gängigen Frontend-Framework (z. B. Angular, React oder Vue) kompatibel.
- **A2.2:** Die Tests können über Standard-Test-Runner (z. B. Jest, Vitest) ausgeführt werden.
- **A2.3:** Die Integration erfolgt über einfache Konfigurationsänderungen oder minimale Code-Anpassungen, um den Einführungsaufwand gering zu halten.

A3: Zentrale Dokumentation der Testergebnisse
Die Testergebnisse sollen zentral dokumentiert werden, um den Bearbeitungsstatus und Fortschritt nachvollziehbar zu machen.

- **A3.1:** Die Testresultate werden in einer strukturierten Form (z. B. JSON, HTML) gespeichert.
- **A3.2:** Für jede getestete Komponente oder Seite ist ersichtlich, ob Barrierefreiheitsverstöße vorliegen und welche Maßnahmen zur Behebung empfohlen werden.

3.3.2 Nicht-funktionale Anforderungen

B1: Kosten- und Aufwandsreduktion in Implementierung und Wartung
Aufgrund der hohen Kosten für Barrierefreiheitsmaßnahmen soll das Test-Konzept möglichst kosteneffizient sein, um Investitionshürden zu senken.

- **B1.1:** Verwendung von Open-Source- oder kostenfreien Test-Tools, um zusätzliche Lizenzkosten zu vermeiden.
- **B1.2:** Die Konfiguration und Wartung des Test-Konzepts erfordern kein spezielles Expertenwissen, sodass der Schulungsaufwand gering bleibt.

3.4 Entwicklung des Testkonzepts

Dieses Kapitel stellt ein Testkonzept vor, das darauf abzielt, die Barrierefreiheit moderner Webanwendungen systematisch und nachhaltig zu gewährleisten. Ausgehend von den im vorherigen Abschnitt definierten Anforderungen (siehe Abschnitt 3.3) werden im Folgenden konkrete Ansätze und Methoden erläutert, um eine kontinuierliche Überprüfung der Barrierefreiheit in den Entwicklungsprozess zu integrieren.

Hierzu wird zunächst das Testkonzept selbst vorgestellt. Anschließend erfolgt ein Vergleich automatisierter Testwerkzeuge, um eine fundierte Auswahl für die einzusetzende Technologien treffen zu können. Abschließend wird die Wahl passender Technologien, welche sowohl kosteneffizient als auch einfach in bestehende Entwicklungsumgebungen integrierbar sind, erläutert.

3.4.1 Vorstellung des Testkonzepts

Das Testkonzept verfolgt mehrere Hauptziele: Es soll die Barrierefreiheit sicherstellen, indem es gewährleistet, dass die Webanwendungen den gesetzlichen und normativen Anforderungen entsprechen. Zudem soll es in den Entwicklungsprozess integriert werden, um durch die Einbindung in bestehende CI/CD-Pipelines eine kontinuierliche Überprüfung der Barrierefreiheit zu ermöglichen. Die Kombination von automatisierten und manuellen Testmethoden soll eine umfassende Abdeckung der Prüfkriterien erreichen. Außerdem soll es Entwicklern möglich sein das Konzept ohne erhebliche Mehrbelastung anwenden zu können.

Basierend auf den zuvor bestimmten Anforderungen (siehe Kapitel 3.3) ergeben sich folgende Spezifikationen des Testkonzepts: Die Auswahl der Tools und Verfahren muss eine Integration in bestehende Entwicklungsumgebungen und -prozesse ermöglichen. Die

Integration des Testkonzepts in den Entwicklungsprozess erfolgt durch die Einbindung der Tests in die CI/CD-Pipeline, bzw. Unit-Tests, sodass die Barrierefreiheit bei jedem Build[1] überprüft wird. Bereiche, welche nicht automatisiert getestet werden können, müssen durch manuelle Tests abgedeckt werden. Während dem aktiven Entwicklungsprozess wird ein Linter[2] eingesetzt. Dieser weist bereits beim Schreiben des Quellcodes auf mögliche Barrierefreiheitsverstöße hin, sodass diese direkt behoben werden können. In festgelegten Intervallen oder Meilensteinen müssen manuelle Tests durchgeführt werden, um jene Aspekte zu prüfen, die nicht durch automatisierte Tests abgedeckt sind.

Die Auswahl geeigneter Testwerkzeuge ist entscheidend für den Erfolg des Testkonzepts. Kriterien wie die Kompatibilität mit einer möglichst großen Auswahl an Technologien und Frameworks, Aktualität und Wartung der Tools, sowie die Lizenzierung spielen dabei eine Rolle. Für die automatisierten Tests werden Werkzeuge wie *axe-core*, *jest-axe*, *WAVE* und *Pa11y* in Betracht gezogen, da diese in CI/CD-Pipelines integrierbar sind und aktuelle Standards unterstützen. Für die manuellen Tests werden Methoden wie die Verwendung von Checklisten basierend auf den WCAG 2.1, Screenreader-Tests, sowie die Prüfung der Tastaturnavigation vorgeschlagen.

Ein weiterer Bestandteil des Testkonzepts ist die Qualitätssicherung, bzw. die Dokumentation der Ergebnisse. Das Konzept sieht vor, dass eine Dokumentation in Form einer Logdatei erstellt wird, welche konkrete Informationen zum Status der Barrierefreiheit, oder möglicher Mängel enthält. Diese Informationen ermöglichen es den Entwicklern die identifizierten Barrieren gezielt zu beheben.

3.4.2 Vergleich automatisierter Testwerkzeuge für Barrierefreiheit

Die Gewährleistung der Barrierefreiheit von Webanwendungen erfordert Teststrategien, die sowohl manuelle als auch automatisierte Methoden integrieren. Im Rahmen des entwickelten Testkonzepts wurden spezifische Auswahlkriterien für automatisierte Testwerkzeuge festgelegt. Diese Kriterien umfassen die Unterstützung der WCAG Richtlinien, Integrationsfähigkeit in bestehende Entwicklungsprozesse, Aktualität und Wartung des Tools, sowie die Lizenzierung. Um geeignete Werkzeuge zu identifizieren, wird die Auflistung der Web Accessibility Initiative des World Wide Web Consortiums herangezogen. Auf dieser Plattform[3] können Barrierefreiheitstools nach verschiedenen Kriterien gefiltert werden, wie beispielsweise unterstützte Richtlinien, Arten von Tests, Integrationsmöglichkeiten und Lizenzmodelle. Durch die Anwendung der definierten Kriterien wurden

1 Ein Build bezeichnet den Prozess, bei dem der Quellcode in eine ausführbare oder bereitstellbare Version umgewandelt wird.
2 Ein Linter ist ein Werkzeug, welches den Quellcode automatisch auf potenzielle Fehler, Verstöße oder Best Practices überprüft.
3 `https://www.w3.org/WAI/test-evaluate/tools/list/`

folgende Tools ermittelt: *axe-core*, *WAVE* und *IBM Equal Access accessibility-checker*. Im Zuge der Literaturrecherche ist außerdem *pa11y* ermittelt worden.

axe-core

axe-core ist eine Open-Source-JavaScript-Bibliothek zur automatisierten Prüfung der Barrierefreiheit von Webinhalten. Entwickelt von Deque Systems, unterstützt *axe-core* die WCAG-Richtlinien (2.0, 2.1 und 2.2 auf den Stufen A, AA und AAA) sowie Best Practices, um gängige Barrierefreiheitsprobleme zu identifizieren. Es kann nahtlos in verschiedene Test-Frameworks und Browser integriert werden und deckt durchschnittlich 57 % der WCAG-Verstöße automatisiert ab, während für die restlichen Prüfungen manuelle Eingriffe erforderlich sind (vgl. [Deq24a]).

Die *axe-core* Engine wird als Grundlage für Accessibility-Tools wie *Lighthouse*[1] von Google verwendet, das standardmäßig im Chrome-Browser implementiert ist. Außerdem bietet die Bibliothek erweiterte Integrationsmöglichkeiten, um Barrierefreiheitstests frühzeitig in den Entwicklungsprozess zu integrieren. Für Visual Studio Code ist mit dem *axe-linter*[2] eine Erweiterung verfügbar, die Entwicklern direktes Feedback zu potenziellen Problemen gibt (vgl. [Deq24b]).

Zusätzlich zur API bietet *axe-core* umfangreiche Anpassungsmöglichkeiten und durch die Unterstützung der Community werden über 15 verschiedene Sprachen abgedeckt. Updates werden in regelmäßigen Abständen (3-5 Monate) veröffentlicht. Diese umfassen neue Regeln und Sicherheitsaktualisierungen, wobei die Sicherheitsunterstützung für bis zu 18 Monate gewährleistet wird (vgl. [Deq24a]).

jest-axe

jest-axe ist ein Open-Source-Testwerkzeug, welches in Verbindung mit dem Test-Framework Jest[3] verwendet wird und auf der Engine von *axe-core* basiert. Es wurde als individuell konfigurierbarer Jest-Matcher entwickelt und ermöglicht die Integration der Barrierefreiheitstests in den Entwicklungsprozess (Unit-Tests, CI/CD-Pipeline), um Probleme frühzeitig zu identifizieren. Das Tool ist mit gängigen Frontend-Frameworks wie React, Vue und Angular kompatibel. Insbesondere für komponentenbasierte Frameworks wie React, Angular oder Vue bietet *jest-axe* die Möglichkeit, die *region*-Regel zu deaktivieren und dadurch Tests isolierter Komponenten zu ermöglichen (vgl. [Col24]).

1 https://developer.chrome.com/docs/lighthouse?hl=de
2 https://marketplace.visualstudio.com/items?itemName=deque-systems.vscode-axe-linter
3 https://jestjs.io/

Ein Nachteil von *jest-axe* ist jedoch, dass Farbkontrastprüfungen in JSDOM[1] nicht funktionieren und daher durch zusätzliche Prüfungen abgedeckt werden müssen (vgl. [Col24]).

WAVE

WAVE (*Web Accessibility Evaluation Tool*), entwickelt von WebAIM, ist eine Sammlung an Werkzeugen für die Evaluierung der Barrierefreiheit von Webinhalten. Webinhalte werden gegen die Richtlinien der WCAG 2.1 geprüft. Zu den frei nutzbaren Angeboten von WAVE gehören die Browser-Erweiterungen für Chrome, Firefox und Edge.

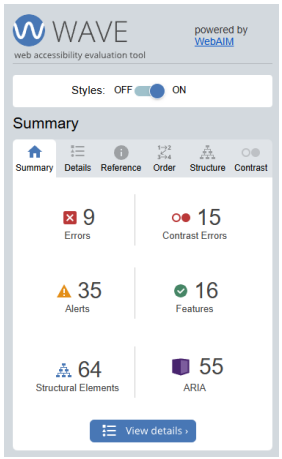

Abbildung 3.2: WAVE Browsererweiterung [Web24b].

Diese ermöglichen es, Webseiten direkt im Browser zu analysieren, ohne Daten an externe Server zu senden, was eine private und sichere Prüfung garantiert. Vorteilhaft ist, dass die Browsererweiterungen die gerenderte Version der Seite überprüfen. Dadurch können sowohl CSS-Styles, als auch dynamische Inhalte evaluiert werden (vgl. [Web24b]). Im Gegensatz zu Open-Source-Lösungen wie *axe-core* ist WAVE ein proprietäres System. Während die Browser-Erweiterungen frei nutzbar sind, erfordert die Nutzung der API eine Lizenz (vgl. [Web24c], [Web24a]).

IBM Equal Access Accessibility Checker

Der IBM Equal Access Accessibility Checker ist eine JavaScript-basierte Lösung für die Bewertung der Barrierefreiheit von Webinhalten und ein zentraler Bestandteil des IBM Equal Access Toolkits. Das Tool unterstützt die WCAG-Richtlinien (Version 2.1).

1 JSDOM ist eine JavaScript-Bibliothek. Sie bietet eine serverseitige Nachbildung des Document Object Model, um Webanwendungen oder Skripte in einer Node.js-Umgebung ohne echten Browser auszuführen. JSDOM wird häufig für Unit-Tests verwendet (vgl. [JSD24]).

Neben der Integration in Testframeworks wie Selenium, Puppeteer und Playwright ist das Tool auch für die Verwendung in CI/CD-Pipelines ausgelegt (vgl. [IBM24a]). Ein nützliches Feature des Accessibility Checkers ist die Möglichkeit, Prüfungen mit sog. *Baseline*-Daten zu vergleichen, um Fortschritte zu dokumentieren und die Konsistenz der Ergebnisse sicherzustellen. Testergebnisse können als *Baseline* gespeichert werden, sodass zukünftige Tests dann als erfolgreich gelten, wenn sie mit der *Baseline* übereinstimmen. Weichen die Ergebnisse ab, schlägt der Test fehl. Diese Funktion erweist sich als nützlich, um sog. *false positives*[1] zu identifizieren (vgl. [IBM24b]).

Pa11y

Pa11y ist eine Sammlung von Open-Source-Werkzeugen, welche die Barrierefreiheit von Webseiten prüfen und verbessern. Es werden verschiedene Tools zur Verfügung gestellt, darunter eine CLI, ein Web-Dashboard und ein JSON-basierter Webservice (vgl. [Tea24a]). Die CLI ermöglicht es einzelne Webseiten direkt über die Kommandozeile, mit Eingabe einer URL, zu testen. Die Tests basieren standardmäßig auf dem HTML_CodeSniffer[2], können jedoch auch mithilfe der *axe-core*-Engine durchgeführt werden. Pa11y unterstützt die Ausgabe von Testergebnissen in verschiedenen Formaten: JSON, CSV und HTML. Zusätzlich kann die CLI durch JSON-Konfigurationsdateien angepasst werden, was eine nahtlose Integration in bestehende Entwicklungsumgebungen ermöglicht (vgl. [Tea24d]).

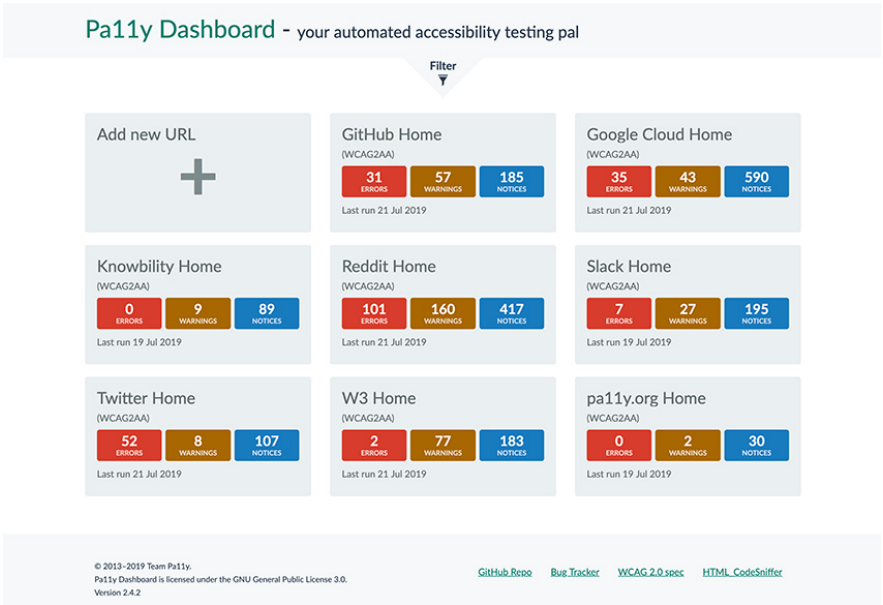

Abbildung 3.3: Pa11y Dashboard [Tea24c].

1 Fälschlicherweise als bestanden angezeigte Tests, obwohl der zugrunde liegende Code fehlerhaft ist.
2 `https://squizlabs.github.io/HTML_CodeSniffer/`

Das *Pa11y Dashboard*, ein webbasiertes Tool, visualisiert die Ergebnisse der Barrierefreiheitstest und ermöglicht es den Nutzern sich auf das Beheben der Fehler zu konzentrieren, anstatt diese zu suchen. Das Dashboard wird im Hintergrund durch den *Pa11y Webservice* unterstützt, welcher als API dient und Entwicklern die Möglichkeit bietet, eigene Dashboards oder Anwendungen anzubinden (vgl. [Tea24c]). Durch die *Pa11y CI* können diese Barrierefreiheitstest in Continuous-Integration-Umgebungen eingebunden und automatisiert werden. Für die Tests werden URLs oder Sitemaps übergeben und durch Pa11y überprüft (vgl. [Tea24b]).

	axe-core	**jest-axe**	**WAVE**	**IBM Equal Access Accessibility Checker**	**Pa11y**
WCAG-Versionen	2.0, 2.1, 2.2 (A, AA, AAA)	2.0, 2.1, 2.2 (basierend auf axe-core)	2.1	2.1	2.1 (axe-core optional)
Integrationsmöglichkeiten	Test-Frameworks, Browser, CI/CD	Jest (Unit-Tests, CI/CD)	Browser-Erweiterung, API	Selenium, Puppeteer, Playwright, CI/CD	CLI, Web-Dashboard, CI/CD
Lizenz	Open-Source	Open-Source	Proprietär (Browser-Erweiterung kostenlos, API kostenpflichtig)	Open-Source	Open-Source
Aktualität und Wartung	Regelmäßige Updates	Regelmäßige Updates	keine Angabe	Regelmäßige Updates	Regelmäßige Updates
Stärken	Umfassende Anpassbarkeit, Integration in viele Umgebungen	Nahtlose Integration in Jest, flexibel für komponentenbasierte Frameworks	Browserprüfung ohne Datenweitergabe	Baseline-Vergleich zur Reduktion von False Positives	Visualisierung der Testergebnisse, flexible Integration
Schwächen		Keine Farbkontrastprüfung in JSDOM	Lizenzkosten für API, keine Open-Source-Komponenten	Kleinere Community & Bekanntheit	

Tabelle 3.1: Vergleich automatisierter Testwerkzeuge für Barrierefreiheit

3.4.3 Ausgewählte Technologien

Im Rahmen dieser Arbeit wird *jest-axe* als zentrale Technologie zur Prüfung der Barrierefreiheit ausgewählt. Ein entscheidender Aspekt ist die direkte Kompatibilität mit den gängigen Frontend-Frameworks Angular, React und Vue (vgl. [Col24]). Diese Kompatibilität senkt die Einstiegshürde für Entwickler und Unternehmen, da bestehende Technologien und Kenntnisse ohne zusätzlichen Schulungsaufwand weiter genutzt werden können. Besonders für komponentenbasierte Architekturen (wie React und Vue) bietet *jest-axe* flexible Anpassungsmöglichkeiten, wie das Testen isolierter Komponenten.

Technisch basiert *jest-axe* auf der etablierten Engine *axe-core*. Diese gewährleistet eine hohe Zuverlässigkeit der Testergebnisse und deckt wesentliche Erfolgskriterien der Web Content Accessibility Guidelines ab. Durch die Integration als Jest-Matcher lässt sich *jest-axe* nahtlos in bestehende Entwicklungsprozesse, wie Unit-Tests oder CI/CD-Pipelines, einbinden (vgl. [Deq24a]). Ein weiterer Vorteil von *jest-axe* liegt in seiner Open-Source-Lizenzierung, wodurch weder Lizenzgebühren noch andere finanzielle Eintrittsbarrieren anfallen. Dies gewährleistet einen uneingeschränkten Zugang für Entwickler und Unternehmen, unabhängig von ihren finanziellen Ressourcen und erleichtert somit die Integration von Barrierefreiheitstests in bestehende Projekte.

Eine Limitierung von *jest-axe* liegt in der fehlenden Unterstützung für Farbkontrastprüfungen, da diese in der simulierten Umgebung von JSDOM technisch nicht umsetzbar sind (vgl. [Col24]). Um diesen Aspekt abzudecken, sollten ergänzende Werkzeuge wie die Browsererweiterung *WAVE*[1] verwendet werden. Zusätzlich wird der Einsatz eines Linters, wie *axe-Linter*[2], oder ähnlichen Tools empfohlen. Dadurch erhalten die Entwickler bereits während der Entwicklung Hinweise auf potenzielle Barrieren.

1 https://wave.webaim.org/extension/
2 https://marketplace.visualstudio.com/items?itemName=deque-systems.vscode-axe-linter

4 Realisierung

In diesem Kapitel wird die praktische Umsetzung des entwickelten Testkonzepts sowie die Integration in eine bestehende Entwicklungsumgebung beschrieben. Der Integrationsprozess umfasst dabei sowohl die technische Implementierung in die Entwicklungsumgebung als auch die Einbindung in automatisierte Abläufe wie die CI/CD-Pipeline. Zur Demonstration der Praxistauglichkeit wird die Umsetzung exemplarisch am Kundenportal der Sylphen GmbH & Co. KG analysiert. Hierbei wird aufgezeigt, wie die Barrierefreiheit einer konkreten Anwendung durch das Testkonzept bewertet und durch gezielte Anpassungen verbessert werden kann.

4.1 Sylphen GmbH & Co. KG

Die Sylphen GmbH & Co. KG[1] ist ein in Gießen ansässiges IT-Systemhaus, das sich auf die Digitalisierung von Unternehmen (deren IT-Infrastruktur) und die Softwareentwicklung auf Kundenwunsch spezialisiert hat. Mit einem Team von rund 40 IT-Netzwerk-Spezialisten und Softwareentwicklern bietet Sylphen umfassende IT-Dienstleistungen an, welche von der Konzeption und dem Betrieb von IT-Infrastrukturen über individuelle Softwareentwicklung bis hin zu Hosting- und Cloud-Lösungen reichen.

Vorstellung des Kundenportals

Das Kundenportal soll zukünftig als zentrale Plattform für die Kommunikation zwischen Kunden und dem IT-Service-Team des Unternehmens dienen. Dafür ist ein Ticket-System angebunden, welches einerseits die offenen und abgeschlossenen Tickets verwaltet, andererseits den Austausch über einen Websocket-Chat zwischen Kunden und Mitarbeitern ermöglicht. Weiterhin wird die Verwaltung gehosteter Virtueller Maschinen für die Kunden bereitgestellt.

1 `https://www.sylphen.com/`

Die technische Grundlage des Kundenportals basiert auf der folgenden Technologie- und Softwarearchitektur:

Das Frontend wird mit *React*[1], *TypeScript*[2], *Vite*[3] und *React-Bootstrap*[4] entwickelt. Die Kommunikation zwischen Frontend und Backend erfolgt über REST[5]- APIs. Das Backend ist in *Java*[6] unter Nutzung von *Spring Boot*[7] realisiert und verfolgt eine Microservice-Architektur. Als relationale Datenbank dient *PostgreSQL*[8]. Obwohl diese Technologien für die Gesamtfunktionalität des Kundenportals essenziell sind, liegt der Fokus für die Implementierung des Testkonzepts ausschließlich auf dem Frontend, da die Barrierefreiheitsprüfungen vollständig in diesem Bereich erfolgen.

Relevanz der Barrierefreiheit

Die Umsetzung der Barrierefreiheit ist eine zentrale Anforderung dieses Projekts, insbesondere im Hinblick auf die Einhaltung des Barrierefreiheitsstärkungsgesetzes, welches 2025 in Kraft tritt. Ziel ist es, das Kundenportal vollständig WCAG-konform zu gestalten, um sicherzustellen, dass alle Nutzer – einschließlich Menschen mit Behinderungen – uneingeschränkten Zugriff auf die Funktionen und Inhalte der Plattform haben. Dies umfasst die Umsetzung, bzw. Einhaltung der in der Anforderungsanalyse behandelten Merkmale.

4.2 Implementierung des Testkonzepts in eine bestehende Anwendung

In diesem Abschnitt wird die technische Umsetzung des entwickelten Testkonzepts in das oben beschriebene Kundenportal dargestellt. Es werden Tools und Frameworks erläutert, die für die Integration des Testkonzepts verwendet werden. Der Fokus liegt auf der Einrichtung der Testumgebung und Integration in den Entwicklungsprozess.

1 https://react.dev/
2 https://www.typescriptlang.org/
3 https://vite.dev/
4 https://react-bootstrap.github.io/
5 REST (Representational State Transfer) ist ein Architekturstil für Webservices. Ressourcen werden über URLs angesprochen und mit standardisierten HTTP-Methoden (z. B. GET, POST) bearbeitet, ohne dass der Server Sitzungsdaten speichern muss.
6 https://www.java.com/de/
7 https://spring.io/projects/spring-boot
8 https://www.postgresql.org/

4.2.1 Ausgangszustand der Anwendung

Um den Ausgangszustand des Projekts nachzustellen und eine einheitliche Basis zu schaffen, ist die Installation der folgenden Pakete erforderlich:

1. **Vite**[1] (im Rahmen dieser Arbeit zusammen mit React und Typescript)
2. **Vitest**[2]
3. **jest-axe**[3]

Da bei der Installation von Bibliotheken, Frameworks und Abhängigkeiten verschiedenste Probleme auftreten können, welche die Reproduzierbarkeit und Funktionalität des Projekts beeinträchtigen, wird dringend empfohlen die offiziellen Installationsanweisungen der Softwarehersteller zu befolgen.

Nachfolgend werden die notwendigen Schritte zur Installation und Einrichtung der grundlegenden Technologien beispielhaft unter Verwendung des Node Package Managers beschrieben:

1. `npm create vite@latest my-base-app - -template react-ts`
   Dieser Befehl initialisiert ein neues Projekt namens `my-base-app` mithilfe von *Vite*. Das erstellte Projekt basiert auf einer Kombination aus React und TypeScript. Die grundlegende Projektstruktur, Abhängigkeiten und Konfigurationsdateien werden dabei automatisch generiert.

2. `npm install --save-dev @testing-library/react @testing-library/dom`
   Mit diesem Befehl werden Bibliotheken installiert, welche für die Durchführung automatisierter Tests auf React-Komponenten erforderlich sind. Sie unterstützen eine nutzerzentrierte Teststrategie durch semantische DOM-Abfragen.

3. `npm install @types/react @types/react-dom`
   Dieser Befehl installiert TypeScript-Typdefinitionen für die React-Bibliotheken.

4. `npm install --save-dev jest-axe jest-environment-jsdom`
   Mit diesem Befehl werden die benötigten Bibliotheken installiert mit denen die automatisierten Tests der Barrierefreiheit realisiert werden. Dabei bildet `jest-axe` die Grundlage des in dieser Arbeit vorgestellten Test-Konzepts (siehe jest-axe).

1 https://vite.dev/guide/
2 https://vitest.dev/guide/
3 https://github.com/NickColley/jest-axe/tree/main

5. `npm install --save-dev @types/jest-axe`
   Dieser Befehl installiert die TypeScript-Typdefinitionen für die `jest-axe`-Bibliothek.

6. `npm install --save-dev vitest`
   Mit diesem Befehl wird *Vitest* installiert. *Vitest* ist ein modernes Testframework, das speziell für Vite entwickelt wurde und die bestehende Konfiguration aus der `vite.config.js` nutzt, um eine nahtlose Integration in den Entwicklungsprozess zu ermöglichen [Ant24].

Befehle, welche mit `--save-dev` annotiert sind, werden lediglich in den `devDependencies`[1] installiert, da sie nur während des Entwicklungs- und Testprozesses benötigt werden.

Nach der Installation der erforderlichen Pakete ist der grundlegende Tech-Stack für die Implementierung von Barrierefreiheitstests einsatzbereit. Wie im GitHub-Repository von *jest-axe*[2] beschrieben, kann ein erster Tests für React-Komponenten mit dem folgenden Code realisiert werden (siehe Listing 4.1):

```
1  const React = require('react')
2  const App = require('./app')
3
4  const { render } = require('@testing-library/react')
5  const { axe, toHaveNoViolations } = require('jest-axe')
6  expect.extend(toHaveNoViolations)
7
8  it('should demonstrate this matcher`s usage with react testing
       library', async () => {
9    const { container } = render(<App/>)
10   const results = await axe(container)
11
12   expect(results).toHaveNoViolations()
13 })
```

Listing 4.1: Basistest der Barrierefreiheit mit jest-axe in React [Col24]

Um die Unit-Tests auszuführen ist es erforderlich, dass in der `package.json` ein Script ergänzt wird, welches den Test-Runner startet, z. B.:

```
1  {
2    "scripts": {
3      "test": "vitest"
4    }
5  }
```

Listing 4.2: Ergänzung der package.json

1 Entwicklungsabhängigkeiten
2 https://github.com/NickColley/jest-axe/tree/main

4.2.2 Vorstellung der AccessibilityTester Klasse

Die soeben beschriebene Implementierung mit *jest-axe* bietet eine grundlegende Möglichkeit React-Komponenten auf WCAG-Verstöße zu prüfen. In größeren und komplexeren Projekten, mit einer Vielzahl Komponenten, entstehen jedoch Redundanzen in der Testlogik. Zudem fehlt eine zentrale Ergebnisdokumentation.

Basierend auf den in Kapitel 3.1 ermittelten Anforderungen wurde die im folgenden beschriebene Klasse `AccessibilityTester.ts` (siehe Listing: A.1) entwickelt.

Die Klasse `AccessibilityTester` (A.1) dient der Standardisierung und Zentralisierung von Barrierefreiheitstests für React-Komponenten. Sie minimiert Redundanzen in der Testlogik und verbessert durch die Dokumentation der Testergebnisse die Qualitätssicherung des Entwicklungsprozesses.

Die Klasse implementiert eine statische Methode `testElement`, die folgende Schritte durchführt:

1. **Rendering der Komponente**: Die übergebene React-Komponente wird mit `@testing-library/react` gerendert.
2. **Durchführung des Accessibility-Tests**: Mithilfe von `jest-axe` wird der gerenderte Output auf Verstöße gegen WCAG-Richtlinien überprüft.
3. **Aufbereitung der Ergebnisse**: Gefundene Verstöße werden in ein vereinfachtes Format transformiert, um eine übersichtliche Darstellung zu ermöglichen.
4. **Speicherung der Ergebnisse**: Die Testergebnisse werden in der Datei `accessibility-test-results.json` zentral abgespeichert. Dabei wird geprüft, ob für die getestete Komponente bereits Einträge existieren, um diese gegebenenfalls zu aktualisieren.
5. **Validierung der Testergebnisse**: Durch die Verwendung des Matchers `toHaveNoViolations` von `jest-axe` wird sichergestellt, dass der Test fehlschlägt, falls Verstöße gegen die Barrierefreiheit vorliegen.

Die Methode `testElement` akzeptiert zwei Parameter:

- `key`: Ein eindeutiger Bezeichner für die zu testende Komponente.
- `element`: Die zu testende React-Komponente.

Durch die Verwendung des Komponentennamens wird gewährleistet, dass die Ergebnisse für jede Komponente eindeutig identifiziert und bei wiederholten Tests aktualisiert werden können.

Die zentrale Speicherung der Testergebnisse in einer JSON-Datei ermöglicht eine strukturierte Dokumentation und erleichtert die Analyse, sowie die Nachverfolgung von Barrierefreiheitsproblemen über den gesamten Entwicklungszyklus hinweg. Diese Ergebnisse können zudem in weiterführende Prozesse, wie die Generierung von Berichten oder die Integration in Monitoring-Systeme, eingebunden werden.

Die Integration der `AccessibilityTester`-Klasse in bestehende Testumgebungen und CI/CD-Pipelines fördert die kontinuierliche Überwachung der Barrierefreiheit. Entwickler können die Methode `testElement` in ihren Unit-Tests einsetzen, ohne die zugrunde liegende Testlogik individuell implementieren zu müssen. Dies führt zu einer effizienteren Testentwicklung und unterstützt die Einhaltung von Barrierefreiheitsstandards bereits in frühen Phasen des Entwicklungsprozesses.

4.2.3 Integration in bestehende Unit-Tests

Die Integration der Barrierefreiheits-Tests, in bestehende Unit-Tests, erfolgt durch die Ergänzung des Methodenaufrufs:

```
1  // Barrierefreiheitstest mit AccessibilityTester
2  AccessibilityTester.testElement("ActionButton", (
3  <ActionButton items={[]} />
4  ));
```

Listing 4.3: Barrierefreiheitstest mit AccessibilityTester

Die Einbindung ist exemplarisch im Unit-Test *ActionButton.test.tsx* einer Komponente des Kundenportals dargestellt (siehe Listing A.2).

In diesem Beispiel wird die React-Komponente `ActionButton` auf Barrierefreiheitsverstöße geprüft. Die Methode `testElement` der `AccessibilityTester`-Klasse nimmt zwei Parameter entgegen:

- `ActionButton`: Ein eindeutiger Bezeichner, hier der Komponentenname.
- `<ActionButton items={[]} />`: Die zu testende React-Komponente.

Diese einfache und direkte Integration erlaubt es Entwicklern die Barrierefreiheitstests ohne großen Mehraufwand in die Testumgebung aufzunehmen. Dadurch wird die Testabdeckung hinsichtlich der Barrierefreiheit erhöht und potenzielle Verstöße können bereits während der Entwicklungsphase identifiziert und behoben werden.

4.2.4 Integration in den CI/CD-Prozess

Die automatisierte Ausführung der Unit-Tests (einschließlich der Barrierefreiheitstests) wird in den bestehenden CI/CD-Prozess integriert, um eine kontinuierliche Überwachung zu gewährleisten. Dies erfolgt durch die Konfiguration der *Client-Pipeline* in der `gitlab-ci.yml`-Datei des Projekts (siehe Listing A.4).

In der Pipeline wird die `test`-Stage definiert, welche für die Ausführung der Unit-Tests zuständig ist. Sie nutzt das Docker-Image `node:20` um eine konsistente und isolierte Umgebung für die Testausführung bereitzustellen. Der relevante Abschnitt der `gitlab-ci.yml` ist im Folgenden dargestellt:

```
1 test:
2   stage: test
3   tags:
4     - my-runner-tag
5   script:
6     - $INSTALL_DEPS
7     - npm run test
```

Listing 4.4: Test-Stage in der Client-CI/CD-Pipeline

Der `test`-Job wird in der `test`-Stage ausgeführt und verwendet das Skript `npm run test`, welches die Unit-Tests startet. Dabei werden sowohl die Funktionstests als auch die Barrierefreiheitstests ausgeführt. Durch die Einbindung der Barrierefreiheitstests, wie im Listing 4.3 beschrieben, werden potenzielle Barrierefreiheitsverstöße während jedes Pipeline-Durchlaufs automatisch überprüft.

Die Integration in den CI/CD-Prozess ermöglicht es, dass bei jedem Commit oder Merge Request die Tests automatisch ausgeführt werden. Dies wird im Falle des betrachteten Kundenportals durch die Parent-Pipeline gesteuert, welche die Client-Pipeline unter bestimmten Bedingungen auslöst. Die Konfiguration der Parent-Pipeline erfolgt in der übergeordneten `gitlab-ci.yml`-Datei (siehe Listing A.3).

Der relevante Abschnitt der Parent-Pipeline ist der `client_pipeline`-Job:

```
1 client_pipeline:
2   stage: client_pipeline
3   resource_group: "child_pipeline"
4   variables:
5     CHILD_PIPELINE_RUNNER_TAG: "my-runner-tag"
6     CHILD_PIPELINE_EXECUTION_CONTEXT: "client"
7     PARENT_TRIGGER: $CI_PIPELINE_SOURCE
8     DEPLOY_STRATEGY: $DEPLOY_STRATEGY
9   trigger:
10    include: .sub-gitlab-ci.yml
11    strategy: depend
12  rules:
13    - if: $CI_PIPELINE_SOURCE == "push"
14      changes: [client/**/*]
15      when: on_success
16    - if: $MERGE_REQUEST_ID
```

```
17        when: on_success
18      - if: $CI_PIPELINE_SOURCE == "parent_pipeline"
19        when: on_success
20      - if: $CI_PIPELINE_SOURCE == "web"
21        when: on_success
22      - if: $DEPLOY_STRATEGY == "PRODUCTION"
23        when: on_success
24      - if: $DEPLOY_STRATEGY == "STAGING"
25        when: on_success
```

Listing 4.5: Client-Pipeline in der Parent-CI/CD-Pipeline

Der `client_pipeline`-Job startet die Ausführung der Client-Pipeline als Child-Pipeline mittels des `trigger`-Keywords. Die `rules`-Sektion definiert die Bedingungen, unter denen die Pipeline ausgelöst wird, beispielsweise bei Push-Events auf der aktiven Feature-Branch (also während dem aktiven Entwickeln), Merge Requests oder bestimmten Deployment-Strategien. Dadurch wird sichergestellt, dass die Tests nur bei relevanten Änderungen ausgeführt werden.

Durch die kontinuierliche Integration der Barrierefreiheitstests in den CI/CD-Prozess werden Verstöße gegen die WCAG-Richtlinien frühzeitig erkannt. Entwickler erhalten sofortiges Feedback und können Probleme zeitnah beheben. Dies trägt wesentlich zur Qualitätssicherung bei und unterstützt die Einhaltung gesetzlicher Anforderungen, wie dem Barrierefreiheitsstärkungsgesetz.

Die Vorteile dieser Integration für den Entwicklungsprozess sind vielfältig:

- **Frühzeitige Fehlererkennung**: Barrierefreiheitsprobleme werden bereits während der Entwicklung entdeckt.
- **Kontinuierliche Überwachung**: Die automatisierten Tests gewährleisten eine stetige Kontrolle der Barrierefreiheit.
- **Effizienzsteigerung**: Automatisierung reduziert manuellen Testaufwand und beschleunigt den Entwicklungszyklus.
- **Qualitätssicherung**: Die Einhaltung von Barrierefreiheitsstandards wird fortlaufend sichergestellt.

4.3 Durchführung und Auswertung der Tests

Die Durchführung der Barrierefreiheitstests erfolgt sowohl lokal während der Entwicklung, als auch automatisiert innerhalb der CI/CD-Pipeline. Dieser Abschnitt erläutert den Ablauf der Tests und beschreibt wie die Ergebnisse ausgewertet und dokumentiert werden. Der Schwerpunkt liegt dabei auf den automatisierten Tests, während die

manuellen Tests gemäß den etablierten Richtlinien durchgeführt und ausgewertet werden sollten[1].

4.3.1 Durchführung der automatisierten Tests

Die automatisierten Barrierefreiheitstests werden mithilfe der in Abschnitt 4.2.2 beschriebenen `AccessibilityTester`-Klasse in die bestehenden Unit-Tests integriert. Die Tests können sowohl lokal durch die Entwickler, als auch automatisiert durch die CI/CD-Pipeline ausgeführt werden.

Lokale Ausführung der Tests

Entwickler können die Barrierefreiheitstests lokal ausführen, indem sie den folgenden Befehl `npm run test` im Projektverzeichnis ausführen. Dies ermöglicht eine sofortige Rückmeldung über potenzielle Barrierefreiheitsverstöße in der Entwicklungsphase. Die Testergebnisse werden in der Datei `accessibility-test-results.json` gespeichert, welche im Projektverzeichnis abgelegt wird. Durch die lokale Ausführung der Tests können Entwickler Probleme frühzeitig erkennen und beheben, bevor der Code in das Versionskontrollsystem überführt wird.

Ausführung der Tests in der CI/CD-Pipeline

Innerhalb der CI/CD-Pipeline werden die Tests automatisch in der `test`-Stage ausgeführt (siehe Abschnitt 4.2.4). Bei jedem Push oder Merge Request wird die Pipeline ausgelöst, sofern die in Listing 4.5 definierten Regeln zutreffen. Die Ausführung der Tests in der Pipeline stellt sicher, dass Barrierefreiheitsprobleme nicht unbemerkt in das Versionskontrollsystem, oder gar das ausgelieferte Produkt, gelangen.

4.3.2 Auswertung der Testergebnisse

Die `AccessibilityTester`-Klasse sammelt die Ergebnisse der Barrierefreiheitstests und speichert sie in der Datei `accessibility-test-results.json`. Dies erlaubt eine strukturierte Darstellung der Testergebnisse, einschließlich Informationen über die Art der Verstöße, betroffene Elemente und Empfehlungen zur Behebung. Um die

1 Dies kann mittels Checklisten umgesetzt werden. Anlaufstellen sind u.a. die WCAG `https://www.w3.org/WAI/WCAG22/quickref/` oder BITV `https://bitvtest.de/tests-und-beratung/selbst-testen/bitv-selbstbewertung`

Arbeits- und Funktionsweise zu demonstrieren wird exemplarisch die `ActionButton`-Komponente des Kundenportals untersucht. Dabei wird zunächst der Ausgangszustand dieser Komponente analysiert. Identifizierte Verstöße gegen die Barrierefreiheit werden erläutert und die vorzunehmenden Verbesserungen beschrieben.

Der Ausgangszustand der `ActionButton`-Komponente ist im Listing A.5 dargestellt. Die Komponente verwendet Elemente der Bibliothek `react-bootstrap`, um ein Dropdown-Menü bereitzustellen (siehe Abbildung 4.1). Sie besteht aus einem

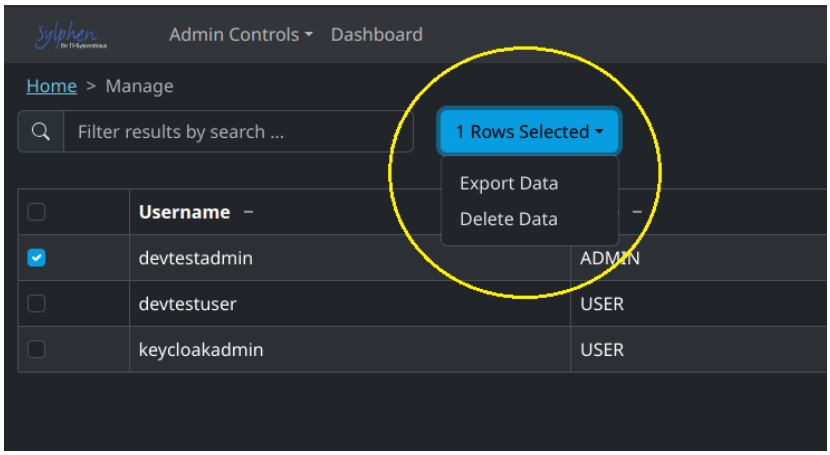

Abbildung 4.1: ActionButton-Komponente

`Dropdown.Toggle` zur Steuerung des ein- und ausklappbaren Menüs sowie einem `Dropdown.Menu`, in dem die einzelnen Menüeinträge angezeigt werden. Die Einträge werden als Objekte mit einem Namen und einer optionalen Aktionsfunktion übergeben. Bei Anklicken eines Eintrags wird die zugehörige Aktion ausgeführt. Diese Vorgehensweise ermöglicht eine flexible, wiederverwendbare und wartungsfreundliche Gestaltung des UI-Elements, ohne Code für jeden Menüeintrag erneut implementieren zu müssen.

Identifizierte Barrierefreiheitsverstöße

Die Testergebnisse werden in der `accessibility-test-results.json`-Datei gespeichert, welche im Anhang abgebildet ist (siehe Listing A.6). Die Einträge zeigen, dass die getestete `ActionButton`-Komponente zwei Verstöße aufweist und die Tests dadurch nicht erfolgreich abgeschlossen werden konnten. Das Resultat zeigt einen kritischen Verstoß gegen die Regel `button-name`, sowie die Verwendung eines ARIA-Attributs, welches fälschlicherweise ohne korrektes *role*-Attribut verwendet wird. Zusätzlich zu den gefunden Mängeln stellt das eingebundene Framework *jest-axe* weitere Informationen bereit. Dazu zählen die Beschreibung des Problems, Hinweise zur Behebung und eine Verlinkung zu weiterführenden Ressourcen. Wird der Test nicht über die Pipeline, sondern direkt über die Kommandozeile gestartet, werden die Ergebnisse zusätzlich in

der Konsole ausgegeben, wie in Abbildung 4.2) zu sehen ist:

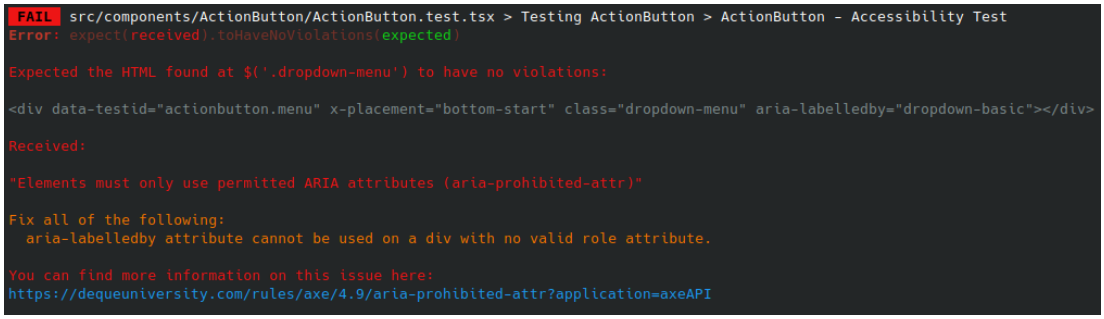

Abbildung 4.2: Testergebnis CLI

Bei der Durchführung der Unit-Tests wurden demnach folgende Verstöße festgestellt (siehe `accessibility-test-results.json` aus Listing A.6):

1. **`button-name`**
   - **Schweregrad**: *critical*
   - **Beschreibung**: Schaltflächen müssen einen erkennbaren Text haben.
   - **Betroffenes Element**: `<button id="dropdown-basic">`
   - **Problem**: Die Schaltfläche hat keinen für Screenreader sichtbaren Text. Weder ein innerer Text noch ein `title`-Attribut sind vorhanden. Dadurch ist die Funktion der Schaltfläche für Screenreadern nicht erkennbar.
2. **`aria-prohibited-attr`**
   - **Schweregrad**: *serious*
   - **Beschreibung**: Elemente dürfen nur erlaubte ARIA-Attribute verwenden.
   - **Betroffenes Element**: `<div class="dropdown-menu">`
   - **Problem**: Es existiert ein `aria-labelledby`-Attribut für welches kein gültiges `role`-Attribut definiert ist. Assistive Technologien können das Attribut daher nicht korrekt interpretieren.

4.3.3 Behebung der gefundenen Verstöße

Um die identifizierten Verstöße zu beheben, werden gezielte Anpassungen an der `ActionButton`-Komponente vorgenommen. Die Schaltfläche `Dropdown.Toggle` er-

hält ein `aria-label`-Attribut, das entweder den Wert von `props.label` oder den Standardwert „Action Button“ enthält (siehe Zeile 26, Listing A.7). Dadurch verfügt die Schaltfläche über einen erkennbaren Namen für Screenreader, selbst wenn kein sichtbarer Text vorhanden ist.

Des Weiteren wird dem `Dropdown.Menu` das Attribut `role="menu"` hinzugefügt (siehe Zeile 35, Listing A.7). Dieses Attribut definiert das Element für assistive Technologien korrekt als Menü und ermöglicht die korrekte Interpretation des vorhandenen `aria-labelledby`-Attributs.

Zusätzlich erhält jedes `Dropdown.Item`-Elementen ein `role="menuitem"`-Attribut, um die Menüeinträge eindeutig zu kennzeichnen und die Navigation innerhalb des Menüs für Nutzer von Screenreadern zu verbessern (siehe Zeile 43, Listing A.7).

Durch diese Anpassungen werden die zuvor identifizierten Barrierefreiheitsverstöße behoben. Die Schaltfläche ist nun für Screenreader erkennbar und das Menü, sowie die Menüeinträge sind korrekt ausgezeichnet.

Erneute Ausführung der Tests

Nach den Anpassungen wird die `ActionButton`-Komponente erneut getestet. Die aktualisierten Testergebnisse zeigen, dass die Komponente nun den Barrierefreiheitstest erfolgreich besteht. Die Dokumentations-Datei `accessibility-test-results.json` wird aktualisiert (siehe Listing A.8). Das Attribut `"success": true` bestätigt, dass keine weiteren Verstöße identifiziert wurden (siehe Screenshot 4.3).

Abbildung 4.3: accessibility-results.json, erfolgreich

4.3.4 Manuelles Testen

Kontrastprüfung mit WAVE

Zur Überprüfung der Farbkontraste wird die zuvor vorgestellte Browsererweiterung WAVE eingesetzt (siehe Kapitel 3.4.2). Dazu wird die zu analysierende Webseite in einem geeigneten Webbrowser geöffnet. Anschließend lässt sich die WAVE-Erweiterung

über das entsprechende Browser-Menü starten. Das Werkzeug analysiert Kontraste sowie generelle Barrieren der Webseite und hebt diese visuell hervor. Nachfolgend wird die Kontrastprüfung auf der Startseite des Kundenportals sowohl im dunklen, als auch im hellen Darstellungsmodus dokumentiert. Die Ergebnisse der Prüfung sind in den Abbildungen 4.4 und 4.5 dargestellt. Die markierten Bereiche zeigen, dass keine Fehler hinsichtlich der Kontrastwerte festgestellt werden.

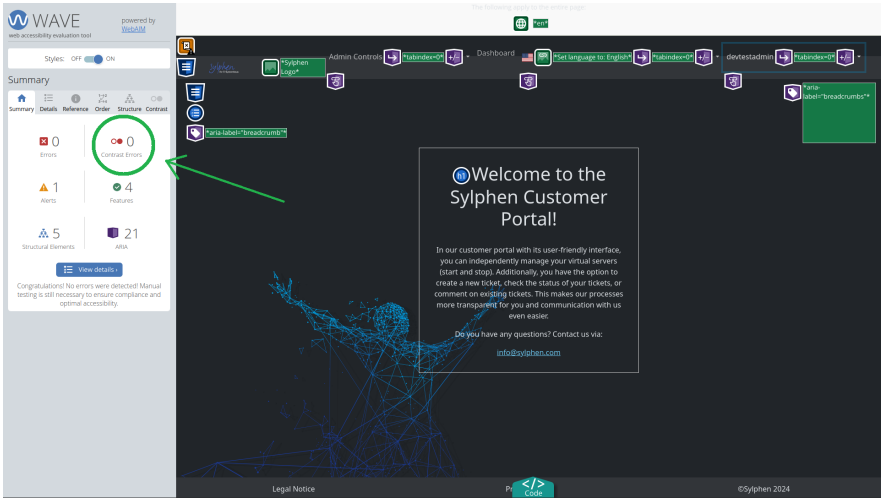

Abbildung 4.4: Kontrastprüfung der Startseite im dunklen Darstellungsmodus mittels WAVE.

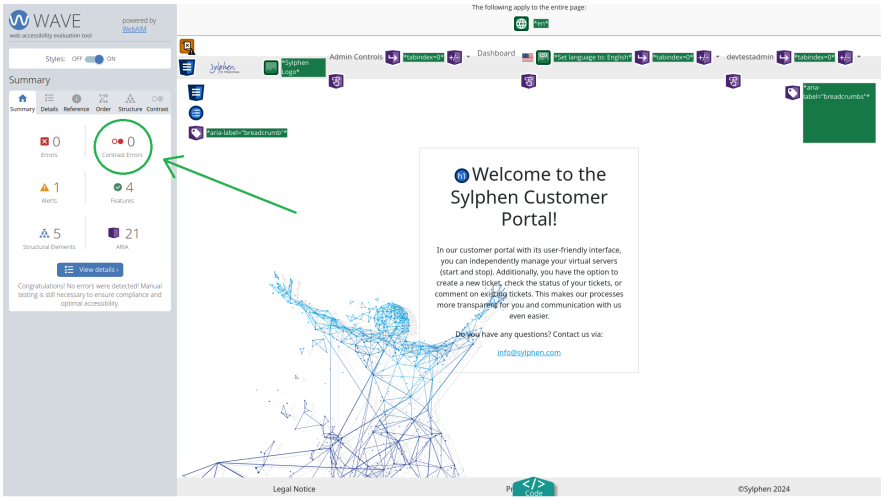

Abbildung 4.5: Kontrastprüfung der Startseite im hellen Darstellungsmodus mittels WAVE.

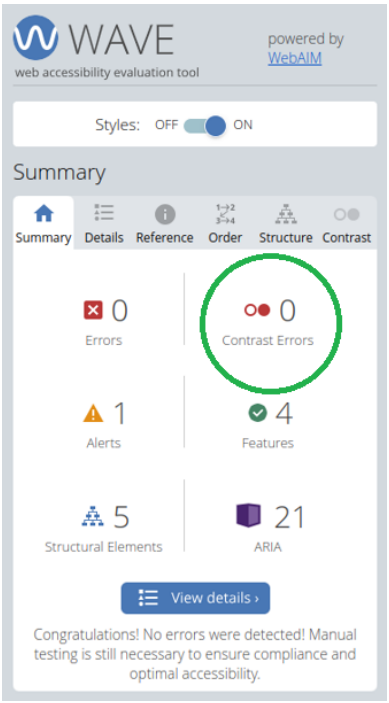

Abbildung 4.6: Testergebnis von WAVE.

Tastaturnavigation

Die Web Content Accessibility Guidelines (WCAG) fordern, dass sämtliche interaktive Elemente einer Webseite allein mit der Tastatur bedienbar sind (vgl. [Egg24b]). Dies umfasst unter anderem die Möglichkeit, Navigationsleisten, Links, Schaltflächen sowie Dropdown-Menüs mittels Tastatur ansteuern und auslösen zu können. Diese Anforderung gewährleistet, dass Nutzer, welche auf alternative Eingabegeräte angewiesen sind oder keine Maus verwenden können, ohne Einschränkungen auf alle Inhalte und Funktionen zugreifen können (vgl. [Cam24]). Im vorliegenden Fall dient die Navigationsleiste des Kundenportals als Beispiel für die Prüfung der Tastaturnavigation.

Über die Tabulatortaste werden alle fokussierbaren Elemente der Navigationsleiste nacheinander angesteuert. Der Fokus soll dabei visuell erkennbar sein, etwa durch eine Hervorhebung oder einen farblich abgesetzten Rahmen (siehe Abbildung 4.7). Die erste Betätigung der Tabulatortaste setzt den Fokus auf das erste Element der Navigationsleiste (wie in Abbildung 4.7 zu sehen ist). Durch jede weitere Betätigung wird der Fokus auf das nächste Element verschoben.

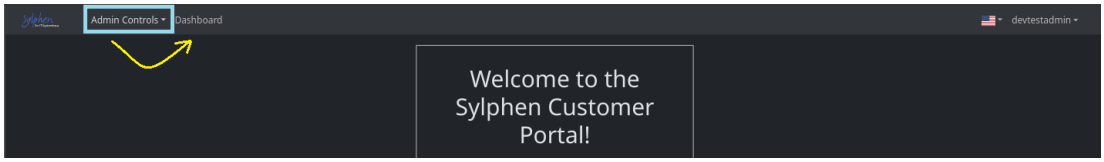

Abbildung 4.7: Fokuszustand initial auf dem Start der Navigationsleiste.

Im gezeigten Beispiel wird bis zum Dropdown-Menü am rechten Rand der Navigationsleiste navigiert (siehe Abbildung 4.8).

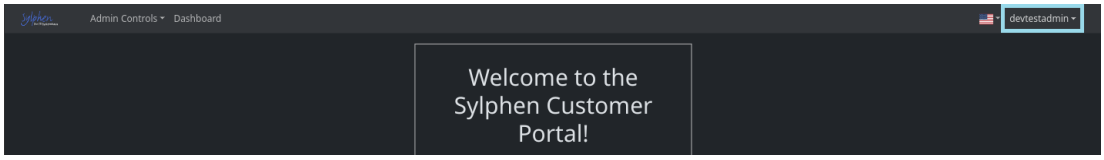

Abbildung 4.8: Fokuszustand auf dem Dropdown-Menü der Navigationsleiste.

Untergeordnete Menüeinträge sollen sich ebenfalls per Tastatur erreichen und auswählen lassen. Das gezeigte Dropdown-Menü wird mittels Enter oder Leertaste angesteuert (siehe Abbildung 4.10).

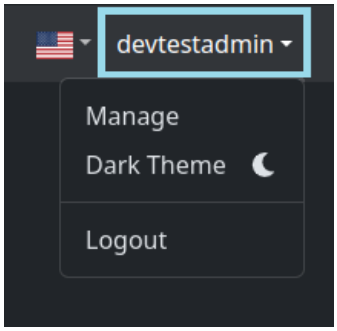

Abbildung 4.9: Fokuszustand auf der Navigationsleiste mittels Tastaturnavigation.

Nutzer navigieren mithilfe der Tastatur durch das Dropdown-Menü, indem sie die Tabulatortaste oder die Pfeiltasten (hoch, runter) verwenden. Wie in Abbildung 4.10 gezeigt, ist auch diese Funktionalität gewährleistet.

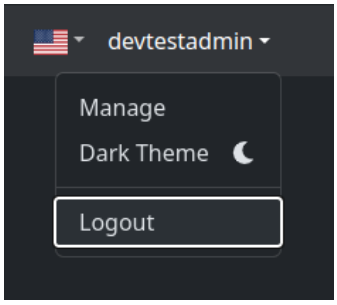

Abbildung 4.10: Fokus auf den Logout-Menüpunkt im Dropdown-Menü.

Um eine Aktion auszuführen betätigen Nutzer die Eingabetaste oder die Leertaste (abhängig von Browser und Implementierung). Das Dropdown-Menü wird durch Drücken der Escape-Taste geschlossen, wodurch die Menüelemente ausgeblendet werden. Nach dem Schließen des Menüs sollte der Fokus korrekt auf das ursprüngliche Steuerelement

(z. B. die Schaltfläche oder den Link, der das Dropdown geöffnet hat) zurückgesetzt werden (vgl. [Puz15]), um die Tastaturnavigation konsistent zu halten und die Benutzerfreundlichkeit sicherzustellen (siehe Abbildung 4.11).

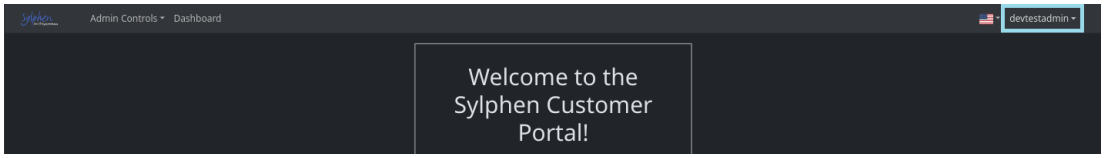

Abbildung 4.11: Fokuszustand zurück auf dem Dropdown-Menü, nach Schließung.

Die beschriebene Tastaturnavigation stellt lediglich einen Ausschnitt der Anforderungen an die Zugänglichkeit einer Webanwendung dar. Sämtliche Unterseiten sowie interaktiven Funktionen einer Anwendung müssen systematisch getestet werden, um sicherzustellen, dass die Navigation, Bedienbarkeit und der Fokusverlauf konsistent und barrierefrei umgesetzt sind (vgl. [Cam24]). Nur durch eine vollständige Prüfung kann garantiert werden, dass alle Nutzer – unabhängig von ihren Fähigkeiten oder Einschränkungen – die Anwendung effektiv nutzen können.

Screenreader

Zusätzlich zur Tastaturnavigation, die bereits eine gute Grundlage für die Verwendung von Screenreadern bietet (vgl. [Gra18]), ist eine konkrete Überprüfung der Barrierefreiheit mithilfe von Screenreader-Technologien unerlässlich. Entwickler können diese Tests durchführen, indem sie die Anwendungen schrittweise durchgehen und die vorgelesenen Inhalte und Interaktionen analysieren. Dabei sollte besonders auf die logische Reihenfolge des Fokus, verständliche Beschreibungen von Bedienelementen sowie auf alternative Texte für visuelle Inhalte geachtet werden. Eine klare semantische Struktur des Codes, beispielsweise durch die Verwendung passender HTML-Elemente und ARIA-Rollen (siehe Abschnitt 2.3.1 und 2.3.2), ist entscheidend für die korrekte Interpretation durch unterstützende Technologien und ermöglicht Nutzern eine nahtlose und barrierefreie Navigation. Die WCAG-Richtlinien enthalten keine konkreten Vorgaben für die Verwendung von Screenreadern. Als Einführung in diese Thematik kann der folgende Artikel vom Bundesministerium des Innern und für Heimat herangezogen werden[1].

1 Bundesministerium des Innern und für Heimat (2024): Screenreader-Test mit NVDA, verfügbar unter: [https://www.barrierefreiheit-dienstekonsolidierung.bund.de/Webs/PB/DE/barrierefreie_it/pruefen/produktorientierter-test/screenreader/screenreader-test-node.html] (Zugriff am 17. Dezember 2024).

5 Abschluss

Dieses Kapitel dient dazu, die behandelten Themen und erzielten Ergebnisse zusammenzuführen und einzuordnen. Zunächst wird eine Zusammenfassung über die behandelten Inhalte gegeben. Im Fazit werden die behandelten Forschungsfragen aufgegriffen und explizit beantwortet. Anschließend werden die erarbeiteten Ergebnisse in einer Diskussion kritisch reflektiert. Abschließend werden weitere Ansätze und mögliche zukünftige Forschungs- und Entwicklungsrichtungen aufgezeigt, um die Barrierefreiheit in der Praxis weiter zu verbessern.

5.1 Zusammenfassung

Diese Bachelorarbeit widmet sich der Untersuchung der Herausforderungen und Lösungsansätze bei der Implementierung und Überprüfung von Barrierefreiheit in modernen Webanwendungen. Ziel war es, ein Verständnis für die technischen und organisatorischen Herausforderungen zu entwickeln und ein Testkonzept zu erarbeiten, das sowohl den Anforderungen der Web Content Accessibility Guidelines (WCAG) als auch dem Barrierefreiheitsstärkungsgesetz (BFSG) gerecht wird.

Zu Beginn dieser Arbeit wurden die theoretischen Grundlagen und die Prinzipien der Barrierefreiheit erläutert. Relevante rechtliche und normative Rahmenbedingungen, wie das Barrierefreiheitsstärkungsgesetz (BFSG) und die Anforderungen der Web Content Accessibility Guidelines (WCAG) wurden beschrieben. Die notwendigen technischen Hintergründe und grundlegenden Technologien, wie semantisches HTML und WAI-ARIA, wurden als Einstiegspunkt in die Bearbeitung der Forschungsfragen gelegt.

Anschließend wurden die spezifischen Herausforderungen der Barrierefreiheit analysiert. Neben der technischen Komplexität wurden auch organisatorische Schwierigkeiten, wie begrenzte Budgets und mangelnde Sensibilisierung im Unternehmen, thematisiert.

Darauf aufbauend wurde ein Testkonzept entwickelt, das automatisierte und manuelle Prüfmethoden kombiniert. Eine Auswahl an Test-Frameworks und -Tools wurde untersucht, um die besten Ansätze zur Überprüfung der Barrierefreiheit zu identifizieren. Aufbauend auf dem ausgewählten Framework wurde eine Klasse entwickelt, die in

bestehende Unit-Tests integriert werden kann. Die Integration dieses Konzepts wurde exemplarisch am Kundenportal der Firma Sylphen GmbH & Co. KG demonstriert, um die praktische Anwendbarkeit und Effektivität der entwickelten Lösung aufzuzeigen.

5.2 Fazit

Die im Laufe dieser Arbeit erlangten Erkenntnisse erlauben die Beantwortung der beiden zu Beginn formulierten Forschungsfragen wie folgt:

1. **Welche Herausforderungen existieren bei der Implementierung von Barrierefreiheit in Webanwendungen?**
   Es existieren sowohl technische, als auch organisatorische Herausforderungen bei der Implementierung von Barrierefreiheit. Aus technischer Perspektive sind insbesondere komplexe Interaktionen, dynamische Inhalte und die korrekte semantische Auszeichnung entscheidende Faktoren, die eine nahtlose Umsetzung von Barrierefreiheit erschweren. Auf organisatorischer Ebene wurden zeitliche und finanzielle Ressourcenbeschränkungen, fehlendes Fachwissen und geringe Sensibilisierung innerhalb von Unternehmen als wesentliche Hindernisse erkannt. Insbesondere die hohen Kosten für nachträgliche Anpassungen und Barrierefreiheitszertifikate stellen Herausforderungen dar, die frühzeitig im Entwicklungsprozess adressiert werden müssen. Auch die Kosten für externe Prüfungen und Zertifizierungen erweisen sich als hoch, was den Prozess weiter erschwert. Zudem mangelt es an umfangreichen, zertifizierten Schulungsangeboten, die gezielt auf die Bedürfnisse von Webentwicklern zugeschnitten sind. Zwar existieren vereinzelt Angebote, jedoch bieten diese zumeist nur ein grundlegendes Verständnis und kein tiefgreifendes Fachwissen. Für komplexe Web-Projekte werden häufig höhere Prüfkosten (5.000–10.000 €, in Einzelfällen sogar 8.000–20.000 €) genannt, welche für kleinere Unternehmen eine erhebliche Belastung darstellen können. Diese Erkenntnisse verdeutlichen, dass Barrierefreiheit eine multidimensionale Aufgabe ist, die sowohl technisches Know-how als auch organisatorische Anpassungen erfordert. Die mangelnde Verfügbarkeit von umfangreichen Schulungsangeboten und die damit einhergehende Schwierigkeit, internes Expertenwissen aufzubauen, erschwert die Umsetzung barrierefreier Webanwendungen zusätzlich.

2. **Wie sieht ein Test-Konzept für die Barrierefreiheit von Webanwendungen aus und wie kann es in den Entwicklungsprozess integriert werden?**
   Eine Analyse von Testframeworks und -Tools führte zur Auswahl von *jest-axe* als zentrales Werkzeug, das durch eine eigens entwickelte *AccessibilityTester*-Klasse erweitert wurde. Diese Klasse ermöglicht eine einfache Integration in bestehende Unit-Tests und stellt sicher, dass Barrierefreiheitsaspekte bereits während der

Entwicklung kontinuierlich geprüft werden können. Das Konzept wurde exemplarisch im Kundenportal der Firma Sylphen GmbH & Co. KG implementiert und demonstriert. Die Integration in die CI/CD-Pipeline des Projekts ermöglicht eine kontinuierliche Überprüfung der Barrierefreiheit. Manuelle Tests, etwa die Überprüfung der Tastaturnavigation, Kontrastprüfung und Screenreader-Kompatibilität, ergänzen die automatisierten Verfahren. Dies stellt sicher, dass Barrierefreiheitsverstöße frühzeitig identifiziert und behoben werden. Kosten und Aufwände für nachträgliche Anpassungen werden dadurch minimiert.

5.3 Diskussion

Die Ergebnisse dieser Arbeit verdeutlichen, dass Barrierefreiheit ein vielschichtiges und anspruchsvolles Thema ist, welches technisches Know-how, organisatorische Strukturen und Prozesse sowie rechtliche Rahmenbedingungen erfordert. Im folgenden werden die Resultate dieser Arbeit kritisch hinterfragt.

In Bezug auf die erste Forschungsfrage ist festzustellen, dass die Datenbasis zu den Herausforderungen bei der Implementierung von Barrierefreiheit insgesamt begrenzt ist. Zudem bezieht sich die zitierte Studie von *Yavuz Inal et al.* explizit auf nordische Länder (vgl. [Ina20]). Obwohl diese Studie wertvolle Einblicke liefert, ist ihre Aussagekraft mit Einschränkungen zu betrachten, da kulturelle, rechtliche und strukturelle Unterschiede zwischen Ländern und Regionen die Aussagekraft beeinflussen können. Zudem unterliegen die Angaben zu Kosten und Schulungsangeboten deutlichen Variationen. Während einige Quellen relativ niedrige Kosten für einfache Prüfungen nennen (teils unter 1.000 Euro), können umfassende Audits oder Zertifizierungen bei komplexen Webanwendungen in Einzelfällen sogar bis zu 20.000 Euro betragen. Ebenso ist die Schulungssituation nicht einheitlich. Zwar existieren einzelne Kurse zu WCAG oder EN 301 549, doch bleiben umfangreiche, anerkannte und auf Webentwickler ausgerichtete Zertifizierungsangebote rar (siehe Abschnitt 3.2). Insgesamt zeigen diese Erkenntnisse, dass die identifizierten Herausforderungen schwer in allgemeingültige Zahlen oder Maßnahmen zu überführen sind und machen deutlich, dass weitere Forschung und eine breitere Datengrundlage erforderlich sind, um differenzierte Handlungsempfehlungen ableiten zu können.

Die im Rahmen der zweiten Forschungsfrage entwickelte *AccessibilityTester*-Klasse ermöglicht eine einfache Integration in vorhandene Unit-Tests und damit den Zugang zur Prüfung von Barrierefreiheitsaspekten. Die exemplarische Implementierung im Kundenportal der Firma Sylphen demonstriert die praktische Anwendbarkeit des Konzepts und verdeutlicht, wie automatisierte Tests nahtlos in bestehende CI/CD-Pipelines integriert werden können.

Nachfolgend wird anhand der definierten Anforderungen und Akzeptanzkriterien (siehe Abschnitt 3.3) beurteilt, inwieweit das entwickelte Test-Konzept diese erfüllt.

A1: Kontinuierliche Prüfung der Barrierefreiheit im Entwicklungsprozess

- **A1.1:** Erfüllt. Durch die Integration der Barrierefreiheits-Tests in die CI/CD-Pipeline werden diese bei jedem neuen Build automatisiert ausgeführt.
- **A1.2:** Erfüllt. Entwickler erhalten nach der Testausführung ausführliche Berichte über potenzielle Verstöße, wodurch unmittelbares Feedback zur Verbesserung möglich ist.
- **A1.3:** Erfüllt. Bei festgestellten Verstößen schlägt der Test fehl, um sicherzustellen, dass Probleme vor dem Deployment behoben werden.

A2: Nahtlose Integration in bestehende Entwicklungsumgebungen

- **A2.1:** Erfüllt. Das Testkonzept (insbesondere *jest-axe*) ist kompatibel mit gängigen Frontend-Frameworks wie React, Angular oder Vue.
- **A2.2:** Erfüllt. Da *jest-axe* auf Jest basiert.
- **A2.3:** Erfüllt. Die Integration erfordert nur minimale Konfigurations- und Code-Anpassungen, wodurch zusätzliche Aufwände geringgehalten werden.

A3: Zentrale Dokumentation der Testergebnisse

- **A3.1:** Erfüllt. Die Ergebnisse werden in einer zentralen JSON-Datei abgelegt, welche eine strukturiert Auswertung erlaubt.
- **A3.2:** Erfüllt. Für jede getestete Komponente werden detaillierte Informationen zu Verstößen und Empfehlungen zur Behebung bereitgestellt.

B1: Kosten- und Aufwandsreduktion in Implementierung und Wartung

- **B1.1:** Erfüllt. Durch den Einsatz von Open-Source-Testtools wie *jest-axe* entstehen keine zusätzlichen Lizenzkosten.
- **B1.2:** Erfüllt. Die Konfiguration und Wartung erfordern kein spezielles Expertenwissen, da auf gängige Werkzeuge und standardisierte Abläufe zurückgegriffen wird.

Trotzdem muss angemerkt werden, dass die Verwendung dieses Konzepts nicht garantiert, dass sämtliche Anforderungen an das Barrierefreiheitsstärkungsgesetzes, oder den Web Content Accessibility Guidelines erfüllt werden. Barrierefreiheit ist kontextabhängig und jede Anwendung weist individuelle Anforderungen auf, die spezifisch geprüft und

bearbeitet werden müssen. Automatisierte Tests können viele Barrieren identifizieren, erfassen jedoch nicht die gesamte Bandbreite der Barrierefreiheit. Ergänzend müssen manuelle Tests - im besten Fall durch betroffene Personen - durchgeführt werden. Nur so kann sichergestellt werden, dass alle relevanten Anforderungen erfüllt sind.

5.4 Weitere Ansätze

Neben der dargestellten Kombination aus automatisierten und manuellen Prüfungen existieren weitere, bisher nur angeschnittene Möglichkeiten, um die Barrierefreiheit in Webanwendungen sicherzustellen.

Nutzerzentrierte Evaluierung

Das Einbeziehen von Personen mit Behinderungen, in den Testprozess, ermöglicht es praxisnahe Einblicke zu gewinnen und spezifische Barrieren zu identifizieren. Dies ermöglicht es Hindernisse zu entdecken, welche weder durch automatisierte Tests, noch durch manuelle Testen von nicht beeinträchtigten Personen, entdeckt werden können. Diese Vorgehensweise stellt sicher, dass die entwickelten Produkte den tatsächlichen Bedürfnissen aller Nutzer entsprechen.

Verwendung barrierefreier Komponentenbibliotheken

Der Einsatz von vorgefertigten, barrierefreien UI-Komponenten, kann die Entwicklung erleichtern und die Einhaltung von Barrierefreiheitsstandards fördern. Beispielsweise bietet *KoliBri* (Komponentenbibliothek für die Barrierefreiheit) standardisierte, barrierefreie Bausteine für Webanwendungen an, welche die Barrierefreiheit gemäß BGG, BITV 2.0, EN 301 549 und WCAG sicherstellen. Die Komponentenbibliothek wird vom *Informationstechnikzentrum Bund* als Open Source entwickelt und zur Wiederverwendung freigegeben (vgl. [Inf24], [Bune]).

Accessibility-Overlays

Eine weitere Möglichkeit um eingeschränkten Nutzern ein positives Nutzungserlebnis zu bieten sind *Accessibility-Overlays*. Dabei handelt es sich um ein zusätzliches Menü, mit dem die Webseite auf die persönlichen Bedürfnisse des Nutzers angepasst werden kann. Sie bieten die Möglichkeit den Kontrast anzupassen, Schriftgrößen zu verändern oder die Tastaturnavigation zu verbessern - so zumindest der Grundgedanke. In dem

Podcast „Wo wir sind ist vorne" spricht *Daniela Kubesch* als Gast über Ihre Forschungen zum Thema Accessibility-Overlays und erklärt, dass diese die Barrierefreiheit oft eher einschränken, als verbesseern (vgl. [Wo 24]).

Schulungen und Sensibilisierungsmaßnahmen

Die kontinuierliche Weiterbildung von Entwicklern und Designern im Bereich Barrierefreiheit ist entscheidend. Dies fördert das Bewusstsein für Barrieren und befähigt dazu, barrierefreie Lösungen von Anfang an zu konzipieren. Das Bundesministerium des Innern und für Heimat betont die Bedeutung solcher Maßnahmen für die Umsetzung barrierefreier digitaler Angebote und bietet einen Einstieg in diese Thematik (vgl. [Bunc]). Die von den Autoren *Yavuz Inal et al.* identifizierten Hürden (siehe Abbildung 3.1 aus Kapitel 3.2), wie mangelnde Schulungen und fehlende Sensibilisierung, unterstreichen die Notwendigkeit dieser Maßnahmen. Ohne ein fundiertes Verständnis für Barrierefreiheit können selbst gut gemeinte technische Lösungen ineffektiv umgesetzt sein.

Literaturverzeichnis

[Aba19] Abascal, Julio; Arrue, Myriam und Valencia, Xabier: Tools for Web Accessibility Evaluation, in: Yeliz Yesilada und Simon Harper (Herausgeber) *Web Accessibility: A Foundation for Research*, Springer, London, 2 Aufl. (2019), S. 479–503

[Ado] Adobe: Single-page applications (SPAs) — what they are and how they work, URL https://business.adobe.com/blog/basics/learn-the-benefits-of-single-page-apps-spa, Zugriff am: 05. August 2024

[Akt] Aktion Mensch e.V.: Barrierefreie Website: Kosten, URL https://www.aktion-mensch.de/inklusion/barrierefreiheit/barrierefreie-website/kosten-barrierefreie-website, Zugriff am: 17. Dezember 2024

[Akt21] Aktion Mensch e.V.: Barrieren im Alltag – wer sie wahrnimmt und wen sie behindern. Ergebnisse einer bundesweiten Umfrage (2021), S. 5

[Als20] Alsaeedi, Abdullah: Comparing Web Accessibility Evaluation Tools and Evaluating the Accessibility of Webpages: Proposed Frameworks. *Information* (2020), Bd. 11(1), URL https://www.mdpi.com/2078-2489/11/1/40

[Ant24] Anthony, Fu; Matías, Capeletto und contributors, Vitest: Vitest Next Generation Testing Framework: A Vite-native testing framework. It's fast! (2024), URL https://vitest.dev/, Zugriff am: 19. November 2024

[Bar] Barrierefreiheit Umsetzen Malekzadeh & Franzke GbR: BIK BITV-Test: Kosten, Preise, Dauer, Siegel, URL https://barrierefreiheit-umsetzen.de/bik-bitv-test/, Zugriff am: 17. Dezember 2024

[BD24] Bundesrepublik Deutschland, vertreten durch den Bundesminister der Justiz: Gesetz zur Gleichstellung von Menschen mit Behinderungen (Behindertengleichstellungsgesetz - BGG), §4 Barrierefreiheit, https://www.gesetze-im-internet.de/bgg/__4.html (2024), Redaktion: Bundesamt für Justiz, Kompetenzzentrum Rechtsinformationssystem des Bundes, Zugriff am: 20. August 2024

[Bei] Beisch, Natalie und Koch, Wolfgang: Aktuelle Aspekte der Internetnutzung in Deutschland ARD/ZDF-Onlinestudie: Weitergehende Normalisierung

der Internetnutzung nach Wegfall aller Corona-Schutzmaßnahmen, URL https://www.ard-zdf-onlinestudie.de/files/2023/MP_23_2023_Onlinestudie_2023_Fortschreibung.pdf, Zugriff am: 30. Juli 2024

[BG] BFSG-Gesetz.de: BFSG (Barrierefreiheitsstärkungsgesetz), URL https://bfsg-gesetz.de/, Zugriff am: 01. August 2024

[Buna] Bundesministerium des Innern und für Heimat: Barrierefreiheitsstärkungsgesetz (BFSG), URL https://www.barrierefreiheit-dienstekonsolidierung.bund.de/Webs/PB/DE/gesetze-und-richtlinien/barrierefreiheitsstaerkungsgesetz/barrierefreiheitsstaerkungsgesetz-node.html;jsessionid=27AA568F3B292D80761AB4A0EA5A91E3.live882, Zugriff am: 30. Juli 2024

[Bunb] Bundesministerium des Innern und für Heimat: Digitale Barrierefreiheit, URL https://www.barrierefreiheit-dienstekonsolidierung.bund.de/Webs/PB/DE/barrierefreie_it/digitale-barrierefreiheit/digitale-barrierefreiheit-node.html, Zugriff am: 30. Juli 2024

[Bunc] Bundesministerium des Innern und für Heimat: Digitale Barrierefreiheit. In einer zunehmend digitalen Welt ist Barrierefreiheit der Schlüssel. Er öffnet uns die Türen zu einer Welt, in der Menschen mit unterschiedlichen Fähigkeiten gleichberechtigt teilhaben können. Doch, was bedeutet eigentlich digitale Barrierefreiheit?, URL https://www.barrierefreiheit-dienstekonsolidierung.bund.de/Webs/PB/DE/barrierefreie_it/digitale-barrierefreiheit/digitale-barrierefreiheit-node.html, Zugriff am: 10. Dezember 2024

[Bund] Bundesministerium des Innern und für Heimat: Harmonisierte Europäische Norm (EN) 301 549, https://www.barrierefreiheit-dienstekonsolidierung.bund.de/Webs/PB/DE/gesetze-und-richtlinien/en301549/en301549-node.html, Zugriff am: 22. August 2024

[Bune] Bundesministerium des Innern und für Heimat: KoliBri - Die Komponenten-Bibliothek für die Barrierefreiheit, URL https://www.barrierefreiheit-dienstekonsolidierung.bund.de/SharedDocs/kurzmeldungen/Webs/PB/DE/fokusthemen/kolibri_fokus.html?utm_source=chatgpt.com, Zugriff am: 10. Dezember 2024

[Bun24] Bundesrepublik Deutschland, vertreten durch den Bundesminister der Justiz: Verordnung zur Schaffung barrierefreier Informationstechnik nach dem Behindertengleichstellungsgesetz (Barrierefreie-Informationstechnik-Verordnung - BITV 2.0), https://www.gesetze-im-internet.de/bitv_2_0/BJNR184300011.html (2024), Redaktion: Bundesamt für Justiz, Kompetenzzentrum Rechtsinformationssystem des Bundes, Zugriff am: 22. August 2024

[Cam24] Campbell, Alastair; Adams, Charles; Montgomery, Rachael Bradley; White, Kevin; Abou-Zahra, Shadi; Lee, Steve und Henry, Shawn Lawton: WCAG 2.2 Techniques: General Technique G202 (2024), URL https://www.w3.org/WAI/WCAG22/Techniques/general/G202.html, Zugriff am: 17. Dezember 2024

[Col24] Colley, Nick: jest-axe Github Repository (2024), URL https://github.com/nickcolley/jest-axe, Zugriff am: 17. November 2024

[Cro24] Croll, Jutta: Barrierefreiheit – im Internet – im Wandel der Zeiten. Ein Abriss der Entwicklungen seit der Jahrtausendwende, in: Vanessa Heitplatz und Leevke Wilkens (Herausgeber) *Die Rehabilitationstechnologie im Wandel: Eine Mensch-Technik-Umwelt Betrachtung*, Eldorado, Dortmund (2024), S. 190–199

[Deq] Deque Systems: axe Accessibility Linter, https://marketplace.visualstudio.com/items?itemName=deque-systems.vscode-axe-linter, Zugriff am: 28. Oktober 2024

[Deq24a] Deque Labs: axe-core: Accessibility testing engine for websites and HTML-based user interfaces (2024), URL https://github.com/dequelabs/axe-core, Zugriff am: 17. November 2024

[Deq24b] Deque Systems: The ultimate digital accessibility testing toolkit (2024), URL https://www.deque.com/axe/devtools/, Zugriff am: 17. November 2024

[deu] deutschland.de: Deutschland in Zahlen und Fakten, URL https://www.deutschland.de/de/topic/politik/deutschland-und-europa/deutschland-in-zahlen-und-fakten, Zugriff am: 30. Juli 2024

[DIA] DIAS GmbH - Daten, Informationssysteme und Analysen im Sozialen: BIK BITV-Test für barrierefreie Webseiten, URL https://bitvtest.de/tests-und-beratung/bik-bitv-test-web

[DIA24] DIAS GmbH - Daten, Informationssysteme und Analysen im Sozialen: Beschreibung des Prüfverfahrens (Web) (2024), URL https://bitvtest.de/test-methodik/web/beschreibung-des-pruefverfahrens, Zugriff am: 17. November 2024

[Dow19] Dowden, Martine und Dowden, Michael: *Tools, Technologies, and Resources*, Apress, Berkeley, CA (2019), S. 95–116, URL https://doi.org/10.1007/978-1-4842-4881-2_6

[Egg24a] Eggert, Eric; Abou-Zahra, Shadi; Vanderheiden, Gregg; Reid, Loretta Guarino; Caldwell, Ben; Henry, Shawn Lawton und Lemon, Gez: WCAG 2.2 Quick Reference (2024), URL https://www.w3.org/WAI/WCAG22/quickref/?showtechniques=pageinfo, Zugriff am: 29. Oktober 2024

[Egg24b] Eggert, Eric; Abou-Zahra, Shadi; Vanderheiden, Gregg; Reid, Loretta Guarino; Caldwell, Ben; Henry, Shawn Lawton und Lemon, Gez: Web Content Accessibility Guidelines (WCAG) Quick Reference (2024), URL https://www.w3.org/WAI/WCAG22/quickref/#keyboard-accessible, Zugriff am: 17. Dezember 2024

[eko] ekom21 – KGRZ Hessen: Barrierefreie IT - Barrieren beseitigen, URL https://www.ekom21.de/seminare/seminar/490/, Zugriff am: 17. Dezember 2024

[Eur16] Europäisches Parlament und der Rat der Europäischen Union: Richtlinie (EU) 2016/2102 des Europäischen Parlaments und des Rates über den barrierefreien Zugang zu den Websites und mobilen Anwendungen öffentlicher Stellen (2016), URL https://eur-lex.europa.eu/legal-content/DE/TXT/?uri=CELEX%3A32016L2102, Zugriff am: 22. August 2024

[Eve] Everett, Ruth: How & Why Accessibility Matters for SEO, URL https://www.searchenginejournal.com/seo-accessibility/379582/, Zugriff am: 31. Juli 2024

[Fog14] Fogli, Daniela; Provenza, Loredana Parasiliti und Bernareggi, Cristian: A universal design resource for rich Internet applications based on design patterns. *Universal Access in the Information Society* (2014), Bd. 13: S. 205–226, URL https://doi.org/10.1007/s10209-013-0291-6

[Gra18] Graap, Fabian; Haase, Sylvia; Jäckel, Stefanie; Steil, Hannah; Voigt, Henrik; Zehendner, Eberhard und Zimmer, Florian: Freie Werkzeuge zur Beurteilung des Barrierefreiheitsgrades von Webseiten, in: Raimund Dachselt und Gerhard Weber (Herausgeber) *Mensch und Computer 2018 - Workshopband*, Gesellschaft für Informatik e.V., Bonn, S. 31–42

[IBM24a] IBM: accessibility-checker (2024), URL https://www.npmjs.com/package/accessibility-checker, Zugriff am: 17. November 2024

[IBM24b] IBM: README IBM accessibility-checker: accessibility-checker NODE (2024), URL https://github.com/IBMa/equal-access/blob/master/accessibility-checker/README.md, Zugriff am: 17. November 2024

[Ina20] Inal, Yavuz; Guribye, Frode; Rajanen, Dorina; Rajanen, Mikko und Rost, Mattias: Perspectives and Practices of Digital Accessibility: A Survey of User Experience Professionals in Nordic Countries, in: *Proceedings of the 11th Nordic Conference on Human-Computer Interaction: Shaping Experiences, Shaping Society*, NordiCHI '20, Association for Computing Machinery, New York, NY, USA, URL https://doi.org/10.1145/3419249.3420119

[Inf24] Informationstechnikzentrum Bund (ITZBund): KoliBri - Public UI. Die barrierefreie Web Component Bibliothek (2020 - 2024), URL https://public-ui.github.io/, Zugriff am: 10. Dezember 2024

[Ini23] Initiative D21 e. V.: D21-Digital-Index 2023/2024, Eine Studie der Initiative D21, durchgeführt von Kantar (2023), URL https://initiatived21.de/publikationen/, Zugriff am: 30. Juli 2024

[JSD24] JSDOM und Contributors: JSDOM Github Repository (2024), URL https://github.com/jsdom/jsdom#readme, Zugriff am: 30. November 2024

[Moz24] Mozilla contributors: Core Learning Path: Semantic HTML, https://developer.mozilla.org/en-US/curriculum/core/semantic-html/ (2024), URL https://developer.mozilla.org/en-US/curriculum/core/semantic-html/, Zugriff am: 08. Oktober 2024

[Nes] Nestler UUX Consulting GmbH: Seminare, Schulungen und Weiterbildung. Barrierefreiheit WCAG BFSG BITV EAA, URL https://bfsg-seminare.de/, Zugriff am: 17. Dezember 2024

[Nur24] Nurthen, James; Cooper, Michael und Henry, Shawn Lawton: *WAI-ARIA Overview*, World Wide Web Consortium (W3C) (2024), URL https://www.w3.org/WAI/standards-guidelines/aria/, Zugriff am: 08. Oktober 2024

[Puz15] Puzis, Yury; Borodin, Yevgen; Soviak, Andrii; Melnyk, Valentyn und Ramakrishnan, I. V.: Affordable web accessibility: a case for cheaper ARIA, in: *Proceedings of the 12th International Web for All Conference*, W4A '15, Association for Computing Machinery, New York, NY, USA, URL https://doi.org/10.1145/2745555.2746657

[staa] stackoverflow.co: 2023 Developer Survey, URL https://survey.stackoverflow.co/2023/#most-popular-technologies-webframe, Zugriff am: 05. August 2024

[Stab] State of JavaScript: 2023 State of JavaScript survey, URL https://2023.stateofjs.com/en-US/, Zugriff am: 05. August 2024

[Sta24] Statistisches Bundesamt (Destatis): Statistisches Bundesamt Deutschland - GENESIS-Online (2024), URL

https://www-genesis.destatis.de/genesis/online?operation=abruftabelleBearbeiten&levelindex=1&levelid=1722343925798&auswahloperation=abruftabelleAuspraegungAuswaehlen&auswahlverzeichnis=ordnungsstruktur&auswahlziel=werteabruf&code=22711-0002&auswahltext=&werteabruf=Werteabruf#abreadcrumb, Zugriff am: 30. Juli 2024

[Swe] Sweet, John: Accessibility in Single Page Apps (Part 1), URL https://johnsweetaccessibility.com/2020/05/accessibility-in-spas-part-1/, Zugriff am: 05. August 2024

[Tea24a] Team Pa11y: Pa11y (2024), URL https://pa11y.org/, Zugriff am: 17. November 2024

[Tea24b] Team Pa11y: Pa11y CI Github Repository (2024), URL https://github.com/pa11y/pa11y-ci, Zugriff am: 17. November 2024

[Tea24c] Team Pa11y: Pa11y Dashboard Github Repository (2024), URL https://github.com/pa11y/pa11y-dashboard, Zugriff am: 17. November 2024

[Tea24d] Team Pa11y: Pa11y Github Repository (2024), URL https://github.com/pa11y/pa11y, Zugriff am: 17. November 2024

[The] The Stepstone Group GmbH: Web-Entwickler/in Gehälter in Deutschland, URL https://www.stepstone.de/gehalt/Web-Entwickler-in.html, Zugriff am: 17. Dezember 2024

[The16] Thesmann, Stephan: *Interface Design*, Springer Fachmedien, Wiesbaden (2016), URL http://link.springer.com/10.1007/978-3-658-03857-1

[Vie15] Vieritz, Helmut: *Barrierefreiheit im virtuellen Raum: Benutzungszentrierte und modellgetriebene Entwicklung von Weboberflächen*, Springer Fachmedien, Wiesbaden (2015), URL https://link.springer.com/10.1007/978-3-658-10704-8

[Web24a] WebAIM: WAVE API (2024), URL https://wave.webaim.org/api/

[Web24b] WebAIM: WAVE Browser Extensions (2024), URL https://wave.webaim.org/extension/, Zugriff am: 17. November 2024

[Web24c] WebAIM: WAVE Stand-alone API and Testing Engine (2024), URL https://wave.webaim.org/standalone, Zugriff am: 17. November 2024

[Whi23] White, Kevin; Velleman, Eric; Hansma, Michel und Lange, Vera: Selecting Web Accessibility Evaluation Tools (2023), URL https://www.w3.org/WAI/test-evaluate/tools/selecting/, Zugriff am: 28. Oktober 2024

[Wo 24] Wo wir sind ist vorne: Accessibility Overlays mit Daniela Kubesch, Podcast (2024), verfügbar unter: https://www.podcast.de/episode/

649840847/accessibility-overlays-mit-daniela-kubesch, Zugriff am: 10. Dezember 2024, relevante Stellen ab Minute 20:30

[Wor] World Wide Web Consortium (W3C): Introduction to Understanding WCAG, URL https://www.w3.org/WAI/WCAG21/Understanding/intro#understanding-the-four-principles-of-accessibility, Zugriff am: 22. August 2024

[Wor24a] World Wide Web Consortium (W3C): Accessibility, Usability, and Inclusion, https://www.w3.org/WAI/fundamentals/accessibility-usability-inclusion/ (2024), Zugriff am: 31. Juli 2024

[Wor24b] World Wide Web Consortium (W3C®) and Shadi Abou Zahra: Accessibility Principles, https://www.w3.org/WAI/fundamentals/accessibility-principles/ (2024), Zugriff am: 09. November 2024

[Yam] Yampolsky, Yoni: Why Web Accessibility is Critical for SEO, URL https://accessibe.com/blog/knowledgebase/web-accessibility-and-seo, Zugriff am: 31. Juli 2024

[Zug24] Zugang für alle, Stiftung zur behindertengerechten Technologienutzung: Accessibility-Zertifizierung (2024), URL https://access-for-all.ch/leistungen/zertifizierung/#kostenuebersicht, Zugriff am: 11. November 2024

Abkürzungsverzeichnis

API Application Programming Interface

ARIA Accessible Rich Internet Applications

AT Assistive Technologien

BFSG Barrierefreiheitsstärkungsgesetz

BGG Behindertengleichstellungsgesetz

BITV Barrierefreie-Informationstechnik-Verordnung

CI/CD Continuous Integration/Continuous Delivery

CLI Command-Line-Interface

DIN Deutsches Institut für Normung

DOM Document Object Model

EN Europäische Normen

HTML Hypertext Markup Language

IKT Informations- und Kommunikationstechnologien

JSON JavaScript Object Notation

REST Representational State Transfer

RIA Rich Internet Application

SEO Search Engine Optimization

SPA Single Page Application

UI User Interface

URL Uniform Resource Locator

UX User Experience

W3C World Wide Web Consortium

WAI Web Accessibility Initiative

WCAG Web Content Accessibility Guidelines

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

.

A Quellcode

A.1 AccessibilityTester.ts

```
1  import { render } from "@testing-library/react";
2  import { writeFileSync, readFileSync } from "fs";
3  import { axe, toHaveNoViolations } from "jest-axe";
4  import { join } from "path";
5
6  // Add jest-axe matcher to Jest
7  expect.extend(toHaveNoViolations);
8
9  export abstract class AccessibilityTester {
10   public static async testElement(key: string, element:
         React.ReactElement) {
11     return test(`${key} - Accessibility Test`, async () => {
12
13       const { container } = render(element);
14
15       // Run accessibility test using axe
16       const results = await axe(container);
17
18       // Prepare violations in a simplified format
19       const violations = results.violations.map((violation) => ({
20         id: violation.id,
21         impact: violation.impact,
22         description: violation.description,
23         help: violation.help,
24         helpUrl: violation.helpUrl,
25         nodes: violation.nodes.map((node) => ({
26           target: node.target,
27           failureSummary: node.failureSummary,
28           html: node.html,
29         })),
30       }));
31
32       const result = {
33         component: key,
34         success: violations.length === 0,
35         violations: violations.length > 0 ? violations : undefined,
36       };
37
38       // save results to file for documentation
39       const fileName = "accessibility-test-results.json";
40       const dirPath = join(__dirname, fileName);
41
42       // Read existing file content
43       let fileContent: Array<typeof result> = [];
44       try {
45         const data = readFileSync(dirPath, { encoding: "utf-8" });
46         fileContent = JSON.parse(data);
47       } catch (e) {
48         // If file doesn't exist or is invalid, start with an empty array
49         fileContent = [];
50       }
51
52
```

```
53        // Check if component already exists
54        const index = fileContent.findIndex(item => item.component ===
              key);
55
56        if (index !== -1) {
57          // Replace existing result
58          fileContent[index] = result;
59        } else {
60          // Append new result
61          fileContent.push(result);
62        }
63
64        // Write updated results back to the file
65        writeFileSync(dirPath, JSON.stringify(fileContent, null, 2), {
              encoding: "utf-8" });
66
67        // Use jest-axe matcher to check for violations
68        expect(results).toHaveNoViolations();
69      });
70    }
71 }
```

Listing A.1: AccessibilityTester-Klasse zur Automatisierung von Barrierefreiheitstests

A.2 ActionButton.test.tsx

```
1  import { act } from "react";
2  import { render, screen, waitFor } from "@testing-library/react";
3  import userEvent from "@testing-library/user-event";
4  import ActionButton from "./ActionButton"
5  import { AccessibilityTester } from
       "../../util/test-utils/AccessibilityTester";
6
7
8  describe("Testing ActionButton", () => {
9    test("Should be rendered to the screen", async () => {
10     await act(async () => {
11       render(<ActionButton items={[]} />)
12     })
13
14     expect(await
           screen.findByTestId("actionbutton.menu")).toBeInTheDocument();
15   });
16
17   test("Should be rendered with no children", async () => {
18     await act(async () => {
19       render(<ActionButton items={[]} />)
20     })
21     expect((screen.queryByTestId("actionbutton.item")))
22       .not.toBeInTheDocument();
23   });
24
25   test("Should be rendered with two children", async () => {
26     await act(async () => {
27       render(<ActionButton items={[
28         { name: "test1", action: () => { } },
29         { name: "test2", action: () => { } }
30         ]} />)
31     })
32
33     const menu = await screen.findByTestId("actionbutton.menu")
34
35     await act(async () => {
36       userEvent.click(menu);
37     })
38
39     waitFor(() => {
40       expect(screen.queryAllByTestId("actionbutton.item").length).toBe(2);
41     });
42   });
43
44   // Accessibility test using AccessibilityTester
45   AccessibilityTester.testElement("ActionButton", (
46   <ActionButton items={[]} />
47   ));
48 })
```

Listing A.2: Integration in bestehende Unit-Tests

A.3 Parent CI/CD-Pipeline (gitlab-ci.yml)

```yaml
image: openjdk:18

variables:
  DEPLOY_STRATEGY:
  value: "NONE"
  options:
    - "PRODUCTION"
    - "STAGING"
    - "NONE"
  description: "The deployment target. Set to 'NONE' by default."

stages:
  - client_pipeline
  - maven_lifecycle
  - dtrack
  - deploy_decision
  - deploy_server
  - deploy_virtual_machine_service
  - deploy_ticket_service

client_pipeline:
  stage: client_pipeline
  resource_group: "child_pipeline"
  variables:
  CHILD_PIPELINE_RUNNER_TAG: "my-runner-tag"
  CHILD_PIPELINE_EXECUTION_CONTEXT: "client"
  PARENT_TRIGGER: $CI_PIPELINE_SOURCE
  DEPLOY_STRATEGY: $DEPLOY_STRATEGY
  trigger:
  include: .sub-gitlab-ci.yml
  strategy: depend
  rules:
  - if: $CI_PIPELINE_SOURCE == "push"
    changes: [client/**/*]
    when: on_success
  - if: $MERGE_REQUEST_ID
    when: on_success
  - if: $CI_PIPELINE_SOURCE == "parent_pipeline"
    when: on_success
  - if: $CI_PIPELINE_SOURCE == "web"
    when: on_success
  - if: $DEPLOY_STRATEGY == "PRODUCTION"
    when: on_success
  - if: $DEPLOY_STRATEGY == "STAGING"
    when: on_success

maven_lifecycle:
  # Compiles, builds, and packages the Java backend services using Maven.

dtrack:
  # Performs dependency vulnerability analysis for the backend services.

deploy_server:
  # Deploys the server application to the target environment.

deploy_virtual_machine_service:
  # Deploys the virtual machine service to the target environment.

deploy_ticket_service:
  # Deploys the ticket service to the target environment.
```

Listing A.3: Parent-CI/CD-Pipeline (gitlab-ci.yml)

A.4 Client CI/CD-Pipeline (gitlab-ci.yml)

```
image: node:20

stages:
  - test
  - build
  - deploy
  - sonar
  - dtrack

cache:
  key: NPM-CACHE-${CI_COMMIT_REF_SLUG}
  paths:
    - .npm

variables:
  INSTALL_DEPS: npm ci --cache .npm --prefer-offline

test:
  stage: test
  tags:
  - my-runner-tag
  script:
  - $INSTALL_DEPS
  - npm run test

build:
  # Compiles and bundles the source code into an executable version.

sonar:
  # Performs code analysis for quality and vulnerabilities.

dtrack:
  # Checks dependencies for known security issues.

deploy:
  # Publishes the application to the target environment.
```

Listing A.4: Client-CI/CD-Pipeline (gitlab-ci.yml)

A.5 ActionButton.tsx (Ausgangszustand)

```tsx
import { Key } from 'react'
import { Dropdown } from 'react-bootstrap';

export interface ActionButtonProps {
  label?: string;
  items: Array<ActionButtonItem>;
  marginStart?: string;
  marginEnd?: string;
  marginTop?: string;
  marginBottom?: string;
  disabled?: boolean;
}

export type ActionButtonItem = {
  name: string;
  action?: (value?: object) => object | void | undefined;
};

export default function ActionButton(props: ActionButtonProps) {
  return (
    <Dropdown data-testid="actionbutton">
      <Dropdown.Toggle
        variant="outline-primary"
        id="dropdown-basic"
        disabled={props.disabled}
      >
        {props.label}
      </Dropdown.Toggle>
      <Dropdown.Menu
        data-testid="actionbutton.menu"
        renderOnMount
        popperConfig={{ strategy: "fixed" }}
      >
        {props.items.map((item) => {
          return (
            <Dropdown.Item
              onClick={() => handleClick(item.name)}
              data-testid="actionbutton.item"
              key={item.name}
            >
              {item.name}
            </Dropdown.Item >
          );
        })}
      </Dropdown.Menu>
    </Dropdown>
  );

  function handleClick(key: Key) {
    props.items.forEach((item) => {
      if (item.name === key && item.action !== undefined) {
        item.action();
      }
    });
  }
}
```

Listing A.5: ActionButton.tsx (Ausgangszustand)

A.6 accessibility-test-results.json (Ausgangszustand)


```
[
  {
  "component": "ActionButton",
  "success": false,
  "violations": [
    {
    "id": "button-name",
    "impact": "critical",
    "description": "Ensures buttons have discernible text",
    "help": "Buttons must have discernible text",
    "helpUrl": "https://dequeuniversity.com/rules/axe/4.9/",
    "nodes": [
    {
      "target": ["#dropdown-basic"],
      "failureSummary": "Fix any of the following:
      Element does not have inner text that is visible to screen
      readers
      Element has no title attribute
      Element's default semantics were not overridden with role=\"none\"
          or role=\"presentation\"",
      "html":
      "<button
      type=\"button\"
      id=\"dropdown-basic\"
      aria-expanded=\"false\"
      class=\"dropdown-toggle
      btn
      btn-outline-primary\"
      ></button>"
    }
    ]
      },
      {
    "id": "aria-prohibited-attr",
    "impact": "serious",
    "description": "Ensures ARIA attributes are not prohibited for an
        element's role",
    "help": "Elements must only use permitted ARIA attributes",
    "helpUrl": "https://dequeuniversity.com/rules/axe/4.9/",
    "nodes": [
      {
        "target": [".dropdown-menu"],
        "failureSummary": "Fix all of the following:
          aria-labelledby attribute cannot be used on a div with no
              valid role attribute.",
      "html":
        "<div
          data-testid=\"actionbutton.menu\"
          x-placement=\"bottom-start\"
          class=\"dropdown-menu\"
          aria-labelledby=\"dropdown-basic\"
         ></div>"
      }
    ]
    }
  ]
  }
]
```


Listing A.6: accessibility-test-results.json (Ausganszustand)

A.7 ActionButton.tsx (barrierefrei)

```
1  import { Key } from 'react'
2  import { Dropdown } from 'react-bootstrap';
3
4  export interface ActionButtonProps {
5    label?: string;
6    items: Array<ActionButtonItem>;
7    marginStart?: string;
8    marginEnd?: string;
9    marginTop?: string;
10   marginBottom?: string;
11   disabled?: boolean;
12 }
13
14 export type ActionButtonItem = {
15   name: string;
16   action?: (value?: object) => object | void | undefined;
17 };
18
19 export default function ActionButton(props: ActionButtonProps) {
20   return (
21   <Dropdown data-testid="actionbutton">
22   <Dropdown.Toggle
23   variant="outline-primary"
24   id="dropdown-basic"
25   disabled={props.disabled}
26   aria-label={props.label || "Action Button"}
27   >
28   {props.label}
29   </Dropdown.Toggle>
30
31   <Dropdown.Menu
32   data-testid="actionbutton.menu"
33   renderOnMount
34   popperConfig={{ strategy: "fixed" }}
35   role="menu"
36   >
37   {props.items.map((item) => {
38       return (
39       <Dropdown.Item
40       onClick={() => handleClick(item.name)}
41       data-testid="actionbutton.item"
42       key={item.name}
43       role="menuitem"
44       >
45       {item.name}
46       </Dropdown.Item >
47       );
48     })}
49   </Dropdown.Menu>
50   </Dropdown>
51   );
52
53   function handleClick(key: Key) {
54     props.items.forEach((item) => {
55       if (item.name === key && item.action !== undefined) {
56         item.action();
57       }
58     });
59   }
60 }
```

Listing A.7: ActionButton.tsx (barrierefrei)

A.8 accessibility-test-results.json (barrierefrei)

```
[
  {
  "component": "ActionButton",
  "success": true
  }
]
```

Listing A.8: accessibility-test-results.json (barrierefrei)