Bachelorarbeit im Studiengang Media Systems an der Hochschule für Angewandte Wissenschaften Hamburg

Version: 1.1 Datum: 13.12.2022

Titel der Arbeit

Entwicklung einer Visual Studio Erweiterung zur Anzeige von Webseitenelementen und Überprüfung der Barrierefreiheit im IDE-Editor

Student

Tom Klanthe

Matrikelnummer [redacted]

Zusammenfassung

Das Einhalten der Barrierefreiheit ist ein wichtiger Bestandteil auf modernen Webseiten. Eingeschränkte Personengruppen werden damit bei der Bedienung unterstützt.

Das Ziel dieser Bachelorarbeit ist es, eine Visual Studio Erweiterung zu konzipieren und prototypisch umzusetzen, welche Entwicklern schon während des Entwicklungsprozesses helfen soll, eventuelle Fehler bezüglich der Barrierefreiheit aufzudecken.

Es werden beispielhaft drei Barrierefreiheitsrichtlinien analysiert und umgesetzt: Das Einhalten eines hohen Kontrastverhältnis zwischen Text und Hintergrund, das Einhalten einer sinnvollen Reihenfolge von Überschriftelementen und die korrekte Benennung von Eingabeelementen für den Screenreader.

Das Ergebnis der Arbeit zeigt, dass eine effizientere Webentwicklung mit den richtigen Werkzeugen möglich ist.

Abstract

To support disabled people using the modern web, it is of great importance to meet certain accessibility standards.

This bachelor thesis aims to design and develop a Visual Studio extension, which supports web developers regarding the accessibility during the development process.

Three accessibility standards will be analyzed and implemented: contrast ratio between text and its background, the order of heading elements and the naming of input elements.

The results show that the usage of certain tools makes meeting accessibility standards more efficient.

1 Inhaltsverzeichnis

2 Motivation

Das Einhalten der Barrierefreiheit auf modernen Webseiten ist wichtig, um eingeschränkte Personengruppen, wie z.B. Menschen mit Sehbehinderung, Erblindung, sowie Gehörlose und schwerhörige Menschen, eine komfortable Bedienung solcher Webseiten zu ermöglichen. Der Schritt zur Umsetzung der Barrierefreiheit ist bei der Entwicklung einer Webseite leicht zu übersehen bzw. wird eventuell ignoriert, wenn der Entwickler selbst nicht zu einer der genannten Personengruppen gehört oder sich noch nicht mit der Thematik beschäftigt hat.

Der digitale Informationsaustausch gehört für die meisten Menschen zum Alltag und erfordert das moderne Webapplikation bestimmte Anforderungen erfüllen, um diese möglichst vielen Personen zugänglich zu machen. Neben Performanz und Kompatibilität mit möglichst vielen Endgeräten gehört auch die Einhaltung der Barrierefreiheit dazu.

In der Barrierefreie-Informationstechnik-Verordnung (BITV) wird festgelegt, dass „Informationen und Dienstleistungen öffentlicher Stellen, die elektronisch zur Verfügung gestellt werden, [...] für Menschen mit Behinderungen zugänglich und nutzbar zu gestalten sind.“ [1] Der seit 2005 bestehende BIK BITV-Test dient dazu die Barrierefreiheit von Webauftritten zu evaluieren und führt detailliert auf, wie Webseiten auf Barrierefreiheit überprüft werden müssen. [2]

Im Gegensatz zur BITV, welche die deutsche Norm für barrierefreies Webdesign definiert, ist in den Web Content Accessibility Guidelines (WCAG) der internationale Standard definiert. Dieser befindet sich seit Juni 2018 in der Version 2.1. [3]

Auf europäischer Ebene befindet sich die vom europäischen Institut für Telekommunikationsnormen entwickelte Norm EN 301 549 in der seit Februar 2022 geltenden Version 3.2.1. [4]

Um das Einhalten dieser Standards zu gewährleisten, wurden zahlreiche Werkzeuge entwickelt, welche es Entwicklern ermöglichen ohne viel Mehraufwand zu überprüfen, ob und zu welchem Grad eine entwickelte Webseite barrierefrei ist. Im Folgenden eine kleine Auswahl an populären Werkzeugen:

- Google Lighthouse ist in den Google Chrome Entwicklerwerkzeugen integriert. Dieses Werkzeug testet fertige Webseiten nicht nur auf Barrierefreiheit, sondern auch auf Performance, Best Practices, User Experience und Search Engine Optimization (SEO). [5] Ein Nachteil jedoch ist, dass die Analyse einer Webseite zeitaufwendig ist (bis zu einer Minute pro Analyse) und jede Verschachtelung einer Webseite erneut geprüft werden muss. Außerdem ist das Werkzeug nicht in anderen gängigen Browsern vorhanden, die ebenfalls unterstützt werden sollten.

- WAVE Web Accessibility Evaluation Tool ist ein Werkzeug, welches jede beliebige Webseite analysiert und direkt im Browser auf mögliche Fehlerquellen in Bezug auf Barrierefreiheit hinweist. [6] Der Nachteil hier ist, dass die Webseite, die getestet werden soll, öffentlich verfügbar sein muss. Somit ist es für die laufende Entwicklung oftmals nicht zu gebrauchen.
- NVDA ist ein Werkzeug für sehbehinderte Menschen, welches nicht nur Webinhalte, sondern sämtliche Inhalte des Betriebssystems in hörbarer Form wiedergibt. Für dieses Werkzeug ist es wichtig, dass Webseiteninhalte korrekt gegliedert sind, damit beim Navigieren mit der Tabulator-Taste Elemente in der richtigen Reihenfolge vorgelesen werden und das Navigieren erleichtert wird. [7] Diese Art von Werkzeug wird als Screenreader bezeichnet, da es Bildschirminhalte vorliest.

Diese Werkzeuge sind entweder kontraintuitiv und zeitaufwendig zu benutzen, bzw. stehen nur bedingt im Entwicklungsprozess einer Webseite zur Verfügung. Demnach könnte es sinnvoll sein, bestimmte Tests zur Barrierefreiheit dem Entwickler direkt im Entwicklungsprozess zugänglich zu machen, damit dieser somit diesen wichtigen Teilbereich der Webentwicklung abdecken kann.

Der momentan übliche Entwicklungsprozess besteht aus den Schritten Entwickeln, Testen und Nachbessern. Im Thema Barrierefreiheit kann man das Testen und Nachbessern deutlich einkürzen, wenn schon während der Entwicklung Schwächen im Code aufgezeigt und damit behoben bzw. umgangen werden.

Durch die Entwicklung eines weiteren Werkzeugs, welches direkt im Editor einer IDE (integrated development environment) eventuelle Probleme der Barrierefreiheit aufzeigt, könnte effizienter entwickelt werden. Das oft zeitintensive Kompilieren des Codes nach jeder Änderung entfällt. Maßnahmen, welche der Barrierefreiheit dienen, können somit schneller getroffen werden. Durch das Aufzeigen von problematischen Stellen kann ein Entwickler so zielgerichtet Lösungen finden.

3 Zentrale Zielsetzung der Arbeit

Das Ziel dieser Bachelorarbeit ist es, eine Visual Studio Erweiterung zu konzipieren und prototypisch umzusetzen, welche es Entwicklern ermöglicht, direkt in der IDE Webseiteninhalte auf Barrierefreiheit zu überprüfen. Die folgenden Teilbereiche der Barrierefreiheit sind zu prüfen:

1. **Kontrastverhältnisse**: Die Wahl von Farben, welche einen ausreichenden Kontrast zueinander besitzen ermöglicht nicht nur beeinträchtigten Personen die Bedienung einer Webanwendung, sondern gestaltet auch die Bedienung für nicht-beeinträchtigten Personen angenehmer.

2. **Reihenfolge von Überschriften**: Die korrekte Reihenfolge von Überschriften und die Benennung von HTML-Elementen ist wichtig, damit sehbehinderten Menschen das Navigieren, mit z.B. Werkzeugen wie NVDA, problemlos ermöglicht wird.
3. **Kennzeichnung von Eingabefeldern**: Eingabefelder müssen immer gekennzeichnet werden damit Benutzer erkennen können, welche Eingabe von Ihnen erwartet wird. Gerade sehbehinderten Menschen wird die Benutzung somit vereinfacht, bzw. erst möglich gemacht.

Zur Untersuchung dieser Punkte wird jeweils eine bestehende HTML-Struktur analysiert. Es werden zunächst nur statische Inhalte überprüft. Erst wenn diese Untersuchung erfolgreich abgeschlossen wurde, sollten dynamische Inhalte einbezogen werden, die einen erheblichen Mehraufwand verursachen.

Die Überprüfung soll der Benutzer, hier der Entwickler der Webanwendung, bei einem Klick auf eine Schaltfläche in der IDE ausführen. Ein geöffnetes HTML-Dokument wird dann auf die oben genannten Punkte bzgl. der Barrierefreiheit geprüft und die Ergebnisse direkt in der Fehlerliste der IDE angezeigt.

Dadurch das eventuelle Fehler der Barrierefreiheit direkt in der IDE angezeigt werden, entfällt das Überprüfen von nicht-dynamischen Inhalten im Web Browser. Dynamische Inhalte müssten immer noch manuell überprüft werden, da es nicht möglich ist solche Inhalte ohne z.B. Daten, welche von einer Datenbank bezogen werden, zu prüfen.

Aufgrund der hohen Verbreitung von Microsoft Visual Studio, die mit 29,71% Marktanteil die weltweit am häufigsten benutzte IDE ist, und persönlicher Erfahrung des Autors, wird die Erweiterung für Microsoft Visual Studio konzipiert und prototypisch umgesetzt. [8]

4 Vorgehensweise

Anfangs werden die Grundlagen der Barrierefreiheit für moderne Webapplikationen erläutert: Es wird dargestellt, wie die Gesetzeslage ist und welche modernen Möglichkeiten und Werkzeuge es gibt, um die Barrierefreiheit zu gewährleisten und zu Testen. Auch sollen internationale und nationale Richtlinien wie die WCAG und die BITV analysiert werden, welche das Verständnis der Thematik vertiefen sollen. Dies wird dann auch eine effizientere Umsetzung der Visual Studio Erweiterung ermöglichen.

Darauf folgt die Konzeption, in der die Umsetzung der Erweiterung definiert wird. Hauptsächlich werden hier die Architektur, die Datenmodellierung, das Design der Benutzeroberfläche und die Benutzbarkeit festgelegt und erläutert. Da die Gewährleistung eines barrierefreien Webdesign sehr umfangreich ist, wird evaluiert welche spezifischen Vorgaben nach WCAG in der Visual Studio

Erweiterung geprüft werden sollen, bzw. in welchem Umfang dies überhaupt möglich ist, da wie oben beschrieben nur nicht-dynamische Bereiche einer Webseite geprüft werden können.

Anschließend wird das Konzept prototypisch umgesetzt.

Zum Schluss folgt eine Auswertung, ob eine Erkennung der möglichen Fehlerquellen bzgl. der Barrierefreiheit während der Entwicklung anhand des konzipierten und umgesetzten Prototyps möglich ist.

Die Arbeit endet mit einer Zusammenfassung, einem Fazit, und gibt einen Ausblick auf zukünftige Weiterentwicklungen.

5 Gesetzeslage und Richtlinien von Barrierefreiheit im modernen Web

Barrierefreiheit ist ein wichtiges und breitgefächertes Thema. Im Folgenden wird aufgeführt welche Kernpunkte dieses Themas im modernen Web in Bezug auf die aktueller Gesetzeslage und globaler Regelungen bzw. Empfehlungen von unabhängigen Institutionen von besonderer Wichtigkeit sind.

5.1 WCAG

Die Web Content Accessibility Guidelines (WCAG) definieren den internationalen Standard für barrierefreies Webdesign, welche sich seit Juni 2018 in der Version 2.1 befindet. [3] Der WCAG-Standard wurde von dem World Wide Web Consortium (W3C) definiert. Das W3C ist eine 1994 gegründetes Gremium welches technische Webstandards definiert und entwickelt, die den einfacheren Zugang zu einer einheitlichen Webgestaltung ermöglichen. [9] Ein Beispiel eines solchen Standards ist der Webbrowser. Die meisten benutzen den W3C-Standard, welcher ihnen ermöglicht Hypertext Markup Language (HTML) und Cascading Style Sheets (CSS) zu interpretieren.

In der WCAG wird konkret definiert welche Maßnahmen von Webentwicklern getroffen werden müssen, um barrierefreie Webinhalte zu gewährleisten. [3] Hierbei wird sich an eine weitreichende Personengruppe mit Behinderungen gerichtet: Menschen mit Erblindung und unzureichender Sicht, Gehörlose, Menschen mit eingeschränkter Bewegung, Sprachbehinderte, Photosensitive sowie die Kombination aller dieser genannten Behinderungen. Die WCAG gelten für Webanwendungen auf Desktops, Laptops, Tablets und mobile Geräte. Maßnahmen, die nach WCAG getroffen werden, erleichtern auch die Bedienung von Menschen ohne Einschränkungen.

Die WCAG 2.1 besteht aus vier Grundprinzipien mit 13 Richtlinien. [10] Die Grundprinzipien stellen die Basis barrierefreier Webgestaltung dar, die Richtlinien geben klare Vorgaben an, an denen sich Webentwickler- und Designer orientieren können:

1. Wahrnehmbarkeit

    a. Textalternativen: Bereitstellen von Texten für Nicht-Text-Inhalte
    b. Zeitbasierte Medien: Bereitstellen von Alternativen für Audio- und Videoinhalte (Untertitel, Gebärdensprache etc.)
    c. Anpassbar: Inhalte die auf unterschiedlichen Arten dargestellt werden können (z.B. verschiedene Bildschirmauflösungen)
    d. Unterscheidbar: Inhalte sind leicht zu trennen (z.B. Vorder- und Hintergrund)
2. Bedienbarkeit
    a. Per Tastatur zugänglich: Funktionalitäten per Tastatureingabe ermöglichen
    b. Ausreichend Zeit: Zeit, um Inhalte zu lesen und benutzen
    c. Anfälle und physische Reaktionen: Gestaltung von Inhalten die zu keiner physischen Reaktion führen (z.B. Epilepsie)
    d. Navigierbar: Erleichterung bei der Navigation von Inhalten (z.B. Fokus Reihenfolge)
    e. Eingabemodalitäten: Andere Eingabemöglichkeiten als Tastatur ermöglichen (z.B. Mauszeigergesten)
3. Verständlichkeit
    a. Lesbar: Inhalte lesbar und verständlich gestalten
    b. Vorhersehbar: Vorhersehbares Aussehen und Verhalten von Inhalten
    c. Hilfestellung bei der Eingabe: Vermeidung von Fehlern der Benutzer durch Hilfestellungen an Eingabefeldern
4. Robustheit
    a. Kompatibel: Kompatibilität zu möglichst vielen Benutzeragenten (Webbrowsern)

Jede dieser Richtlinien erhält unterschiedliche Erfolgskriterien. Insgesamt befinden sich in der Version 2.1 der WCAG 78 Erfolgskriterien. Diese werden in drei Konformitätsstufen unterteilt:

1. Konformitätsstufe A (30 Erfolgskriterien)
2. Konformitätsstufe AA (20 weitere Erfolgskriterien)
3. Konformitätsstufe AAA (28 weitere Erfolgskriterien, also insgesamt 78)

5.2 EN 301 549

Der europäische Standard EN 301 549 ist eine Norm, welche vom europäischen Institut für Telekommunikationsnormen geschaffen wurde, um Anforderungen an barrierefreies Webdesign und Erstellung von Testmechanismen auf europäischer Ebene zu spezifizieren. [11] Seit Februar besteht diese Norm in der Version 3.2.1. Seit Version 2.1.2 wird die WCAG 2.1 Standards mit in die Norm aufgenommen. [12] In der Norm werden auf europäischer Ebene die Mindestanforderungen an die Barrierefreiheit der Informations- und Kommunikationstechnologien beschrieben.

50 Erfolgskriterien der WCAG (Konformitätsstufe AA) müssen erfüllt sein, um der EN 301 549 zu entsprechen.

5.3 BITV

Die Barrierefreie-Informationstechnik-Verordnung (BITV) bestimmt die Schaffung barrierefreier Informationstechnik nach dem Behindertengleichstellungsgesetz. Es soll gewährleistet werden das öffentlich zugängliche Webauftritte des Bundes für eingeschränkte Nutzergruppen benutzbar sind. Die Verordnung gilt für Webseiten, mobile Anwendungen, elektronisch gestützte Verwaltungsabläufe sowie grafische Programmieroberflächen. [1]

Menschen nach §3 des Behindertengleichstellungsgesetzes sollen eine uneingeschränkte Nutzung von Informationstechnik ermöglicht werden. [1]

Die Verordnung besteht seit 2002, die neueste Fassung in der Version 2.0 besteht seit 2011. [1]

Die BITV bezieht sich konkret auf die Prioritätsstufen der Priorität 1 der WCAG in der Version 1.0.

Im Gegensatz zur WCAG ist die BITV in zwei Prioritäten aufgeteilt: [13]

1. Priorität 1 (Muss): Beschreibt Anforderungen, die eine Webseite erfüllen muss, um BITV-konform zu sein. Sie definieren unerlässliche Mindestanforderungen.
2. Priorität 2 (Soll): Beschreibt Anforderungen, die erfüllt werden sollten, wenn es sich um zentrale Einstiegsangebote handelt.

Die folgende Tabelle zeigt die Kriterien der BITV aufgeteilt nach ihren zwei Prioritätsstufen und nach den Grundprinzipien. [13]

Grundprinzip	Priorität 1 (Muss)	Priorität 2 (Soll)
Wahrnehmbarkeit	▪ Alternativen für jeden Nicht-Text-Inhalt ▪ Alternativen für zeitgesteuerte Medien (Text-Alternativen, Tonspur, Untertitel) ▪ Inhalte ohne Informations- oder Strukturverlust (Informationen, Strukturen, Beziehungen, Reihenfolge, Merkmale) ▪ Wahrnehmung des Inhalts und Unterscheidung zwischen Vorder- und Hintergrund (Farbe, Audio-Kontrolle, Kontrast mindestens 4,5:1 bzw. bei	▪ Alternativen für zeitgesteuerte Medien (Gebärdensprache, erweiterte Audio-Deskription, Volltext-Alternative, Live-Audio-Inhalte) ▪ Wahrnehmung des Inhalts und Unterscheidung zwischen Vorder- und Hintergrund (Kontrast mindestens 7:1, bei Großschrift mindestens 4,5:1, keine oder abschaltbare Hintergrundgeräusche, bei visueller Präsentation von Textblöcken Vorder- und Hintergrundfarben auswählbar, Zeilenbreite maximal 80 Zeichen, Text nicht in Blocksatz,

Grundprinzip	Priorität 1 (Muss)	Priorität 2 (Soll)
	Großschrift mindestens 3:1, veränderbare Textgröße mindestens bis auf 200%, Vermeiden von Schriftgrafiken)	Zeilenabstand mindestens 1,5 Zeilen, Abstand zwischen Absätzen größer als Zeilenabstand, Text im Vollbildmodus mindestens bis auf 200% vergrößerbar, Vermeiden von Schriftgrafiken)
Bedienbarkeit	▪ Tastatur-Zugänglichkeit (Tastaturbedienbarkeit, keine Tastaturfalle) ▪ ausreichend Zeit zum Lesen und Verwenden (zeitbezogene Anforderungen wie ausschalten, verlängern, Hinweis auf Zeitablauf, Bewegungen anhalten, beenden und ausblenden) ▪ keine epileptischen Anfälle erzeugen (Ausnahme: "general flash / red flash"-Schwellen) ▪ Orientierungs- und Navigationshilfen (Elementgruppen, Webseiten-Titel, Fokus-Reihenfolge, Zweck eines Links im Kontext, alternative Zugangswege, Beschriftungen wie Überschriften und Label, sichtbarer Fokus, Standort)	▪ Tastatur-Zugänglichkeit (Tastaturbedienbarkeit) ▪ ausreichend Zeit zum Lesen und Verwenden der Inhalte (keine Zeitbegrenzung, Unterbrechungen aufschieben oder unterdrücken, Wiederanmeldung ohne Datenverlust) ▪ keine epileptischen Anfälle auslösen ▪ Orientierungs- und Navigationshilfen (Zweck eines Links aus Linktext ersichtlich, Abschnittsüberschriften)
Verständlichkeit	▪ lesbare und verständliche Texte (vorherrschende Sprache) ▪ Aufbau und Benutzung sind klar (Kontext-Änderung bei Fokussierung oder Eingabe nur nach entsprechendem Hinweis, einheitliche Navigation und Bezeichnung) ▪ Fehlervermeidung und -korrektur (Fehleridentifizierung, Beschriftungen, Korrekturvorschläge, Fehlervermeidung)	▪ Texte sind lesbar und verständlich (ungebräuchliche Wörter und Abkürzungen erläutern, einfache Sprache, ansonsten zusätzliche erklärende Inhalte, korrekte Aussprache aufzeigen) ▪ Aufbau und Benutzung vorhersehbar (Kontextänderungen nur auf Nutzer-Anforderung oder abschaltbar) ▪ Fehlervermeidung und -korrektur (kontextabhängige Hilfen, Fehlervermeidung durch Möglichkeiten wie rückgängig machen, korrigieren und bestätigen)

Grundprinzip	Priorität 1 (Muss)	Priorität 2 (Soll)
Robustheit	▪ Kompatibilität mit Benutzeragenten (Syntaxanalyse, Name, Rolle, Wert)	

Der Vorteil der WCAG gegenüber der BITV ist das es in 3 unterschiedliche Prioritätsgruppen unterteilt wird, anstatt nur 2. Dies ermöglicht einen stufenweisen Aufbau barrierefreier Webseiten.

Im Gegensatz zur WCAG ist die BITV rechtsverbindlich. Behörden und öffentliche Institutionen sind dazu verpflichtet ihre Webauftritte barrierefrei nach der BITV zu gestalten.

5.4 BIK BITV-Test

Der seit 2005 bestehende BIK BITV-Test dient dazu, die Barrierefreiheit von Webauftritten zu evaluieren und führt detailliert auf, wie Webseiten auf Barrierefreiheit überprüft werden müssen. [2]

Der Test basiert auf Basis der BITV 2.0 bzw. EN 301 549.

Auf der Webseite des BITV befindet sich ein Verzeichnis der Prüfschritte und detaillierte Beschreibungen der Prüfverfahren, mit denen Webseiten überprüft werden. Es gibt insgesamt 98 Prüfschritte. Zusätzlich zu den WCAG 2.1 Prüfschritten gibt es 38 weitere Anforderungen, um Webinhalte gemäß EN 301 549 umzusetzen.

Neben dem BITV-Test wird auch ein BIK WCAG-Test angeboten um Webseiten auf Barrierefreiheit nach WCAG 2.1 zu Testen.

Auch finden Webentwickler externe Prüfstellen, welche man mit einer Prüfung seiner Lösung beauftragen kann.

Jeder Prüfschritt beinhaltet Informationen darüber was geprüft wird, warum es geprüft wird und wie es geprüft wird, sowie welche Kriterien erfüllt werden müssen, um bestimmte Webseitenelemente barrierefrei zu gestalten. Auch wird jeder Prüfschritt nach WCAG 2.1 eingeordnet und die WCAG-Bestimmungen direkt für den einfachen Zugriff verlinkt. Dies verringert die Zeit, die von Entwicklern aufgebracht werden muss, um die jeweiligen Prüfschritte direkt in der WCAG zu recherchieren, verstehen und umzusetzen, besonders da nur die Prüfschritte im BITV-Test vorhanden sind, um die Barrierefreiheit nach BITV umzusetzen. Im Folgenden ist ein Beispiel eines solchen Prüfschritts abgebildet. In jeder der Kategorien befinden sich Details zu dem besagtem Prüfschritt.

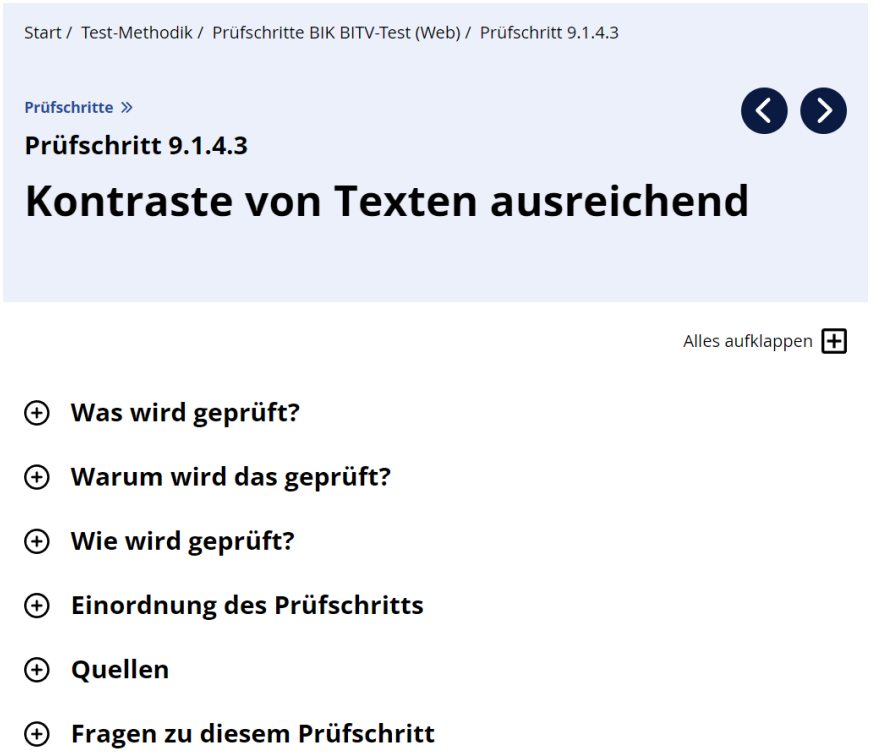


Abbildung 1: Beispiel einer Unterseite eines Prüfschritts des BITV-Tests

Für die Überprüfungsschritte der Visual Studio Erweiterung wird sich auf die Barrierefreiheitspunkte des WCAG 2.1 Standards bezogen. Diese definieren wie die verschiedenen Teilbereiche der Barrierefreiheit überprüft werden müssen.

Nur die Überprüfung der Kontrastverhältnisse ist BITV-konform, da in diesem Teilbereich die Konformitätsstufen AA und AAA überprüft werden. In den anderen beiden Bereichen (Überschriften und Benennung) wird teilweise nur nach Konformitätsstufe A geprüft.

6 Problemstellung und Anwendungsbeispiel

Als Anwendungsbeispiel dient ein Team von Entwicklern in einem mittelgroßen IT-Unternehmen, welches ein webbasiertes Verwaltungswerkzeug entwickelt mit der Anforderung einer barrierefreien Bedienung. In der internen Qualitätssicherung fällt auf, das bestimmte Richtlinien der Barrierefreiheit noch nicht erfüllt sind. Durch die regelmäßige Nutzung der Visual Studio Erweiterung während des Entwicklungsprozesses wird somit sichergestellt das die Richtlinien effizient eingehalten werden.

7 Konzeption der Visual Studio Erweiterung

In der Visual Studio Erweiterung wird auf drei Kernpunkte der Barrierefreiheit nach WCAG eingegangen. Diese drei Teilbereiche wurden gewählt, da sie mit zu den sinnvollsten Bestandteilen

eines barrierefreien Webdesigns gehören. Außerdem ist es technisch möglich, diese im Rahmen der Erweiterung und dieser Bachelorarbeit umzusetzen. Andere Aspekte zur Überprüfung der Barrierefreiheit erfordern z.T. einen enormen technischen Aufwand, bzw. sind nicht möglich umzusetzen, da einige Überprüfungen nicht automatisiert werden können, wie beispielsweise ob zu Videos korrekte Untertitel abrufbar sind.

Die Erweiterung wird sich zudem auf statische Webinhalte beschränken, da die Überprüfung dynamischer Inhalte erheblich mehr Arbeitsaufwand mit sich bringt.

7.1 Kontrastverhältnisse

Bei der Überprüfung der Kontrastverhältnisse wird sich auf das Grundprinzip „Wahrnehmbarkeit“ und dessen Richtlinie „Unterscheidbarkeit“ bezogen.

Die Erweiterung überprüft die WCAG-Punkte: [10]

- 1.4.3 (Konformitätsstufe AA) – Kontrast (Minimum): Die Darstellung von Bildern und Texten weisen ein Kontrastverhältnis von mindestens 4,5:1 auf. Großer Text und Bilder von großem Text weisen ein Kontrastverhältnis von mindestens 3:1 auf.
- 1.4.6 (Konformitätsstufe AAA) – Kontrast (erhöht): Die Darstellung von Bildern und Texten weisen ein Kontrastverhältnis von mindestens 7:1 auf. Großer Text und Bilder von großem Text weisen ein Kontrastverhältnis von mindestens 4,5:1 auf.

Diese beiden Kriterien wurden geschaffen, um den Kontrast zwischen dem Hintergrund und Fließtext zu überprüfen und auf ein Mindestmaß festzulegen, damit moderat sehbehinderte Menschen Inhalte lesen können, ohne externe Software zur Erhöhung der Kontrastverhältnisse nutzen zu können.

Da für farbenblinde Menschen Farbton und Sättigung keine bzw. nur minimale Einschränkungen der Lesbarkeit verursachen, wird der Kontrast zwischen zwei Elementen so berechnet, dass die Farbe kein Schlüsselfaktor der Berechnung darstellt. [14]

Text, welcher nur einen dekorativen Zweck erfüllt und keine Informationen enthält, wird nach WCAG-Standard ignoriert. Ein Beispiel hierfür sind zufällige Wörter, welche für den Hintergrund benutzt werden, jedoch ohne Verlust von Informationen ausgetauscht oder repositioniert werden können. [14]

Größerer Text besteht aus Buchstaben mit dickeren Charakteren und ist somit auch bei niedrigerem Kontrast leichter zu lesen. Deswegen ist für große Texte ein kleinerer Kontrastunterschied von 3:1 nach WCAG-Punkte 1.4.3 und 4,5:1 nach WCAG-Punkt 1.4.6 erforderlich.

Dies gibt Webdesignern die Möglichkeit mit einer größeren Farbpalette für große Texte zu arbeiten, was hilfreich für das Design einer Webseite sein kann, besonders für Titel.

Das Kontrastverhältnis von 3:1 ist das empfohlene Minimum nach ISO-9241-3 und ANSI-HFES-100-1988 für Standardtext. Das Verhältnis von 4,5:1 wurde beschlossen, um den wahrgenommenen Kontrastverlust, welche aus unscharfem Sehen und Farbenblindheit entstehen können, auszugleichen. [14]

Sollten auf einer Webseite Bilder mit Texten vorkommen, so müssen diese auch korrekte Kontrastverhältnisse zum Hintergrund besitzen. Dies gilt z.B. für Texte, welche in Pixel gerendert wurden und dann in einem Bildformat gespeichert wurden. Dies wird und kann von der Erweiterung jedoch nicht überprüft werden, da unklar ist, welcher Pixel im Bild zum Hintergrund und welcher zum Text gehört.

Sollte die Leuchtdichte eines Texts oder des Hintergrunds variieren, z.B. wenn sich eine Textur darauf befindet, so sollte der Text eine Schattierung bzw. der Hintergrund unmittelbar hinter dem Text hinzugefügt werden, um einen WCAG konformen Kontrast zu gewährleisten.

Das Kontrastverhältnis zweier Farben wird wie folgt berechnet: [15]

1. Berechnung der relativen Leuchtdichte **L** jedes Buchstaben (außer sie sind alle uniform) mit der Formel:
   - $\mathbf{L} = 0{,}2126 * \mathbf{R} + 0{,}7152 * \mathbf{G} + 0{,}0722 * \mathbf{B}$

     wobei **R**, **G** und **B** wie folgt definiert werden:
     - Wenn $R_{sRGB} <= 0{,}03928$

       dann $\mathbf{R} = R_{sRGB} / 12{,}92$

       sonst $\mathbf{R} = ((R_{sRGB} + 0{,}055) / 1{,}055)^{2{,}4}$
     - Wenn $G_{sRGB} <= 0{,}03928$

       dann $\mathbf{G} = G_{sRGB} / 12{,}92$

       sonst $\mathbf{G} = ((G_{sRGB} + 0{,}055) / 1{,}055)^{2{,}4}$
     - Wenn $B_{sRGB} <= 0{,}03928$

       dann $\mathbf{B} = B_{sRGB} / 12{,}92$

       sonst $\mathbf{B} = ((B_{sRGB} + 0{,}055) / 1{,}055)^{2{,}4}$

   R_{sRGB}, G_{sRGB} und B_{sRGB} werden wie folgt definiert:

   - $R_{sRGB} = R_{8bit} / 255$
   - $G_{sRGB} = G_{8bit} / 255$
   - $B_{sRGB} = B_{8bit} / 255$

2. Berechnung der relativen Leuchtdichte der Hintergrundpixel direkt neben den Buchstaben mit derselben Formel wie in Schritt 1.
3. Berechnung des Kontrastverhältnisses mit der Formel
    - (L1 + 0,05) / (L2 + 0,05), wobei
        - L1 die relative Leuchtdichte der helleren Vorder- bzw. Hintergrundfarbe darstellt
        - L2 die relative Leuchtdichte der dunkleren Vorder- bzw. Hintergrundfarbe darstellt

Die Abkürzung sRGB steht für „Standard Red Green Blue“ und ist ein Farbraum für Grafiken im Web. Der Farbraum ist Additiv, das bedeutet das Farben durch das Mischen der drei Grundfarben Rot, Grün und Blau gebildet wird. sRGB wurde 1996 von Microsoft und HP kreiert. Später wurde sRGB durch die IEC (International Electrotechnical Commission) standardisiert. [16]

Die folgende Tabelle zeigt drei Beispiele auf, welche die Kontrastverhältnisse visuell darstellen.

Die Farbe eines Textes innerhalb eines `<p>`-Elements (Vordergrund) wird mit der Hintergrundfarbe eines sich darüber liegenden `<div>`-Elements verglichen.

Die Farben wurden über das `style`-Attribut direkt an die HTML-Elements als Hexadezimalwert vergeben.

Farbwerte im Hexadezimalformat werden mit #RRGGBB definiert und stellen den Farbwert in 24 bit dar.

Gerendertes HTML	HEX-Wert Hintergrund	HEX-Wert Vordergrund	Kontrast-verhältnis	WCAG 1.4.3	WCAG 1.4.6
Lorem ipsum dolor sit amet	#61ADD5	#AFFEFC	2,2:1	Nein	Nein
Lorem ipsum dolor sit amet	#000000	#985E3B	4:1	Ja	Nein
Lorem ipsum dolor sit amet	#000000	#FFFFFF	21:1	Ja	Ja

Das HTML des letzteren obigen Beispiels setzt sich wie folgt zusammen:

```
<div style="background-color: #000000">
    <p style="color: #FFFFFF">
        Lorem ipsum dolor sit amet
    </p>
</div>
```

Abbildung 2: HTML einer Textzeile

Mit externen Werkzeugen wie z.B. Colour Contrast Analyser (CCA) kann das Kontrastverhältnis zwischen zwei Farben analysiert werden. CCA zeigt auch an ob und mit welcher Konformitätsstufe Kontrastverhältnisse nach WCAG 2.1 konform sind. [17] Dies ist nützlich, um schnell Kontraste zu prüfen, ohne selbst das Ergebnis mit der vorher genannten Formel zu berechnen.

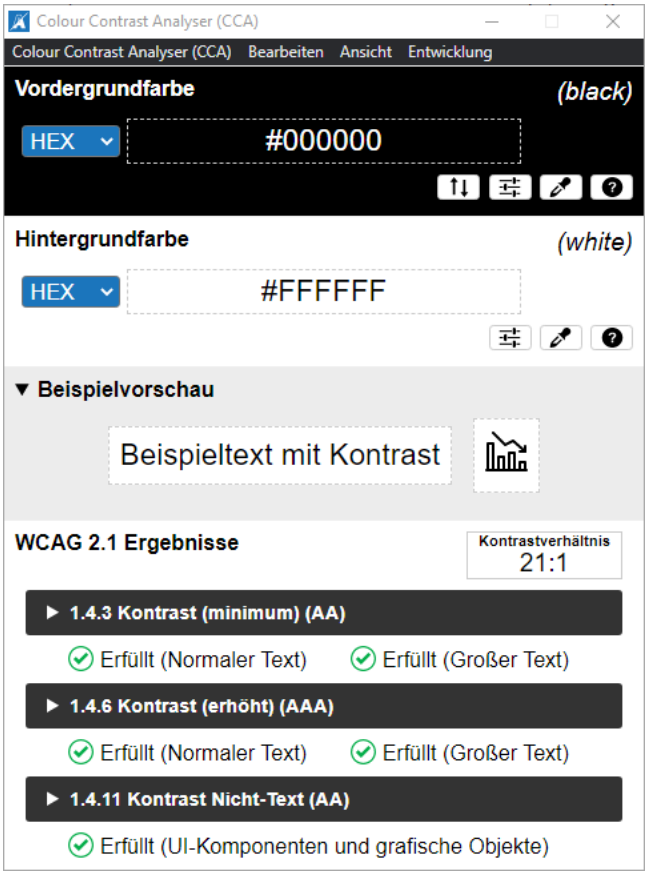


Abbildung 3: Colour Contrast Analyser von TPGi

7.2 Reihenfolge von HTML-Überschriften

Bei der Überprüfung der Reihenfolge von HTML-Elementen wird sich auf das Grundprinzip „Wahrnehmbarkeit“ und dessen Richtlinie „Anpassbar“ bezogen.

Die Erweiterung überprüft die WCAG-Punkte: [10]

- 1.3.1 (Konformitätsstufe A) – Info und Beziehungen: Überschriften müssen korrekt mit den HTML-Strukturelementen `<h1>` bis `<h6>` oder äquivalenten ARIA-Rollen und Attributen ausgezeichnet sein und die Inhalte der Seite erschließen.
- 2.4.10 (Konformitätsstufe AAA) – Abschnittsüberschriften: Überschriften werden genutzt, um den Inhalt zu gliedern.

In HTML gibt es die Überschriftenelemente <h1> bis <h6>. <h1> stellt die höchste Stufe eines Überschriftenelements dar, <h6> die geringste. Optisch unterscheiden sich die Elemente von ihrer Schriftgröße (von <h1> aus absteigend). Diese Elemente werden benutzt, um Webseiteninhalte zu gliedern und somit dem Benutzer das Navigieren zu erleichtern. Die Erweiterung prüft ob Überschriften vorhanden sind, und wenn ja, ob sie sich in der korrekten Reihenfolge befinden.

7.3 Benennung von HTML-Elementen durch Label

Bei der Überprüfung der Benennung von HTML-Elementen wird sich auf das Grundprinzip „Bedienbarkeit“ und dessen Richtlinie „Eingabemodalitäten“ bezogen.

Die Erweiterung überprüft die WCAG-Punkte: [10]

- 2.5.3 (Konformitätsstufe A) – Beschriftung (Label) im Namen: Bei Bestandteilen der Benutzerschnittstelle mit Beschriftungen (Labels), die Text oder Bilder eines Textes enthalten, enthält der Name den Text, der visuell angezeigt wird.
- 2.4.6 (Konformitätsstufe AA) – Überschriften und Label Beschreiben ein Thema oder einen Zweck

Die Erweiterung prüft, ob sich in dem HTML <input>-Elemente befinden, und ob zu jedem dieser Elemente ein dazugehöriges <Label>–Element existiert.

<Label>–Elemente sind nicht nur visuell mit einem <input>–Element verknüpft, sondern auch programmatisch. Das bedeutet das z.B. Screenreader den Text eines Labels ausgeben können, wenn der Benutzer das Eingabefeld fokussiert. Dies macht es für eingeschränkte Benutzer, welche assistierende Technologien benutzen, einfacher zu verstehen welche Daten eingegeben werden müssen.

Außerdem fokussiert der Browser bei einem Klick auf das Label das dazugehörige Eingabefeld. Dies vergrößert die Fläche für die Fokussierung der Eingabe, was Benutzern im Allgemeinen die Bedienung erleichtert, besonders wenn die Inhalte über mobile Geräte aufgerufen werden.

7.4 Überprüfung der Vorgaben

Die drei Vorgaben werden untersucht in dem das HTML, welches sich bei der Entwicklung einer Webseite in dem Editor von Visual Studio befindet, analysiert wird. Per Iteration über den Code werden dann mit Hilfe von regulären Ausdrücken die verschiedenen HTML-Elemente als Text analysiert und die Vorgaben überprüft.

Vor der Analyse geprüft, ob HTML-Elemente verschachtelt sind, damit die sich in der Verschachtelung befindenden Elemente miteinander verglichen werden können. Ein Beispiel dafür ist

die unterschiedliche Einfärbung von zwei nebeneinanderstehenden <`div`>–Containern. Sollte sich Text in den beiden Containern befinden, so darf nur die Farbe des Textes mit der Hintergrundfarbe des jeweiligen Containerelements verglichen werden. Diese Prinzipien gelten auch für die Überprüfung der Überschriften und der Benennung von Eingabefeldern.

Die HTML-Attribute werden mit Hilfe von regulären Ausdrücken aus dem Code extrahiert und somit zur Überprüfung der Vorgaben benutzt.

Sollte die HTML-Struktur gegen diese Vorgaben verstoßen, so wird der Entwickler über die Fehlerliste über die Art des Fehlers, und an welche Stelle er vorkommt, informiert.

8 Funktionsweise für den Anwendungsbenutzer

Der Name der Erweiterung zum Testen der Barrierefreiheit wird „AccessibilityChecker“, kurz AC, sein.

Zum Installieren des AC muss die Datei `AccessibilityChecker.vsix` geöffnet werden. Wenn Visual Studio installiert ist, wird die Erweiterung hinzugefügt. Sollte Visual Studio bei der Installation schon geöffnet sein, muss es neugestartet werden, damit der AC vollständig installiert wird.

Der AC wird gestartet, indem der Benutzer ein Dokument im Visual Studio Editor geöffnet hat (vorzugsweise das an dem gerade gearbeitet wird) und dann oben in der Hauptmenüleiste in dem Untermenü „Extras“ auf den Button der Erweiterung klickt.

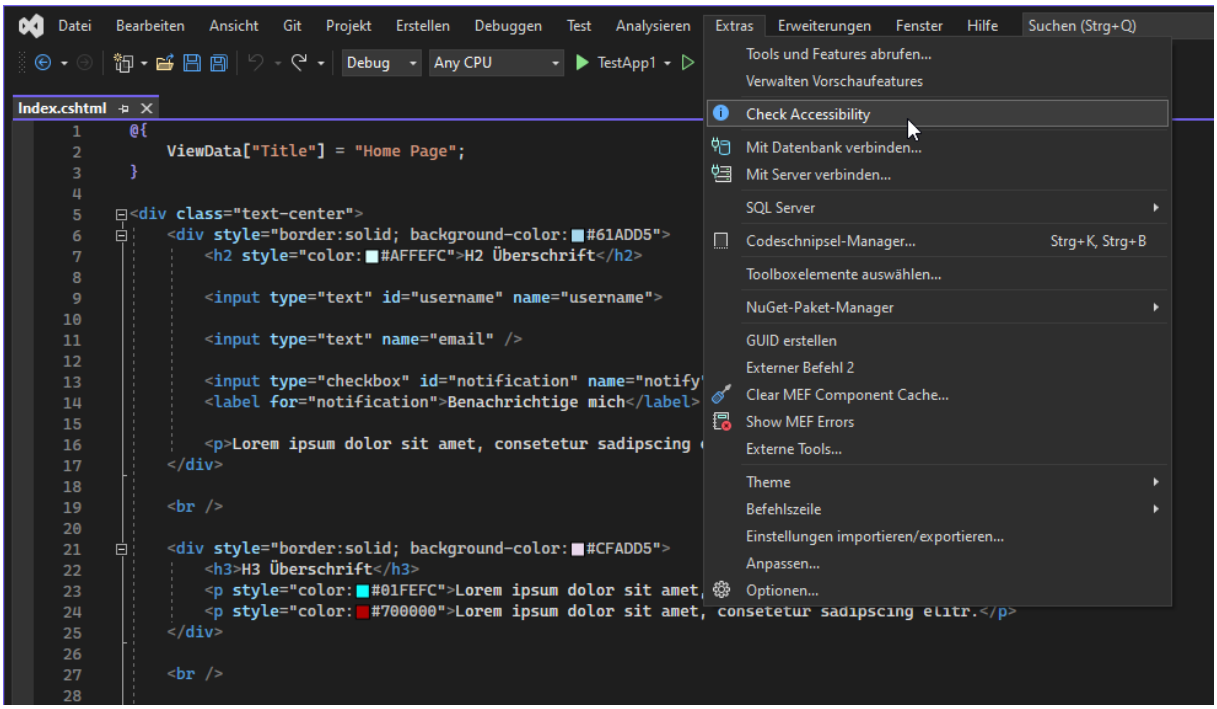


Abbildung 4: Visual Studio Oberfläche mit Menü, in dem sich der Button zur Ausführung der Erweiterung befindet

Vorher kann der Benutzer in den Visual Studio Einstellungen festlegen, welche der drei Barrierefreiheitskriterien geprüft werden sollen. Diese Optionen findet man in den allgemeinen Visual Studio Optionsmenü. Der Standardwert aller Optionen ist **`true`.**

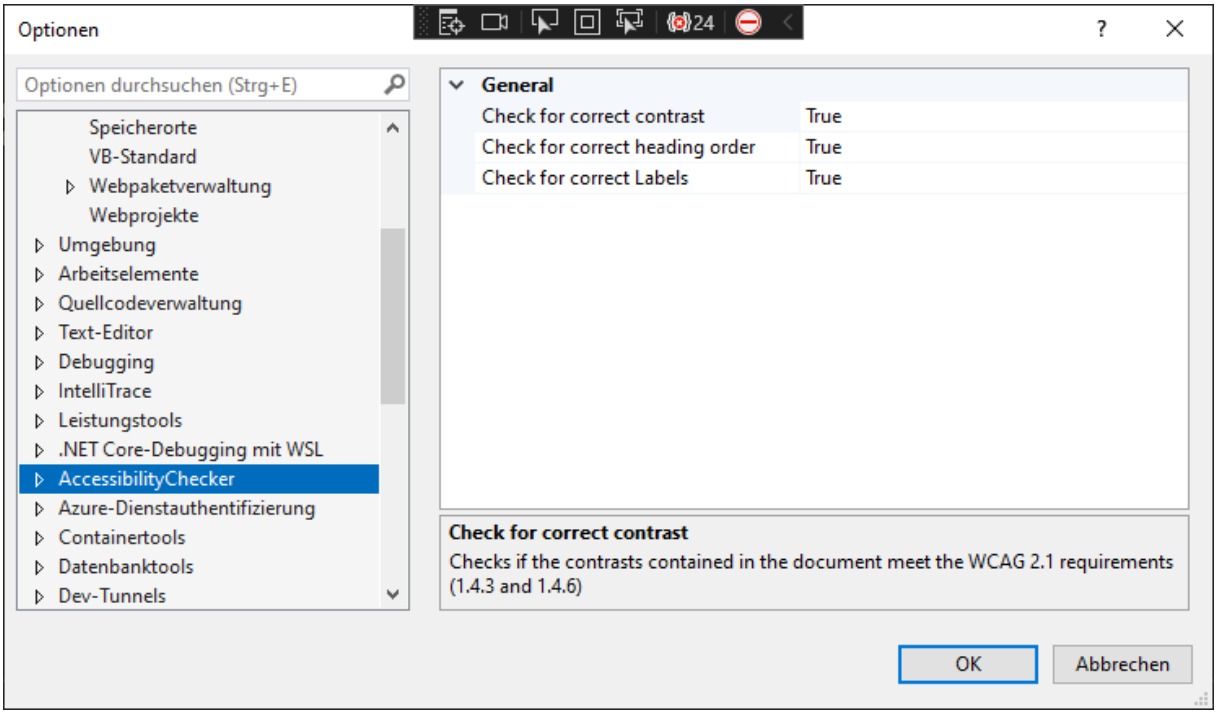


Abbildung 5: Optionen der Erweiterung

Sobald der Benutzer die Erweiterung startet, wird die besagte HTML-Seite überprüft. Sollten sich Mängel in der Barrierefreiheit der Seite ergeben, so werden diese in der Visual Studio Fehlerliste aufgelistet. Die Meldungen enthalten die Art der Mängel sowie die Codezeile, in der sie vorkommen. Dies ermöglicht ein effizientes Beheben der Mängel.

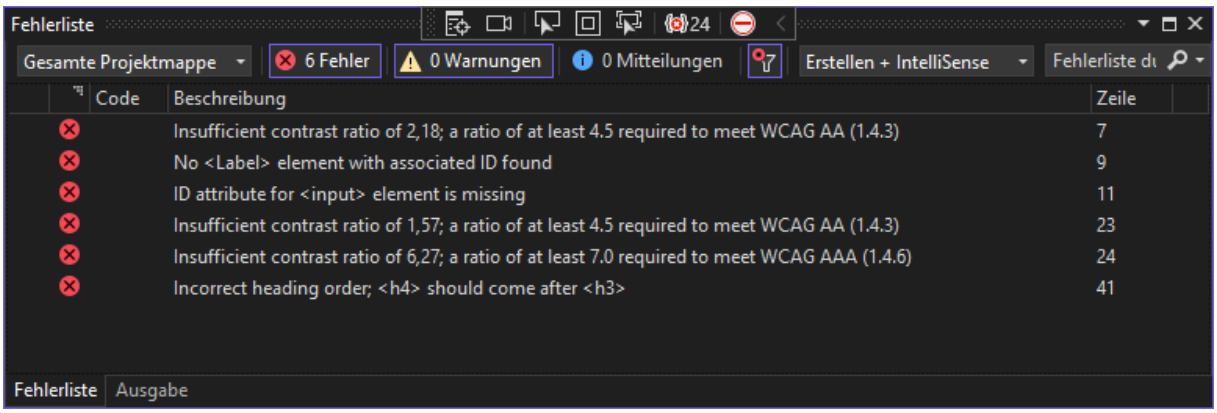


Abbildung 6: Visual Studio Fehlerliste

9 Entwicklung der Erweiterung

Microsoft stellt für die Entwicklung von Visual Studio Erweiterungen ein Software Development Kit (im folgenden VS SDK) bereit. [18] Zur Entwicklung der Erweiterung muss das VS SDK installiert werden. Dies ist über den Visual Studio Installer möglich, welcher eine Vielzahl von sogenannten „Workloads“ für die Entwicklung unterschiedlichster Anwendungen bereitstellt. Ein Workload ist ein Paket mit Werkzeugen, welche zu der Visual Studio Installation hinzugefügt werden.

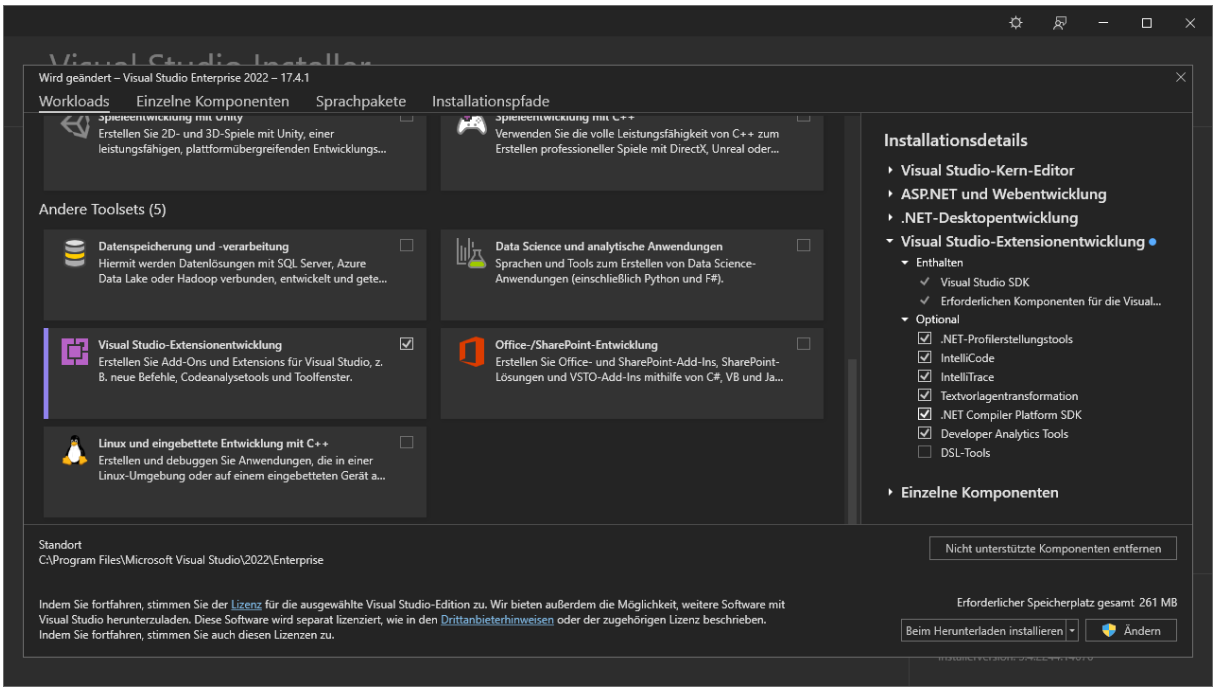


Abbildung 7: Visual Studio Installer mit verfügbaren Workloads

Nach Installation des VS SDK Workload findet man verschiedene Templates zur Erstellung von VSIX-Erweiterungen. Diese Templates ermöglichen einen schnelleren Einstieg in die Entwicklung, da die wichtigsten Dateien mit ihren Grundfunktionen und Kommentaren schon in dem Projekt integriert sind.

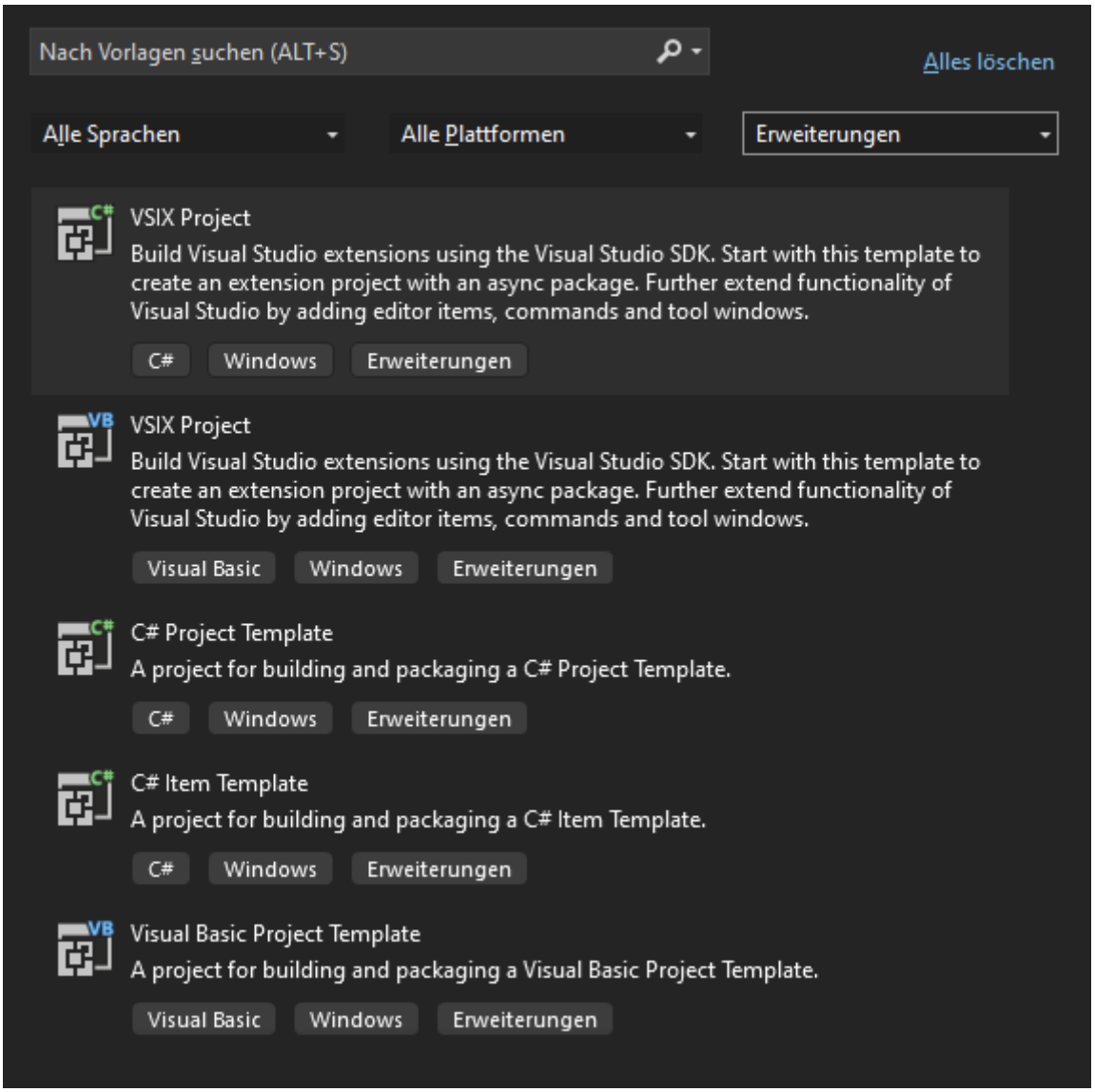


Abbildung 8: Visual Studio Projekt-Templates

9.1 Visual Studio SDK

Das VS SDK ermöglicht die Erweiterung der IDE um eine Vielzahl von Features, wie z.B. den Support für nicht unterstützte Programmiersprachen, inklusive Syntax Highlighting, Kompilierung und Debug Support. Die komplette Oberfläche der IDE kann verändert werden durch z.B. das Hinzufügen von Toolbars, Buttons, Untermenüs oder auch die komplette Erstellung eines neuen Editors.

Das Projekt zur Erstellung einer Erweiterung befindet sich in einem VSIX-Paket (Visual Studio Extension Paket) oder einer VSIX-Datei. Zusammen mit Metadaten wird diese verwendet um Erweiterungen als solche für Visual Studio zu klassifizieren.

Ist das Projekt angelegt können verschiedene Komponenten, welche Visual Studio erweitern, zum Projekt hinzugefügt werden. Folgende Items stehen zur Verfügung: [19]

- Async Package

- Command
- Async Tool Window
- Tool Window
- Extension Pack
- Editor Classifier
- Editor Margin
- Editor Text Adornment
- Editor Viewport Adornment
- Windows Worms Toolbox Control
- WPF Toolbox Control

Zum besseren Erlernen des VS SDK bietet Microsoft diverse YouTube Tutorials und ein Github Repository mit verschiedenen Beispielen von Visual Studio Erweiterungen. [20] Diese sind nützlich, um ein besseres Verständnis über die Struktur und Datenfluss der Projekte zu erlangen.

Es gibt verschiedene Hauptkomponenten, welche Bestandteil einer Erweiterung sind und dessen Funktionalität ausmachend.

9.1.1 VSCT-Datei

Hier werden Befehle deklariert. Es handelt sich um eine XML-Datei, welche die Definition von Button-Befehlen, Menüs, Tastaturabkürzungen, Verlinkungen zu Grafiken und mehr beinhaltet. Die Datei wird in eine `output.dll`-Datei kompiliert, welche von Visual Studio genutzt wird, um die gesamte interne Befehlsstruktur zu generieren. Die Datei dient einzig dazu, um Komponenten in der „Command Table“ zu deklarieren. Es werden von hier aus keine Befehle ausgeführt, sondern lediglich zu ihnen verlinkt.

Die Abkürzung steht für Visual Studio Command Table. Beim Erstellen eines neuen Projektes beinhaltet sie Kommentare, um die einzelnen Komponenten zu verstehen und das Entwickeln zu vereinfachen, ohne notwendigerweise in Microsofts Dokumentation des SDK nachschlagen zu müssen.

9.1.2 CS-Klassen der Komponenten

Hier werden in C# Methoden der verschiedenen Komponenten definiert. Sie beinhalten die gesamte Logik der Applikation und stellen somit den Kern des Erweiterungs-Projektes dar. Sie befinden sich üblicherweise in dem `Command` Ordner der Projektmappe.

9.1.3 Ressourcen

In dem Resources Ordner werden Ressourcen wie z.B. eigens erstellte Grafiken für Icons, Fenster etc. abgelegt damit sie einen festen Punkt im Projekt besitzen auf den zugegriffen werden kann.

9.1.4 Package.cs

Die Package Klasse ist der Einstiegspunkt für die meisten Komponenten. Hier werden Befehlshandler, Fenster, Optionen, Services und andere Komponenten initialisiert und registriert, damit Komponenten im Projekt auf andere Komponenten zugreifen können.

9.1.5 Manifest-Datei

Im VSIX-Manifest werden Metadaten wie der Autor, Versionsnummer, Beschreibung und die Pfade zu möglichen externen Ressourcen, wie z.B. Grafiken, festgelegt.

9.1.6 Optionen

Als Entwickler hat man die Möglichkeit die Visual Studio Erweiterung mit Optionen konfigurierbar zu machen. Die Optionen werden in einer Klasse, üblicherweise im `Options`-Ordner der Projektmappe, festgelegt. Sie können dann wie alle anderen Visual Studio Einstellungen über das Menü `Extras` aufgerufen werden. Vom Benutzer veränderte Optionen werden lokal gespeichert.

9.1.7 Kompilierung

Das Projekt wird in eine `vsix`-Datei kompiliert welche entweder in dem Pfad `/bin/debug` oder `/bin/release` generiert wird. Die hängt von der Build-Konfiguration ab.

9.1.8 Experimentelle Visual Studio Instanz

Um das problemlose Entwickeln und Debuggen einer Erweiterung zu ermöglichen, wird mit jedem Build des Projektes die Erweiterung in eine experimentelle Visual Studio Instanz eingebunden. Diese wird von dem Visual Studio SDK zur Verfügung gestellt und stellt eine sichere Umgebung zum Testen und Entwickeln dar, da Erweiterungen das Verhalten und Aussehen von Visual Studio drastisch verändern kann. Jedes Projekt, welches ein VSIX-Paket enthält, wird automatisch in der experimentellen Instanz gestartet.

9.2 Projektstruktur

Grundsätzlich besteht das Projekt aus den im vorherigen Kapitel beschriebenen Elementen.

Die Projektstruktur der AC-Erweiterung sieht folgendermaßen aus und jede Komponente wird erläutert.

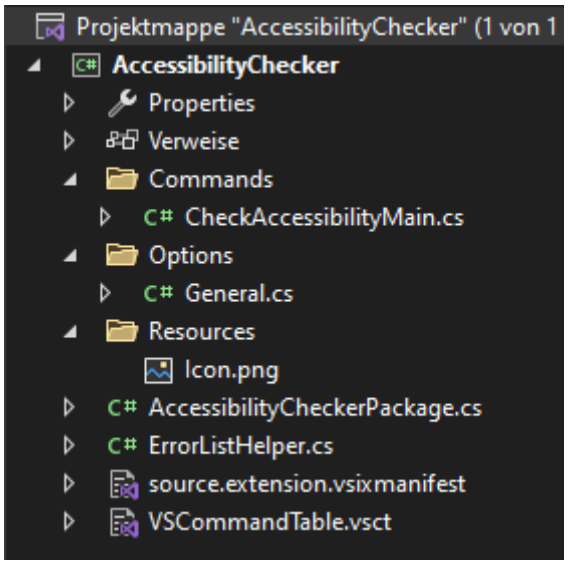


Abbildung 9: Projektstruktur des AC

9.2.1 Manifest

In dem Manifest befinden sich der Name, die Versionsnummer und die Beschreibung der Erweiterung. Die Datei befindet sich im XML-Format. Visual Studio interpretiert diese und erzeugt daraus eine Oberfläche welche verständlicher und leichter zu editieren ist als XML-Datei, welche im Textformat dargestellt wird. Es wurde auch ein Icon eingebunden, welches sich im `Resources` Ordner befindet, um die Erweiterung ansprechender zu gestalten.

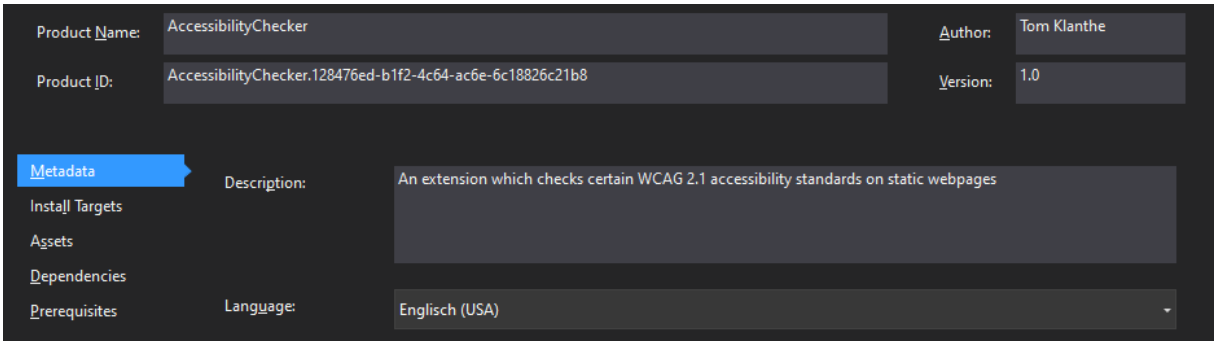


Abbildung 10: Oberfläche der Manifest-Datei

9.2.2 Visual Studio Command Table

In dem Visual Studio Command Table (`vsct`-Datei) wird im XML-Format ein Button zur Ausführung der Erweiterung und dessen Befehl definiert.

Die Datei ist in verschiedene Sektionen gegliedert:

- **Commands**: Hier werden Befehle, Menüs und menügruppen definiert. Diese Sektion benutzt eine Guid, um das Paket zu identifizieren, welches den Befehl enthält, der darin definiert ist.

Guid steht für „Globally Unique Identifier“ und wird in dem Projekt genutzt um einzigartig definierte Elemente, welche Teil von Visual Studios Ecosystems sind, anzusprechen.

- **Groups**: Hier werden Menügruppen definiert. Eine Menügruppe ist ein Container für andere Menüs oder Buttons (Befehle). Man kann sich eine Group als Teil eines Menüs sichtbar zwischen zwei Linien vorstellen. Das Parent einer Gruppe muss ein Menü sein.

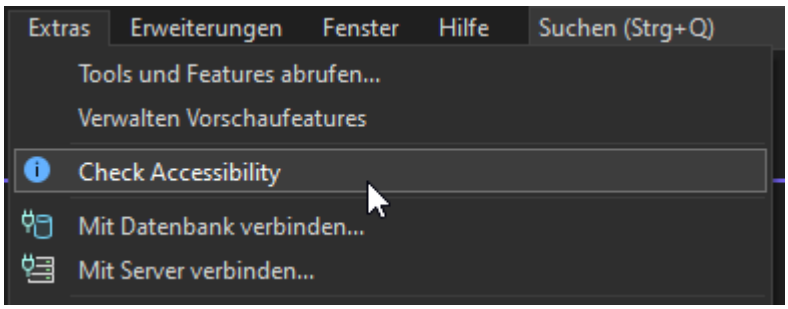


Abbildung 11: Menügruppe mit Button des AC

  Die Group Guid lautet in dem AC-Projekt `AccessibilityChecker` und die Parent Guid `VSMainMenu`, was der obersten Hauptmenüleiste von Visual Studio entspricht. Die ID des Parents ist `Tools`, welches den Button in dem `Extras` Menü platzieren wird.
- **Buttons**: Diese Sektion definiert Elemente, mit denen der Benutzer interagieren kann, wie z.B. Menübefehle oder Buttons.

  Um eine Menügruppe zu definieren, muss dessen ID, die Parent-Menügruppe und die Display Priorität spezifiziert werden. Die ID ist in dem Projekt `CheckAccessibilityMain`, welche die CS-Datei definiert, in der die mit dem Button verknüpfte Logik definiert ist.

  Sollten sich weitere Oberflächenelemente in einem Projekt und im gleichen Parent befinden, so kann mit dem `priority` Attribut die Reihenfolge der Elemente in der Oberfläche festgelegt werden.

  Mit `CommandFlag` hat man die Möglichkeit Parent Elemente weitere Eigenschaften zu geben. In dem Fall wird die Flag `IconIsMoniker` benutzt um das Icon, welches für den Button benutzt wird, als Visual Studio internes Icon zu deklarieren. Dies ermöglicht die Verwendung des `Icon` Elements mit einer Guid und einer ID, welche auf intern vorhandene Icons zugreift. In dem Fall ist dies das Statusinformationssymbol.

  In der `Strings` Sektion wird der für den Benutzern sichtbare Name festgelegt.
- **Symbols**: Hier werden die Menübefehle anhand ihrer Guid gruppiert.

```
<Commands package="AccessibilityChecker">
  <Groups>
    <Group guid="AccessibilityChecker" id="MyMenuGroup" priority="0x0600">
      <Parent guid="VSMainMenu" id="Tools"/>
    </Group>
  </Groups>

  <!--This section defines the elements the user can interact with, like a menu command or a button
      or combo box in a toolbar. -->
  <Buttons>
    <Button guid="AccessibilityChecker" id="MyCommand" priority="0x0100" type="Button">
      <Parent guid="AccessibilityChecker" id="MyMenuGroup" />
      <Icon guid="ImageCatalogGuid" id="StatusInformation" />
      <CommandFlag>IconIsMoniker</CommandFlag>
      <Strings>
        <ButtonText>Check Accessibility</ButtonText>
        <LocCanonicalName>.AccessibilityChecker.MyCommand</LocCanonicalName>
      </Strings>
    </Button>
  </Buttons>
</Commands>

<Symbols>
  <GuidSymbol name="AccessibilityChecker" value="{5ed1040e-48a1-417f-a731-23e274d64404}">
    <IDSymbol name="MyMenuGroup" value="0x0001" />
    <IDSymbol name="MyCommand" value="0x0100" />
  </GuidSymbol>
</Symbols>
```

Abbildung 12: Vollständiger Code der Visual Studio Command Table

9.2.3 Package.cs

In der `AccessibilityCheckerPackage.cs` werden unter anderem die Services für die Optionen und den ErrorListHelper initialisiert.

```
8   namespace AccessibilityChecker
9   {
10      [PackageRegistration(UseManagedResourcesOnly = true, AllowsBackgroundLoading = true)]
11      [InstalledProductRegistration(Vsix.Name, Vsix.Description, Vsix.Version)]
12      [ProvideMenuResource("Menus.ctmenu", 1)]
13      [Guid(PackageGuids.AccessibilityCheckerString)]
14      [ProvideOptionPage(typeof(OptionsProvider.GeneralOptions), "AccessibilityChecker", "General", 0, 0, true, SupportsProfiles = true)]
        0 Verweise
15      public sealed class AccessibilityCheckerPackage : ToolkitPackage
16      {
            0 Verweise
17          protected override async Task InitializeAsync(CancellationToken cancellationToken, IProgress<ServiceProgressData> progress)
18          {
19              await this.RegisterCommandsAsync();
20              ErrorListHelper.Initialize(this);
21          }
22      }
23  }
```

Abbildung 13: Codeausschnitt aus der Package.cs

9.2.4 Fehlerliste

Die Logik für die Ausgabe von eventuellen Barrierefreiheitsfehlern wird in eine Klasse `ErrorListHelper` ausgelagert, um das Projekt übersichtlich zu halten.

Durch die Initialisierung in der `AccessibilityCheckerPackage.cs` Datei wird ein `ErrorListProvider` Objekt erzeugt. Durch die Übergabe eines kontextabhängigen `ServiceProvider` Interfaces an die `Initialize()`-Methode kann nun die interne Fehlerliste von Visual Studio von der Erweiterung benutzt werden.

Fehler werden generiert in dem die `ErrorListHelper.Write()`-Methode in einem Fehlerfall aufgerufen wird. Der Methode werden folgende Werte übergeben:

- `TaskCategory`: Die Überkategorie des Fehlers.
- `TaskErrorCategory`: Die Kategorie des Fehlers. Es gibt insgesamt 3 Kategorien (Error, Message und Warning). Je nach Kategorie wird die Nachricht in der Fehlerliste anders dargestellt. Fehler in der AC-Erweiterung werden immer als Kategorie „Error“ ausgegeben.
- `Text`: Eine Beschreibung der Fehlermeldung.
- `Line`: Die Codezeile in der gegen eine Barrierefreiheitsregel verstoßen wird.

Die Methode erstellt dann ein `ErrorTask`-Objekt, welches dem `ErrorListProvider`-Objekt via einer `Add()`-Methode übergeben wird, welche dann die Fehlermeldung anzeigt.

```
 9    internal class ErrorListHelper : IServiceProvider
10    {
          0 Verweise
11        public object GetService(Type serviceType)
12        {
13            return Package.GetGlobalService(serviceType);
14        }
15
16        public static ErrorListProvider errorListProvider;
17
          1 Verweis
18        public static void Initialize(IServiceProvider serviceProvider)
19        {
20            errorListProvider = new ErrorListProvider(serviceProvider);
21        }
22
          5 Verweise
23        public static void Write(
24            TaskCategory category,
25            TaskErrorCategory errorCategory,
26            string text,
27            int line
28            )
29        {
30            ErrorTask task = new ErrorTask();
31            task.Text = text;
32            task.ErrorCategory = errorCategory;
33            task.Line = line;
34            task.Category = category;
35
36            errorListProvider.Tasks.Add(task);
37        }
38    }
```

Abbildung 14: Code der ErrorListHelper-Klasse

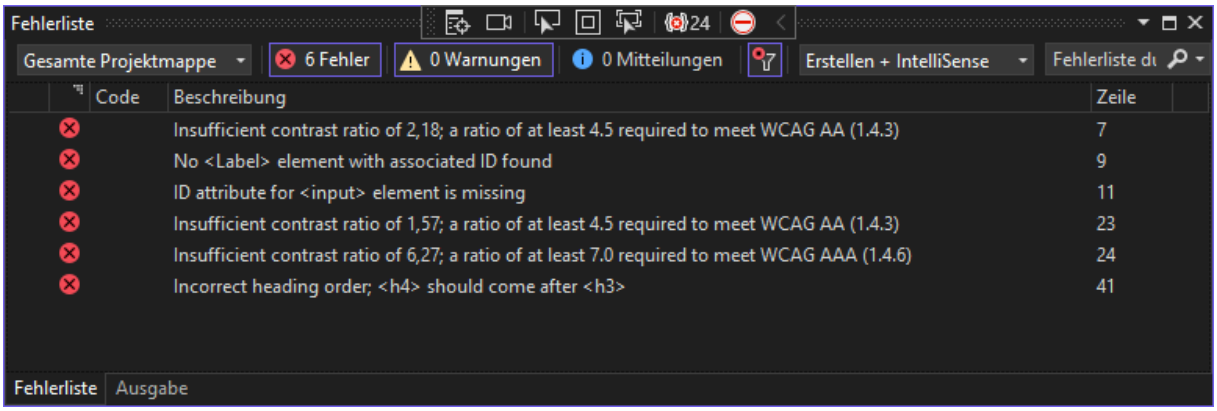
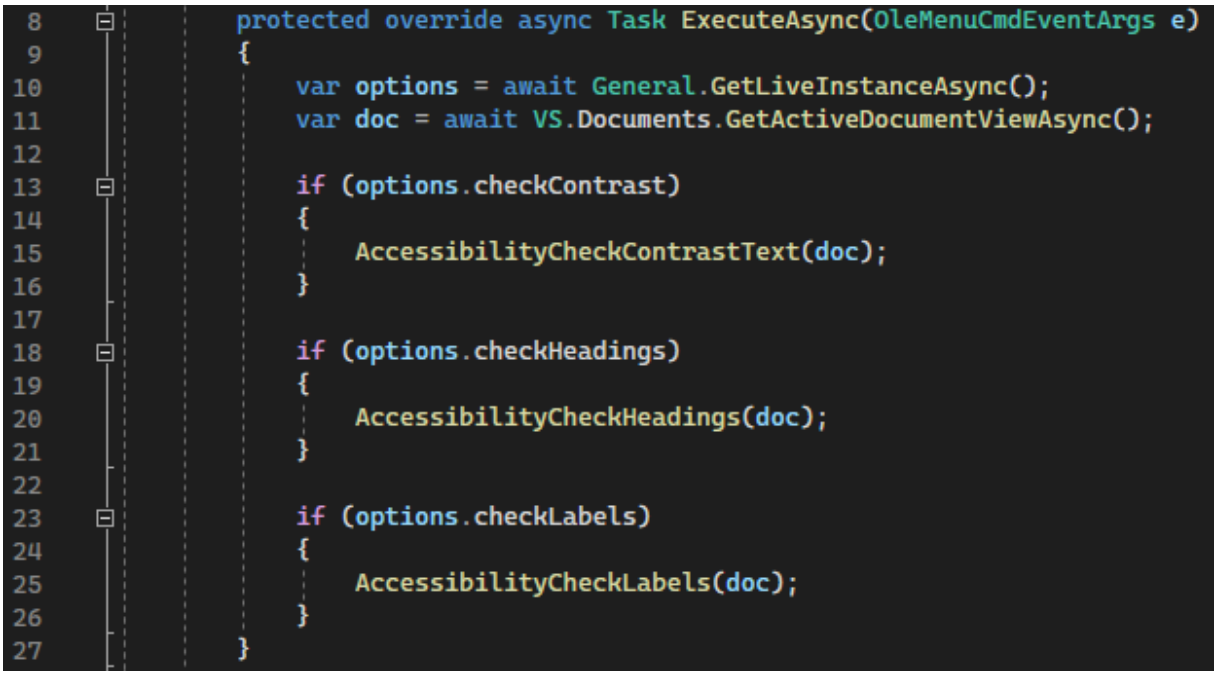


Abbildung 15: Visual Studios Fehlerliste in der Fehler angezeigt werden

9.3 Hauptlogik zur Überprüfung der Barrierefreiheit

Die Logik zur Überprüfung der Barrierefreiheit auf statischen HTML-Seiten befindet sich in der `CheckAccessibilityMain`-Klasse. Die Hauptmethode ist `ExecuteAsync()`, welche nach dem Klick auf den für die Erweiterung hinzugefügten Button ausgeführt wird. In dieser Methode werden zuerst die Optionen instanziiert, damit je nach gewählter Option die jeweiligen Barrierefreiheitskomponenten geprüft werden können. Je nach eingestellter Option werden dann die jeweiligen Methoden aufgerufen:

- `AccessibilityCheckContrastText()`: Die Überprüfung der Kontrastverhältnisse.
- `AccessibilityCheckHeadings()`: Die Überprüfung der korrekten Reihenfolge von Überschriften (`<h*>`-Elemente).
- `AccessibilityCheckLabels()`: Die Überprüfung von Eingabefeldern und deren dazugehörige Label.



```
protected override async Task ExecuteAsync(OleMenuCmdEventArgs e)
{
    var options = await General.GetLiveInstanceAsync();
    var doc = await VS.Documents.GetActiveDocumentViewAsync();

    if (options.checkContrast)
    {
        AccessibilityCheckContrastText(doc);
    }

    if (options.checkHeadings)
    {
        AccessibilityCheckHeadings(doc);
    }

    if (options.checkLabels)
    {
        AccessibilityCheckLabels(doc);
    }
}
```

Abbildung 16: ExecuteAsync-Methode mit den Überprüfungsmethoden

Jeder dieser Methoden wird das momentan geöffnete Dokument als Parameter übergeben. Das VS SDK bietet dazu die `GetActiveDocumentViewAsync()`-Methode, mit der der Inhalt des Editors als Datentyp `DocumentView` abgerufen werden kann.

Jede der Methoden zur Überprüfung der Barrierefreiheit basiert darauf, dass über jede Zeile des momentan geöffneten Dokuments iteriert wird. Um die Zeilen zu überprüfen, wird in jeder Iteration eine Zeile des Dokuments gelesen und einer Variablen des Datentyps String zugewiesen, um somit die Zeile besser analysieren zu können.

Diese Umwandlung geschieht in dem auf den `TextBuffer` des Editors zugegriffen wird und daraus die jeweilige Zeile mit `GetLineFromNumber()` ausgelesen und mit `GetText()` in einen String umgewandelt wird.

Der `TextBuffer` enthält die Daten, die sich in dem Moment innerhalb des Editors befinden.

9.3.1 Voraussetzungen

Damit die HTML-Seite korrekt von dem AC überprüft werden kann gelten bestimmte Voraussetzungen für die Gestaltung der HTML-Seite, sowie die Attributverteilung.

1. Für die Kontrastüberprüfung müssen die zu prüfenden Elemente via inline-CSS gestyled worden sein. Dies Bedeutet das die zu prüfenden Elemente in dem HTML ein `style`-Attribut besitzen müssen. Elemente, welche über CSS gestylt wurden, werden nicht von dem AC überprüft.
   Für die Darstellung von Text werden Paragrafen `<p>` und Überschriften `<h*>` überprüft.

Für die Hintergrundfarbe werden darüberliegende Containerelemente `<div>` überprüft.

2. Für die Überprüfung der Reihenfolge von Überschriften werden Überschriften geprüft, welche sich in demselben Parent-Element befinden, z.B. ein `<div>`-Element.
3. Für die Überprüfung von Eingabefeldern und den dazugehörigen Labels müssen sich beide Elemente demselben Parent-Element befinden, z.B. ein `<div>`-Element.

9.3.2 Kontrastüberprüfung

Bei jeder Iteration, über die sich im Editor befindenden HTML-Zeilen, wird zunächst geprüft, ob sich in der Zeile ein Containerelement `<div>` befindet. Existiert das Element, wird überprüft, ob das Element das Style-Attribut `background-color` besitzt.

Ist dies der Fall so wird die vorher definierte Hilfsvariable vom Datentyp Boolean `isInDiv` auf `true` gesetzt. Dies ist für die Überprüfung der Verschachtelung notwendig.

Außerdem wird eine Variable vom Datentyp String mit der `GetElementBgColorHexAttribute()` Methode zugewiesen. Diese Methode bekommt die derzeit zu analysierende Codezeile als Parameter und liefert den HEX-Wert der Hintergrundfarbe als String zurück. Dies geschieht durch die Überprüfung der Codezeile mit einem regulären Ausdruck, welche nach allen Zeichen zwischen der Zeichenfolge `background-color:#` und dem Attribut-schließenden Anführungszeichen ausliest.

```
207     private static string GetElementBgColorHexAttribute(string input)
208     {
209         String bgColorHex;
210         String bgColorRegex = "(?<=background-color:#).*(?=\"|;)";
211         bgColorHex = Regex.Match(input, bgColorRegex).Value;
212         return bgColorHex;
213     }
```

Abbildung 17: Code zur Bestimmung der Farbwerte für Hintergrundelemente

Jetzt muss noch die Farbe des Textes ermittelt werden. Hierzu wird so lange der Boolean `isInDiv` auf `true` gesetzt ist, über die verbleibenden Zeilen iteriert. Befindet sich in einer dieser Zeilen ein `<p>` oder ein `<h*>` Element, so werden diese auf das Vorkommen eines `color`-Attributs, welches zum Einfärben von Text benutzt wird, geprüft. Ist ein `color`-Attribut in Benutzung, wird wieder die `GetElementBgColorHexAttribute()`-Methode aufgerufen, um den HEX-Wert der Schriftfarbe zu ermitteln. Sollten Textelemente keine Styling-Attribute besitzen, so wird die ermittelte Hintergrundfarbe mit Schwarz (HEX: #000000) verglichen, um somit das Kontrastverhältnis zu ermitteln.

Da nun die beiden Farbwerte für den Vorder- und Hintergrund ermittelt wurden kann jetzt der Kontrast mit der `CheckContrast()` –Methode berechnet werden.

```
private static void AccessibilityCheckContrastText(DocumentView document)
{
    DocumentView doc = document;
    int lineCountInDoc = doc.TextBuffer.CurrentSnapshot.LineCount;
    bool isInDiv;
    string hexValDiv;
    string hexValChild;

    for (int i = 0; i < lineCountInDoc; i++)
    {
        String lineText = doc.TextBuffer.CurrentSnapshot.GetLineFromLineNumber(i).GetText();
        String innerLineText;

        if (lineText.Contains("<div"))
        {
            if (lineText.Contains("background-color"))
            {
                isInDiv = true;
                hexValDiv = GetElementBgColorHexAttribute(lineText);

                while (isInDiv)
                {
                    for (int j = i; j < lineCountInDoc; j++)
                    {
                        innerLineText = doc.TextBuffer.CurrentSnapshot.GetLineFromLineNumber(j).GetText();
                        if (innerLineText.Contains("<p") || innerLineText.Contains("<h"))
                        {
                            if (innerLineText.Contains("color") && isInDiv)
                            {
                                hexValChild = GetTextElementBgColorHexAttribute(innerLineText);
                                CheckContrast(hexValDiv, hexValChild, j);
                            }
                            else
                            {
                                CheckContrast(hexValDiv, "000000", j);
                            }
                        }
                        if (innerLineText.Contains("</div>"))
                        {
                            isInDiv = false;
                            break;
                        }
                    }
                }
            }
        }
    }
}
```

Abbildung 18: Code zur Analyse der Kontrastverhältnisse

Diese bekommt als Parameter die HEX-Werte der zu vergleichenden Farben und die Zeilennummer übergeben. Die Zeilennummer muss übergeben werden, da direkt in der `CheckContrast()`-Methode eine Fehlermeldung zur Ausgabe erzeugt wird. Diese benötigt die Zeilennummer als Parameter, um diese dann in der Fehlerliste mit anzuzeigen.

Die Methode berechnet die Kontraste nach WCAG-Regularien (siehe Abschnitt Kontrastverhältnisse). Zuerst werden für die jeweiligen R, G und B-Werte Variablen vom Typ double erstellt, damit in den folgenden Berechnungen mit Fließkommazahlen gearbeitet werden kann. Da die Methode die HEX-Werte als String übergeben bekommt müssen diese noch in ihre jeweiligen sRGB-Werte umgewandelt werden. Dies geschieht in dem jeweils zwei der sechs Zeichen eines HEX-Wertes via der bereits in C# implementierten `HexNumber()`-Methode in den Datentyp Integer umgewandelt werden. Bei der Deklarierung der Variablen werden die 8bit RGB-Werte schon in sRGB-Werte durch das Dividieren durch 255 umgewandelt.

Die Berechnung der relativen Leuchtdichte wurde zum Zweck der Übersichtlichkeit und Modularität in die Methode `RelativeLuminanceCalculation()` ausgelagert. Diese bekommt die sRGB-Werte als Double übergeben und berechnet dessen Leuchtdichte nach WCAG-Formel.

```
201         private static double RelativeLuminanceCalculation(double val)
202         {
203             val = (val <= 0.03928D) ? (val / 12.92D) : (Math.Pow(((val + 0.055D) / 1.055D), 2.4D));
204             return val;
205         }
206
```

Abbildung 19: Code zur Berechnung der relativen Leuchtdichte

Die finale Kontrastberechnung findet dann wieder in der `CheckContrast()`–Methode nach WCAG-Formel statt (siehe Zeile 186 und 187 in Abbildung 20: Code zur Kontrastberechnung nach WCAG-Formel).

Dann werden die Werte verglichen und bei zu geringem Kontrastverhältnissen eine Fehlermeldung über die Fehlerliste mit dazugehöriger Zeilennummer ausgegeben. Hier wird noch nach den WCAG-Richtlinien 1.4.3 und 1.4.6 unterschieden: Ist der Kontrast kleiner als 4,5, dann wird in der Fehlermeldung auf WCAG 1.4.3 hingewiesen. Ist der Kontrast größer als 4,5, jedoch kleiner als 7, dann wird in der Fehlermeldung auf WCAG 1.4.6 hingewiesen.

```
166         private static void CheckContrast(string hexValDiv, string hexValChild, int lineNumber)
167         {
168             double R1 = (int.Parse(hexValDiv.Substring(0, 2), System.Globalization.NumberStyles.HexNumber)) / 255D;
169             double G1 = (int.Parse(hexValDiv.Substring(2, 2), System.Globalization.NumberStyles.HexNumber)) / 255D;
170             double B1 = (int.Parse(hexValDiv.Substring(4, 2), System.Globalization.NumberStyles.HexNumber)) / 255D;
171
172             double R2 = (int.Parse(hexValChild.Substring(0, 2), System.Globalization.NumberStyles.HexNumber)) / 255D;
173             double G2 = (int.Parse(hexValChild.Substring(2, 2), System.Globalization.NumberStyles.HexNumber)) / 255D;
174             double B2 = (int.Parse(hexValChild.Substring(4, 2), System.Globalization.NumberStyles.HexNumber)) / 255D;
175
176             R1 = RelativeLuminanceCalculation(R1);
177             G1 = RelativeLuminanceCalculation(G1);
178             B1 = RelativeLuminanceCalculation(B1);
179             R2 = RelativeLuminanceCalculation(R2);
180             G2 = RelativeLuminanceCalculation(G2);
181             B2 = RelativeLuminanceCalculation(B2);
182
183             double L1 = 0.2126D * R1 + 0.7152D * G1 + 0.0722D * B1;
184             double L2 = 0.2126D * R2 + 0.7152D * G2 + 0.0722D * B2;
185
186             double contrast = L1 > L2 ? ((L1 + 0.05D) / (L2 + 0.05D)) : ((L2 + 0.05D) / (L1 + 0.05D));
187             contrast = Math.Round(contrast, 2);
188
189             if (contrast < 4.5D)
190             {
191                 ErrorListHelper.Write(0, TaskErrorCategory.Error, "Insufficient contrast ratio of " + contrast.ToString() +
192                     "; a ratio of at least 4.5 required to meet WCAG AA (1.4.3)", lineNumber);
193             }
194             if (contrast < 7D && contrast > 4.5D)
195             {
196                 ErrorListHelper.Write(0, TaskErrorCategory.Error, "Insufficient contrast ratio of " + contrast.ToString() +
197                     "; a ratio of at least 7.0 required to meet WCAG AAA (1.4.6)", lineNumber);
198             }
199         }
```

Abbildung 20: Code zur Kontrastberechnung nach WCAG-Formel

9.3.3 Überprüfung von Überschriften

Die Überprüfung der korrekten Reihenfolge von HTML-Überschriften erfolgt in der `CheckAccessibilityHeadings()`–Methode. Wie auch bei der Überprüfung der Kontrastverhältnisse wird hier über jede Zeile iteriert. Sollte sie ein öffnendes Containerelement, wie

 oder <section> enthalten, wird wieder eine zuvor initialisierte Hilfsvariable isInParent vom Datentyp Boolean auf true gesetzt. In dem Containerelement wird dann wieder mit einem regulären Ausdruck geprüft, ob sich in einer Zeile ein <h*> Element befindet. Es werden die beiden Hilfsvariablen currentHeading und comparingHeading vom Datentyp Integer genutzt um zwei verschiedene Überschriftenelemente zu vergleichen. Anfangs werden beide Variablen mit 0 initialisiert. Sobald ein <h*>-Element gefunden wurde wird mit einem regulären Ausdruck der Wert der Wichtigkeit einer Überschrift ermittelt (Werte von 1 bis 6) und dann currentHeading zugewiesen. Sollten sich danach weitere <h*>-Elemente befinden werden auch deren Werte ausgelesen, comparingHeading zugewiesen, und mit currentHeading verglichen. Sollte nach einer Überschrift also eine Überschrift mit niedrigerem Wert als der der Vorherigen vorkommen, wird eine Fehlermeldung ausgegeben.

Sollte bei der Iteration ein schließendes Containerelement vorkommen, werden currentHeading und comparingHeading wieder auf den Wert 0 zurückgesetzt und isInParent auf false gesetzt. Damit werden eventuelle Verschachtelungen eines HTML-Dokuments beachtet.

```
78      private static void AccessibilityCheckHeadings(DocumentView document)
79      {
80          DocumentView doc = document;
81          int lineCountInDoc = doc.TextBuffer.CurrentSnapshot.LineCount;
82          bool isInParent;
83
84          for (int i = 0; i < lineCountInDoc; i++)
85          {
86              String lineText = doc.TextBuffer.CurrentSnapshot.GetLineFromLineNumber(i).GetText();
87              int currentHeading;
88              int comparingHeading = 0;
89              String innerLineText;
90
91              if (lineText.Contains("<div") || lineText.Contains("<section"))
92              {
93                  isInParent = true;
94
95                  while (isInParent)
96                  {
97                      for (int j = i; j < lineCountInDoc; j++)
98                      {
99                          innerLineText = doc.TextBuffer.CurrentSnapshot.GetLineFromLineNumber(j).GetText();
100                         string reg = Regex.Match(innerLineText, "<h.>").Value;
101                         if (reg != "" && innerLineText.Contains(reg))
102                         {
103
104                             currentHeading = int.Parse(Regex.Match(innerLineText, "(?<=<h).(?=>)").Value);
105                             if (comparingHeading == 0)
106                             {
107                                 comparingHeading = currentHeading;
108                             }
109                             if (currentHeading < comparingHeading)
110                             {
111                                 ErrorListHelper.Write(0, TaskErrorCategory.Error, "Incorrect heading order; <h" + comparingHeading + "> " +
112                                     "should come after <h" + currentHeading + ">", j);
113                             }
114                         }
115                         if (innerLineText.Contains("</div>") || innerLineText.Contains("</section>"))
116                         {
117                             comparingHeading = 0;
118                             currentHeading = 0;
119                             isInParent = false;
120                             break;
121                         }
122                     }
123                 }
124             }
125         }
126     }
```

Abbildung 21: Code zur Überprüfung der Reihenfolge von Überschriften

9.3.4 Überprüfung von Input-Feldern

Bei der Iteration über die Zeilen wird nach dem Vorkommen von `<input>`-Elementen gesucht. Mit einem regulären Ausdruck wird dann zuerst überprüft, ob das Element ein ID-Attribut besitzt. Ist dies nicht der Fall wird eine Fehlermeldung ausgegeben. Jedes `<input>`-Element soll nämlich eine ID besitzen.

Besitzt das Element eine ID wird diese mit einem regulären Ausdruck ausgelesen und einer Variable `inputId` mit dem Datentyp String zugewiesen. Zudem wird die Variable `hasFoundAssociatedLabel` vom Datentyp Boolean mit dem Wert `false` erzeugt.

Es wird weiter über den Code iteriert und in jeder Zeile nach einem `<Label>`-Element mit der dazugehörigen ID gesucht. Dies geschieht auch mit einem regulären Ausdruck, welcher die `inputId` nutzt. Wurde bis zum Ende des HTMLs kein dazugehöriges `<Label>`-Element gefunden wird eine Fehlermeldung ausgegeben.

```
128     private static void AccessibilityCheckLabels(DocumentView document)
129     {
130         DocumentView doc = document;
131         int lineCountInDoc = doc.TextBuffer.CurrentSnapshot.LineCount;
132
133         for (int i = 0; i < lineCountInDoc; i++)
134         {
135             String lineText = doc.TextBuffer.CurrentSnapshot.GetLineFromLineNumber(i).GetText();
136             String innerLineText;
137
138             if (lineText.Contains("<input"))
139             {
140                 if (!Regex.IsMatch(lineText, "(?<=id=\")(.*?)(?=\")"))
141                 {
142                     ErrorListHelper.Write(0, TaskErrorCategory.Error, "ID attribute for <input> element is missing", i);
143                 }
144                 else
145                 {
146                     string inputId = Regex.Match(lineText, "(?<=id=\")(.*?)(?=\")").Value;
147                     bool hasFoundAssociatedLabel = false;
148
149                     for (int j = 0; j < lineCountInDoc; j++)
150                     {
151                         innerLineText = doc.TextBuffer.CurrentSnapshot.GetLineFromLineNumber(j).GetText();
152                         if (Regex.IsMatch(innerLineText, "(?<=<label for=\")" + inputId + "(?=\")"))
153                         {
154                             hasFoundAssociatedLabel = true;
155                         }
156                         if (!hasFoundAssociatedLabel && j == lineCountInDoc - 1)
157                         {
158                             ErrorListHelper.Write(0, TaskErrorCategory.Error, "No <Label> element with associated ID found", i);
159                         }
160                     }
161                 }
162             }
163         }
164     }
```

Abbildung 22: Code zur Überprüfung der Benennung von Input-Elementen

9.3.5 Optionen

Der Code für das Einstellen von Optionen befindet sich in der Klasse `General`, welche von der von Visual Studio bereitgestellten Klasse `BaseOptionsModel` erbt. Die Optionen werden in der `AccessibilityCheckerPackege.cs` initialisiert, damit sie in der Erweiterung benutzt werden können. Die drei verschiedenen Optionen für die Barrierefreiheitsüberprüfungen werden als öffentlich sichtbare Boolean-Datentypen deklariert.

```
12      public class General : BaseOptionModel<General>
13      {
14          [Category("General")]
15          [DisplayName("Check for correct contrast")]
16          [Description("Checks if the contrasts contained in the document meet the WCAG 2.1 requirements (1.4.3 and 1.4.6)")]
17          [DefaultValue(true)]
            1 Verweis
18          public bool checkContrast { get; set; }
19
20          [Category("General")]
21          [DisplayName("Check for correct heading order")]
22          [Description("Checks if the headings contained in the document are sorted according to the WCAG 2.1 requirements (1.3.1)")]
23          [DefaultValue(true)]
            1 Verweis
24          public bool checkHeadings { get; set; }
25
26          [Category("General")]
27          [DisplayName("Check for correct Labels")]
28          [Description("Checks if the <Label> elements contained in the document are named according to the WCAG 2.1 requirements (2.5.3)")]
29          [DefaultValue(true)]
            1 Verweis
30          public bool checkLabels { get; set; }
31      }
```

Abbildung 23: Code der drei möglichen Optionen

Jede Option hat vier Eigenschaften, die in der Klasse definiert werden:

1. `Category`: Die Kategorie die in dem Optionenfenster über den einzelnen Einstellungen sichtbar ist. Hier „General“, da es sich um die generellen Einstellungen handelt.
2. `DisplayName`: Der angezeigte Name der jeweiligen Einstellung.
3. `Description`: Eine detaillierte Beschreibung der Einstellung.
4. `DefaultValue`: Der Wert der ursprünglich gesetzt wird. Hier alle auf `true`, damit der Benutzer die Anwendung nutzen kann, ohne erst alle Optionen aktivieren zu müssen.

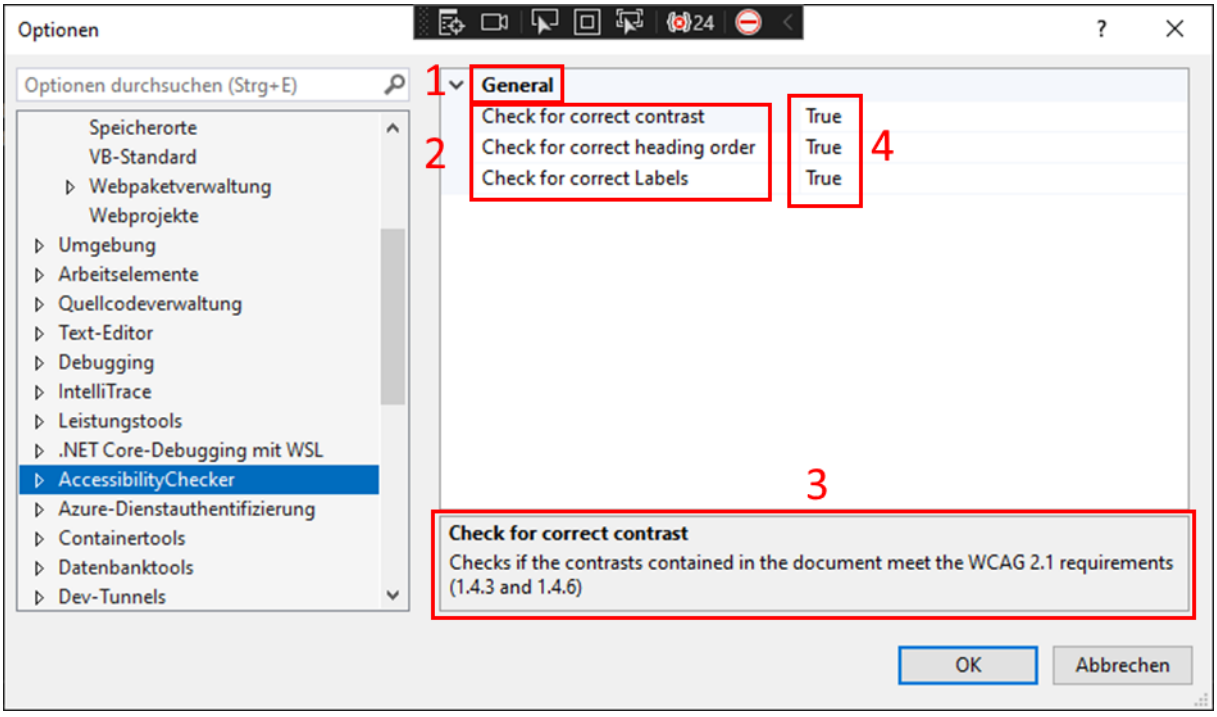


Abbildung 24: Oberfläche des Optionsmenüs

9.4 Visual Studio Experimental Instance

Zum Entwickeln von Visual Studio Erweiterungen bietet das VS SDK eine experimentelle Instanz von Visual Studio, in der Erweiterungen entwickelt, getestet und debugged werden können. Die experimentelle Instanz verhält sich wie eine normale Visual Studio Instanz, jedoch läuft sie getrennt von der Instanz, in der die Erweiterung entwickelt wird, um mögliche Nebenerscheinungen zu vermeiden.

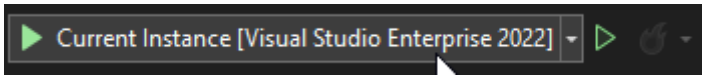


Abbildung 25: Button zum Starten der experimentellen Instanz

Die experimentelle Instanz wird wie jedes andere Visual Studio Projekt über den Ausführungsbutton gestartet.

Es gibt zusätzliche Werkzeuge am oberen Rand jedes Visual Studio Fensters der experimentellen Instanz mit denen man effizient entwickeln kann, z.B. eine Schnellauswahl von Elementen in der Visual Studio Oberfläche.

Abbildung 26: Entwicklungswerkzeuge in der experimentellen Instanz

9.5 Anwendung zum Testen des AC und Beispiele

Zum Entwickeln und Testen des AC wurde eine simple ASP.NET Core Anwendung erstellt. Darin befindet sich die `index.cshtml`-Datei, welche das zu testende HTML enthält. Das Projekt wird dann in der experimentellen Instanz geöffnet.

Im folgenden Beispiel wird eine HTML-Seite mit mehreren Abschnitten auf Barrierefreiheit überprüft.

Die Seite enthält vier Abschnitte:

1. Einen gefärbten `<div>`-Container mit einer gefärbten Überschrift, drei `<input>`-Elemente, von denen das 2. Element kein ID-Attribut, und das 1. Element kein zugehöriges `<Label>` besitzt. Als letztes gibt es noch ein `<p>`-Element, welches nicht eingefärbt ist.
2. Einen eingefärbten `<div>`-Container mit einer nicht eingefärbten Überschrift, sowie zwei unterschiedlich eingefärbten `<p>`-Elementen.
3. Ein `<section>`-Container mit zwei Überschriften und zwei `<p>`-Elementen. Die erste Überschrift ist ein `<h2>`-Element, die zweite ein `<h3>`-Element.
4. Ein `<section>`-Container mit zwei Überschriften und zwei `<p>`-Elementen. Die erste Überschrift ist ein `<h4>`-Element, die zweite ein `<h3>`-Element.

```
1   @{
2       ViewData["Title"] = "Home Page";
3   }
4
5   <div class="text-center">
6       <div style="border:solid; background-color:#61ADD5">
7           <h2 style="color:#AFFEFC">H2 Überschrift</h2>
8
9           <input type="text" id="username" name="username">
10
11          <input type="text" name="email" />
12
13          <input type="checkbox" id="notification" name="notify">
14          <label for="notification">Benachrichtige mich</label>
15
16          <p>Lorem ipsum dolor sit amet, consetetur sadipscing elitr.</p>
17      </div>
18
19      <br />
20
21      <div style="border:solid; background-color:#CFADD5">
22          <h3>H3 Überschrift</h3>
23          <p style="color:#01FEFC">Lorem ipsum dolor sit amet, consetetur sadipscing elitr.</p>
24          <p style="color:#700000">Lorem ipsum dolor sit amet, consetetur sadipscing elitr.</p>
25      </div>
26
27      <br />
28
29      <section style="border:solid">
30          <h2>H2 Überschrift</h2>
31          <p>Lorem ipsum dolor sit amet, consetetur sadipscing elitr.</p>
32          <h3>H3 Überschrift</h3>
33          <p>Lorem ipsum dolor sit amet, consetetur sadipscing elitr.</p>
34      </section>
35
36      <br />
37
38      <section style="border:solid">
39          <h4>H4 Überschrift</h4>
40          <p>Lorem ipsum dolor sit amet, consetetur sadipscing elitr.</p>
41          <h3>H3 Überschrift</h3>
42          <p>Lorem ipsum dolor sit amet, consetetur sadipscing elitr.</p>
43      </section>
44  </div>
```

Abbildung 27: Code der HTML-Testseite

Wenn der AC ausgeführt wird, erscheinen sechs Fehlermeldungen in der Fehlerliste:

1. Unzureichender Kontrast von 2,18 zwischen der Überschrift in Zeile 7 und dem darüber liegenden `<div>`-Container in Zeile 6.
2. Kein zugehöriges `<Label>`-Element zu dem `<input>` in Zeiler 9 gefunden.
3. Das `<input>`-Element in Zeile 11 besitzt keine ID. Somit kann auch kein `<Label>`-Element zugewiesen werden.
4. Unzureichender Kontrast von 1,57 zwischen dem `<p>`-Element in Zeile 23 und dem darüber liegenden `<div>`-Container in Zeile 21. Der Kontrast entspricht somit nicht der WCAG 1.4.3 Richtlinie.

5. Unzureichender Kontrast von 6,27 zwischen dem `<p>`-Element in Zeile 24 und dem darüber liegenden `<div>`-Container in Zeile 21. Der Kontrast entspricht somit nicht der WCAG 1.4.6 Richtlinie.
6. Inkorrekte Reihenfolge von Überschriften in dem letzten Abschnitt. In Zeile 41 befindet sich eine `<h3>`-Überschrift, obwohl sich in dem gleichen `<section>`-Container vorher eine `<h4>`-Überschrift befindet.

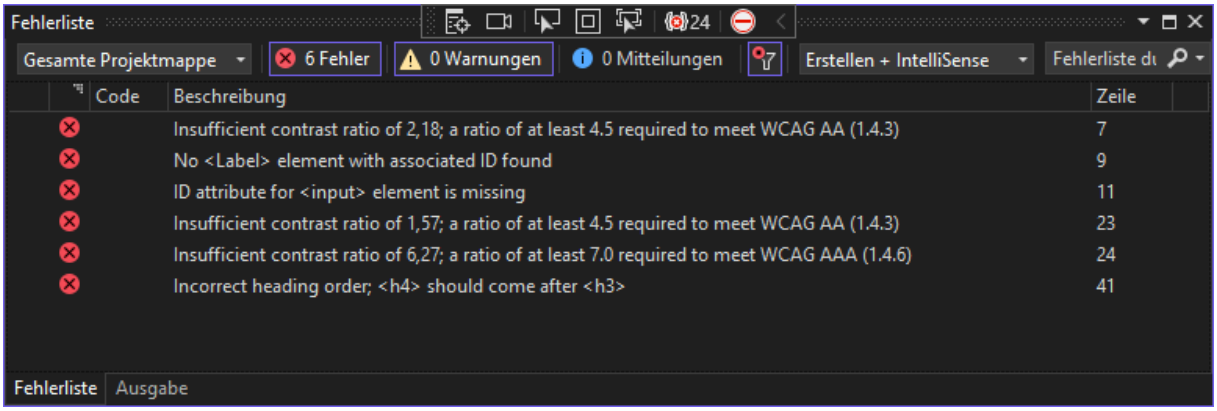


Abbildung 28: Fehlerliste mit ausgegebenen Fehlern, die auf der Testseite gefunden wurden

Bei Ausführung des Projektes wird folgende Seite in dem Standardbrowser angezeigt. Mit einem genauen Blick kann man die niedrigen Kontraste und die inkorrekte Reihenfolge zweier Überschriften erkennen, welche in der Fehlerliste konkret aufgezeigt werden.

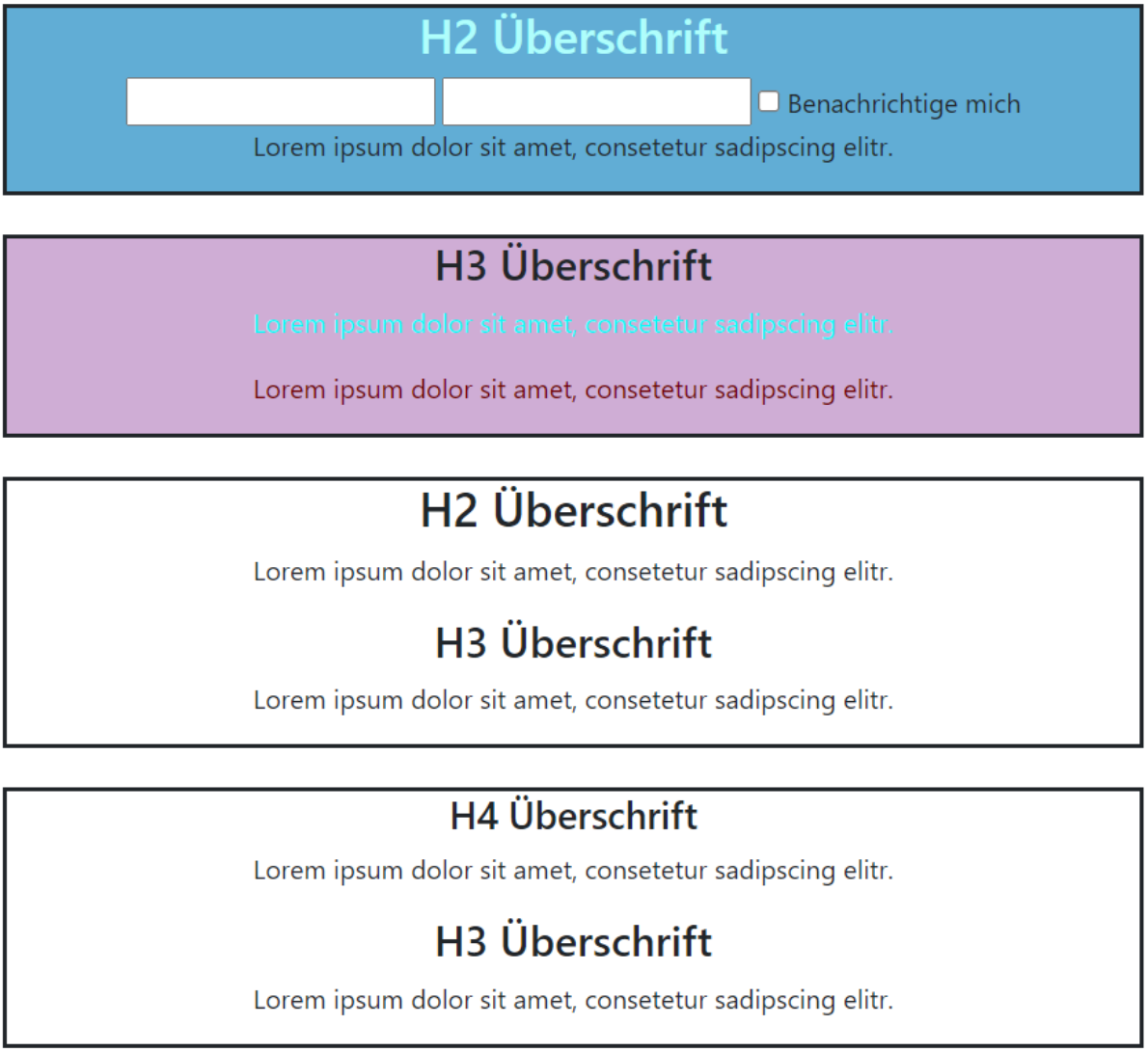


Abbildung 29: Im Browser gerenderte Ansicht der Testseite

Die Umsetzung der Erweiterung zeigt auf, dass das Überprüfen dieser drei Teilbereiche der Barrierefreiheit unter gewissen Voraussetzungen in Visual Studio technisch umsetzbar ist. Die Erweiterung analysiert das HTML durch Iteration über den Code und mit Hilfe von regulären Ausdrücken. Entwicklern werden so zuverlässig eventuelle Probleme punktuell während des Entwicklungsprozesses aufgezeigt. Somit können barrierefreie Webseiten effizienter entwickelt werden.

10 Auswertung

Die im Rahmen dieser Bachelorarbeit konzipierte und entwickelte Visual Studio Erweiterung ermöglicht es Entwicklern mögliche Fehlerquellen schon während des Entwicklungsprozesses ausfindig zu machen.

Die Erweiterung ist jedoch nur unter bestimmten Voraussetzungen möglich, welche in einer modernen produktiven Webentwicklungsumgebung selten gegeben sind. Die Erweiterung beschränkt sich auch nur auf statische Webseiten, welche nicht durch z.B. CSS und Javascript manipuliert werden.

Es werden zwar wichtige Regelungen der Barrierefreiheit überprüft, jedoch sind die Richtlinien so breit gefächert, dass diese Punkte nur einen kleinen Teilbereich testen. Die Entwicklung der Erweiterung zeigt aber auf, dass auch die Implementation von Überprüfungen der restlichen Richtlinien möglich ist, und das Potential besteht, die aufgewendete Zeit während des Entwicklungsprozesses erheblich zu reduzieren.

11 Zusammenfassung, Fazit, und Ausblick

Das Ziel dieser Bachelorarbeit war es die aktuelle Gesetzeslage und Richtlinien zur Gestaltung barrierefreier Webseiten zu analysieren und Teilbereiche davon in einer Anwendung umzusetzen.

Hierzu wurden die Richtlinien der WCAG 2.1, die BITV und der europäische Standard EN 301 549 zusammengefasst und passende Bereiche aufgeführt, sowie ein Konzept entwickelt, wie diese Teilbereiche umgesetzt werden können.

Als Anwendung wurde eine Visual Studio Erweiterung konzipiert und entwickelt, welche Webentwicklern unterstützen soll Webinhalte zu einem gewissen Maße barrierefrei zu gestalten. Da es einen enormen Arbeitsaufwand erfordert jede einzelne WCAG-Richtlinie umzusetzen, wurden nur drei Teilbereiche umgesetzt.

Es wurde untersucht wie Kontrastverhältnisse zwischen zwei Farben auf Basis moderner Anzeigetechnologien berechnet und für Entwickler zugänglich gemacht werden können. Auf Basis der von der WCAG vorgegebenen Formel wurde dann in der Erweiterung die HTML-Struktur analysiert und die Kontraste berechnet und dem Entwickler via Visual Studios interner Fehlerliste mitgeteilt, wenn bestimmte HTML-Elemente nicht WCAG-konform umgesetzt wurden und an welcher Stelle genau sie sich befinden.

Es wurde auch untersucht, ob sich Überschriftenelemente in der korrekten Reihenfolge befinden, um somit eine einfach Navigierbarkeit von Webinhalten zu gewährleisten. Sollten sich Überschriften in einer falschen Reihenfolge befinden wird dies dem Entwickler mitgeteilt.

Die Benutzung von Labels für Eingabefelder ist essenziell, um Benutzern Informationen über die Eingabefelder zu geben. Dies hat auch einen Nutzen für nicht-beeinträchtigte Personen, da die Informationen die Bedienung erleichtern. Bei fehlenden Labels werden die Entwickler darauf hingewiesen.

Die Umsetzung der Überprüfung dieser drei Teilbereiche in einer Visual Studio Erweiterung ist gelungen und kann unter gewissen Voraussetzungen produktiv genutzt werden. Jedoch funktioniert die Erweiterung in dem momentanen Zustand nur auf statischen Webseiten, welche die HTML-Attribute der Elemente, wie z.B. die Hintergrundfarbe, nur inline zugewiesen bekommen. Dies hat den Nachteil das die Modularität, welche bei der modernen Webentwicklung gängig ist, nicht mit dieser Erweiterung kompatibel ist. Auch basiert die Analyse des HTML momentan auf der Iteration über die einzelnen Codezeilen, und der Auswertung dieser über die Suche von spezifischen Zeichenfolgen unter der Anwendung von regulären Ausdrücken. Das Verständnis des Codes und die Erweiterbarkeit gestaltet sich somit schwieriger. Für die Weiterentwicklung wäre es also von Vorteil, die Logik so umzugestalten, dass HTML-Elemente direkt als solche in dem C#-Code der Erweiterung ausgewertet wird. Die Überprüfung weiterer WCAG-Richtlinien könnte implementiert werden, um die Komplexität von modernen Webanwendungen zu unterstützen und diese soweit es geht barrierefrei zu gestalten.

Die Erweiterung kann durch die Implementierung weiterer Überprüfungen gewisser Teilbereiche der Barrierefreiheit erweitert werden. Somit kann sich der Entwicklungsaufwand erheblich verringern, da Webinhalte nicht erst kompiliert und im Webbrowser gerendert werden muss, um sie dort mit externen Werkzeugen zu Testen. Die Kompilierung einer HTML-Seite und deren Ausführung in einem Webbrowser dauert mehrere Sekunden, je nach Entwicklungsumgebung bis zu einer Minute. Die Überprüfung des z.B. Kontrasts würde dann zusätzlich noch Zeit in Anspruch nehmen, da die verwendeten Farben erst in ein externes Werkzeug, wie z.B. den Colour Contrast Analyzer, eingetragen und ausgewertet werden müssen. Durch die Benutzung der Erweiterung werden die Ergebnisse direkt in der IDE angezeigt und somit erheblich Zeit gespart.

Auch eine eventuelle Auslagerung an externe Institutionen zum Testen auf BITV- bzw. WCAG-Konformität kann somit wegfallen.

12 Literaturverzeichnis

[1] Bundesministerium der Justiz, „Verordnung zur Schaffung barrierefreier Informationstechnik nach dem Behindertengleichstellungsgesetz (Barrierefreie-Informationstechnik-Verordnung - BITV 2.0),“ [Online]. Available: https://www.gesetze-im-internet.de/bitv_2_0/BJNR184300011.html. [Zugriff am 11 12 2022].

[2] BIK, „BIK BITV Test,“ [Online]. Available: https://www.bitvtest.de/bitv_test.html. [Zugriff am 11 12 2022].

[3] W3C, „Web Content Accessibility Guidelines (WCAG) 2.1,“ 05 06 2018. [Online]. Available: https://www.w3.org/TR/WCAG21/. [Zugriff am 11 12 2022].

[4] Bundesfachstelle Barrierefreiheit, „Neue Version der EN 301 549 gilt für Behörden,“ 23 09 2021. [Online]. Available: https://www.bundesfachstelle-barrierefreiheit.de/SharedDocs/Kurzmeldungen/DE/neue-version-der-en-301-549.html. [Zugriff am 11 12 2022].

[5] Google, „Google Lighthouse Overview,“ 27 09 2016. [Online]. Available: https://developer.chrome.com/docs/lighthouse/overview/. [Zugriff am 11 12 2022].

[6] Institute for Disability Research, Policy & Practice Utah State University, „WAVE web accessibility evaluation tool,“ 2022. [Online]. Available: https://wave.webaim.org. [Zugriff am 11 12 2022].

[7] BHVD Dresden, „NVDA - der freie ScreenReader für blinde Computernutzer,“ BHVD Dresen, 2022. [Online]. Available: https://nvda.bhvd.de. [Zugriff am 11 12 2022].

[8] P. Carbonnelle, „Top IDE index,“ 12 2022. [Online]. Available: https://pypl.github.io/IDE.html. [Zugriff am 11 12 2022].

[9] W3C, „The history of the Web,“ 28 01 2019. [Online]. Available: https://www.w3.org/wiki/The_history_of_the_Web. [Zugriff am 11 12 2022].

[10] J. Hellbusch, „Erfolgskriterien und Konformitätsbedingungen der Web Content Accessibility Guidelines (WCAG) 2.1,“ 2022. [Online]. Available: https://www.barrierefreies-webdesign.de/richtlinien/wcag-2.1/erfolgskriterien/. [Zugriff am 11 12 2022].

[11] Liquid Impressions KG, „Europäischer Standard EN 301 549,“ Liquid Impressions KG, 2022. [Online]. Available: https://www.barrierefreie-webseite.de/barrierefreies-webdesign/en-301-549-europaeischer-standard/einfuehrung. [Zugriff am 11 12 2022].

[12] W3C, „WCAG 2.1 ADOPTION IN EUROPE,“ 13 09 2018. [Online]. Available: https://www.w3.org/blog/2018/09/wcag-2-1-adoption-in-europe/. [Zugriff am 11 12 2022].

[13] „Barrierefreie Informationstechnik-Verordnung (BITV),“ 21 12 2011. [Online]. Available: http://www.die-barrierefreie-website.de/barrierefrei/barrierefreie-informationstechnik-verordnung.html. [Zugriff am 11 12 2022].

[14] W3C, „Understanding Success Criterion 1.4.6: Contrast (Enhanced),“ 2022. [Online]. Available: https://www.w3.org/WAI/WCAG21/Understanding/contrast-enhanced.html. [Zugriff am 11 12 2022].

[15] W3C, „G17: Ensuring that a contrast ratio of at least 7:1 exists between text (and images of text) and background behind the text,“ 2016. [Online]. Available: https://www.w3.org/TR/WCAG20-TECHS/G17.html#G17-procedure. [Zugriff am 11 12 2022].

[16] M. A. S. C. R. M. Michael Stokes, „A Standard Default Color Space for the Internet - sRGB,“ 05 11 1996. [Online]. Available: https://www.w3.org/Graphics/Color/sRGB.html. [Zugriff am 11 12 2022].

[17] TPGi, „Colour Contrast Analyser (CCA),“ 2022. [Online]. Available: https://www.tpgi.com/color-contrast-checker/. [Zugriff am 11 12 2022].

[18] Microsoft, „Visual Studio SDK,“ Microsoft, 27 09 2022. [Online]. Available: https://learn.microsoft.com/de-de/visualstudio/extensibility/visual-studio-sdk?view=vs-2022. [Zugriff am 11 12 2022].

[19] Microsoft, „Start developing extensions in Visual Studio,“ Microsoft, 09 03 2022. [Online]. Available: https://learn.microsoft.com/en-us/visualstudio/extensibility/starting-to-develop-visual-studio-extensions?source=recommendations&view=vs-2022. [Zugriff am 11 12 2022].

[20] Microsoft, „Visual Studio Extensibility Samples,“ Microsoft, 2022. [Online]. Available: https://github.com/microsoft/VSSDK-Extensibility-Samples. [Zugriff am 11 12 2022].

13 Abbildungsverzeichnis

14 Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit mit dem Titel:

„Entwicklung einer Visual Studio Erweiterung zur Anzeige von Webseitenelementen und Überprüfung der Barrierefreiheit im IDE-Editor“

selbständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Alle Passagen, die ich wörtlich aus der Literatur oder aus anderen Quellen wie z. B. Internetseiten übernommen habe, habe ich deutlich als Zitat mit Angabe der Quelle kenntlich gemacht.

Datum	Unterschrift