Browser-Plugin zur Darstellung von autocomplete-Attributen

Bachelorarbeit

im Studiengang
Medieninformatik

vorgelegt von

Philipp Recke
Matr.-Nr. 37698

am 9. Oktober 2023
an der Hochschule der Medien Stuttgart

Erstprüfer: Prof. Dr. Gottfried Zimmermann
Zweitprüfer: Pirmin Gersbacher, M. Sc.

Kompetenzzentrum Digitale Barrierefreiheit

Ehrenwörtliche Erklärung

Hiermit versichere ich, Philipp Recke, ehrenwörtlich, dass ich die vorliegende Bachelorarbeit mit dem Titel: „Browser-Plugin zur Darstellung von autocomplete-Attributen“ selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden. Ich habe die Bedeutung der ehrenwörtlichen Versicherung und die prüfungsrechtlichen Folgen (§ 24 Abs. 2 Bachelor-SPO (7 Semester) der HdM) einer unrichtigen oder unvollständigen ehrenwörtlichen Versicherung zur Kenntnis genommen.

Nürtingen, 9. Oktober 2023,

Abstract

This work collects currently available testing-tools for success criterion 1.3.5 of the Web Content Accessibility Guidelines and compares them with each other. The comparison shows that current testing-applications are insufficient in their approach and functionality. The testers always have to assess themselves, whether the given autocomplete-attribute-value is correct and needed.

The Autocomplete-Check-Plugin, written as part of this work, is superior to the other testing-tools and stands out by its unique feature of heuristically generated autocomplete-suggestions, thus best supporting the testing-process. The main components of the plugin are shown and their implementation is explained. Additionally, the methodology, applied during this work, is covered. The validation of the plugin is done on previously unknown test-websites and confirms the Autocomplete-Check-Plugin's high accuracy when performing the prediction of autocomplete-values and is therefore an excellent choice when evaluating websites for the success criterion 1.3.5. The plugin is published on GitHub under the MIT-license.

Kurzfassung

Diese Arbeit stellt aktuell verfügbare Prüf-Tools auf Erfolgskriterium 1.3.5 der Web Content Accessibility Guidelines 2.1 (WCAG) zusammen und vergleicht diese miteinander. Der Vergleich zeigt, dass die derzeitigen Test-Anwendungen nicht ausreichend in ihrem Vorgehen und ihrer Funktionalität sind. Die Prüferinnen und Prüfer müssen dabei immer selbst beurteilen, ob der autocomplete-Attributs-Wert korrekt und erforderlich ist.

Das im Zuge dieser Arbeit programmierte Autocomplete-Check-Plugin ist den aktuellen Prüf-Tools vor allem durch das Alleinstellungsmerkmal der heuristisch getroffenen autocomplete-Vorschläge überlegen und unterstützt und komplementiert den Prüfvorgang somit bestens. Die wichtigsten Komponenten des Plugins werden vorgestellt und deren Implementierung erläutert. Außerdem werden die methodischen Vorgehensweisen, die in dieser Arbeit angewendet wurden, behandelt. Die Validierung des Plugins wurde anhand vorher unbekannter Test-Webseiten durchgeführt, sie bescheinigt dem Autocomplete-Check-Plugin eine hohe Genauigkeit bei der Vorhersage von autocomplete-Werten, damit ist es hervorragend geeignet, Webseiten auf das Erfolgskriterium 1.3.5 zu untersuchen. Das Plugin wird unter der MIT-Lizenz auf Github veröffentlicht.

Inhaltsverzeichnis

Bibliografie

[1] Jonathan Avila. *Autocomplete Favlet*. URL: https://github.com/mraccess77/mraccess77.github.io/blob/master/favlets/autocomplete_attribute.js. (commit: f707bfb).

[2] Alaina Birney. *WCAG Version History*. URL: https://accessibleweb.com/wcag/wcag-version-history/. (accessed: 07.09.2023).

[3] Chromium Source Code. *Autofill*. URL: https://source.chromium.org/chromium/chromium/src/+/main:components/autofill/README.md. (version: 120.0.6047.3).

[4] W3C World Wide Web Consortium. *Identify Input Purpose (Level AA)*. URL: https://www.w3.org/WAI/WCAG21/Understanding/identify-input-purpose.html. (accessed: 07.09.2023).

[5] W3C World Wide Web Consortium. *Identify Purpose (Level AAA)*. URL: https://www.w3.org/WAI/WCAG21/Understanding/identify-purpose.html. (accessed: 07.09.2023).

[6] W3C World Wide Web Consortium. *WCAG 3 Introduction*. URL: https://www.w3.org/WAI/standards-guidelines/wcag/wcag3-intro/. (accessed: 07.09.2023).

[7] W3C World Wide Web Consortium. *Web Content Accessibility Guidelines (WCAG) 2.1*. URL: https://www.w3.org/TR/WCAG21/. (accessed: 07.09.2023).

[8] W3C World Wide Web Consortium. *Web Content Accessibility Guidelines (WCAG) 2.2*. URL: https://www.w3.org/TR/WCAG22/. (accessed: 07.09.2023).

[9] Inc. Deque Systems. *axe DevTools | Developer Tools for Accessibility Testing*. URL: https://www.deque.com/axe/devtools/. (accessed: 06.10.2023).

[10] Aktion Mensch e.V. *Richtlinien für barrierefreie Webinhalte (WCAG) 2.1*. URL: https://outline-rocks.github.io/wcag/translations/WCAG21-de/. (accessed: 02.10.2023).

[11] gesetze-im-internet.de. *BITV 2.0 - Verordnung zur Schaffung barrierefreier Informationstechnik nach dem Behindertengleichstellungsgesetz*. URL: https://www.gesetze-im-internet.de/bitv_2_0/BJNR184300011.html. (accessed: 07.09.2023).

[12] Bundesministerium des Innern und für Heimat. *Barrierefreie-Informationstechnik-Verordnung (BITV 2.0)*. URL: https://www.barrierefreiheit-dienstekonsolidierung.bund.de/Webs/PB/DE/gesetze-und-richtlinien/bitv2-0/bitv2-0-node.html. (accessed: 07.09.2023).

[13] 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. (accessed: 07.09.2023).

[14] Bundesministerium des Innern und für Heimat. *Harmonisierte Europäische Norm (EN) 301 549*. URL: https://www.barrierefreiheit-dienstekonsolidierung.bund.de/Webs/PB/DE/gesetze-und-richtlinien/en301549/en301549-node.html. (accessed: 07.09.2023).

[15] Bundesministerium des Innern und für Heimat. *Web Content Accessibility Guidelines 2.1 (WCAG 2.1)*. URL: https://www.barrierefreiheit-dienstekonsolidierung.bund.de/Webs/PB/DE/gesetze-und-richtlinien/wcag/wcag-node.html. (accessed: 07.09.2023).

[16] Libraries.io. *webpack published releases on npm - Libraries.io*. URL: https://libraries.io/npm/webpack/versions. (accessed: 06.10.2023).

[17] Libraries.io. *webpack published releases on npm - Libraries.io*. URL: https://libraries.io/npm/jest/versions. (accessed: 02.10.2023).

[18] Inc. Meta Platforms und affiliates. *Getting Started · Jest*. URL: https://jestjs.io/docs/. (accessed: 06.10.2023).

[19] Inc. Meta Platforms und affiliates. *Jest. Delightful JavaScript Testing*. URL: https://jestjs.io/. (accessed: 02.10.2023).

[20] Microsoft. *TypeScript Documentation*. URL: https://www.typescriptlang.org/docs/. (accessed: 06.10.2023).

[21] npmtrends. *browserify vs esbuild vs parcel vs rollup vs snowpack vs vite vs webpack | npm trends*. URL: https://npmtrends.com/browserify-vs-esbuild-vs-parcel-vs-rollup-vs-snowpack-vs-vite-vs-webpack. (accessed: 06.10.2023).

[22] npmtrends. *cypress vs jasmine vs jest vs mocha vs nightwatch vs puppeteer vs vitest | npm trends*. URL: https://npmtrends.com/cypress-vs-jasmine-vs-jest-vs-mocha-vs-nightwatch-vs-puppeteer-vs-vitest. (accessed: 02.10.2023).

[23] Chris Pederick. *Web Developer*. URL: https://github.com/chrispederick/web-developer/. (version: firefox-2.0.1).

[24] Chrome Web Store. *Web Developer - Chrome Web Store*. URL: https://chrome.google.com/webstore/detail/web-developer/bfbameneiokkgbdmiekhjnmfkcnldhhm. (accessed: 06.10.2023).

[25] Akhila Sri Manasa Venigalla und Sridhar Chimalakonda. *WAccess*. URL: https://github.com/AkhilaSriManasa/WAccess. (commit: 5a0a240).

[26] w3schools. *TypeScript Tutorial*. URL: https://www.w3schools.com/typescript/index.php. (accessed: 06.10.2023).

[27] webpack. *Concepts | webpack*. URL: https://webpack.js.org/concepts/. (accessed: 06.10.2023).

[28] webpack. *webpack GitHub*. URL: https://github.com/webpack/webpack/. (accessed: 06.10.2023).

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Die barrierefreie Gestaltung von Webinhalten ist in unserem Alltag ein wichtiges Thema, die allen Menschen zugute kommt. Eine möglichst barrierefreie Webseite ist einfacher zu bedienen und verbessert die User-Erfahrung. Nicht zuletzt dadurch, aber auch aufgrund der rechtlich vorgeschrieben Barrierefreiheit staatlicher Webangebote durch die Barrierefreie-Informationstechnik-Verordnung 2.0 (BITV)[11], ist die digitale Barrierefreiheit von höchster aktueller Bedeutung. Die 2022 verabschiedete Verordnung zum Barrierefreiheitsstärkungsgesetz (BFSGV) verlangt sogar eine barrierefreie Gestaltung für alle "Produkte und Dienstleistungen, die nach dem 28.06.2025 in den Verkehr gebracht werden"[13]. Der internationale Standard für digitale Barrierefreiheit der Web Content Accessibility Guidelines (WCAG) enthält sogenannte Erfolgskriterien, die für eine barrierefreie Gestaltung, zum Beispiel einer Webseite, befolgt werden müssen. Damit das Prüfen auf diese Erfolgskriterien nicht mit hohem Aufwand händisch erfolgen muss, gibt es für genau diesen Anwendungszweck spezielle Prüf-Tools. Das grundlegende Erfolgskriterium dieser Arbeit ist das WCAG 2.1 Erfolgskriterium 1.3.5, welches vorschreibt, dass benutzerspezifische Eingabefelder und deren Verwendungszweck maschinell erkennbar sein müssen. Besondere Bedeutung kommt hierbei dem autocomplete-Attribut zu, das maschinenleslich einen Verwendungszweck ausdrücken kann.

Die Arbeit stellt eine Liste an aktuell verfügbaren Prüf-Tools zusammen, die Webseiten auf eben jenes Erfolgskriterium überprüfen. Nach Analyse dieser Zusammenstellung soll gezeigt werden, dass die bisherigen Prüf-Tools nicht ausreichend in ihrem Funktionsumfang sind und auch generell in Hinblick auf Usability und Effizienz verbessert werden können. Ein Browser-Plugin soll programmiert werden, das, basierend auf Heuristiken, automatische Vorschläge und Analysen vornimmt und diese sowie Informationen zum aktuellen autocomplete-Wert des geprüften Feldes visuell markiert.

Im theoretischen Teil der Arbeit wird zunächst der Ausgangspunkt der Arbeit, das Erfolgskriterium 1.3.5 vorgestellt und analysiert. Anschließend daran werden die Anforderungen an ein entsprechendes Prüftool aufgestellt. Die Liste an entsprechenden Anwendungen wird einer vergleichenden Untersuchung unterworfen. Dabei werden allgemeine Schwachstellen und Defizite herausgearbeitet. Auf diesen Erkenntnissen aufbauend soll ein Plugin programmiert werden, das die erkannten Schwächen behebt und durch eine heuristische Klassifikation von Eingabefeldern auf autocomplete-Attribut-Werte den Prüfprozess möglichst effizient und einfach gestalten soll.

Im praktischen Teil wird das im Rahmen dieser Bachelorarbeit programmierte Plugin vorgestellt. Dabei wird nicht nur auf die genaue Funktionsweise, sondern auch auf die konkrete Implementierung der wichtigsten Komponenten eingegangen.

Weiter werden die methodischen Vorgehensweisen dieser Arbeit aufgeführt und erläutert.

Zuletzt wird anhand eines Tests die Funktionalität des Plugins überprüft und validiert.

Hinweise

Die Begriffe “Plugin” und “Erweiterung” werden im Laufe dieser Arbeit gleichbedeutend verwendet. Der Begriff “WCAG 2.1 Erfolgskriterium 1.3.5” wird, zugunsten der Lesbarkeit oft verkürzt verwendet, bezieht sich jedoch stets auf das selbe Erfolgskriterium in Version 2.1 der WCAG, auch wenn nicht extra erwähnt. Der Einheitlichkeit halber wird der englische Begriff “autocomplete” immer klein geschrieben. Dies gilt auch für zusammengesetzte Wörter wie beispielsweise “autocomplete-Attribut”, nicht jedoch bei Namen, wie zum Beispiel dem “Autocomplete Favlet”.

2 Theoretischer Teil

2.1 WCAG Erfolgskriterium 1.3.5

WCAG

Die Web Content Accessibility Guidelines (WCAG) des World Wide Web Consortiums (W3C) sind ein internationaler Standard zur barrierefreien Gestaltung von Webinhalten.[15] Version 1.0 des Standards erschien am 5. Mai 1999 und beinhaltete hauptsächlich HTML-spezifische Richtlinien für Webseiten.[2] Hierfür wurden sogenannte Erfolgskriterien (engl. success criteria) eingeführt, die in drei Konformitätsstufen gestaffelt sind: A, AA und AAA – wobei in aufsteigender Reihenfolge jeweils die Priorität abnimmt, die Erfolgskriterien dafür aber umfangreicher und detaillierter werden. Diese Aufteilung der Konformitätsstufen wird noch heute im derzeit gültigen Standard verwendet. Nach WCAG 2.0 im Dezember 2008 wurden im Juni 2018 die aktuellen Richtlinien, WCAG 2.1, veröffentlicht. Neben zahlreicher Verbesserungen der bestehenden WCAG 2.0 Kriterien beinhaltet sie viele neue Erfolgskriterien, die zum Beispiel technologische Neuerungen wie Smartphones berücksichtigen. Aktuell wird an WCAG 2.2, mit einer voraussichtlichen Veröffentlichung im September 2023[8], sowie langfristig an WCAG 3.0 geschrieben[6]. Die WCAG basiert als Grundlage für die Richtlinien zur digitalen Barrierefreiheit der Europäischen Union [14], welche in Deutschland von der Barrierefreie-Informationstechnik-Verordnung (kurz BITV 2.0) umgesetzt wird[12].

Erfolgskriterium 1.3.5 Identify Input Purpose

Das Erfolgskriterium 1.3.5 Identify Input Purpose (dt. Bestimmung des Eingabezwecks) ist in der zweiten Konformitätsstufe AA und sollte somit für eine "gute Zugänglichkeit"[15] der Webinhalte sorgen. Das Kriterium befindet sich in Prinzip 1: "perceivable" (dt. wahrnehmbar) unter Richtlinie 3: "adaptable" (dt. anpassbar). Das Kriterium schreibt vor, dass der Zweck jedes Eingabefelds (engl. input field) programmatisch, also durch eine Software bestimmbar sein muss. Ein maschinelle Bestimmung des Eingabezwecks soll laut Kriterium dann möglich sein, wenn der Zweck des Eingabeelements, welches ausschließlich Nutzerdaten enthalten darf, einer der unten aufgelisteten Kategorien entspricht. Dazu wird vorausgesetzt, dass eine entsprechend technologische

Implementierung beziehungsweise Umsetzung gegeben ist. Die bisher einzige standardisierte technische Lösung dafür ist die Verwendung des autocomplete-Attributs. Ein autocomplete-Wert muss, um einen bestimmten Eingabezweck anzugeben, einer vordefinierten Liste an Verwendungszwecken wie zum Beispiel “email”, “username” und “language” entstammen.

Folgende Werte werden konkret in WCAG 2.1 benannt[7]:

- name
- honorific-prefix
- given-name
- additional-name
- family-name
- honorific-suffix
- nickname
- organization-title
- username
- new-password
- current-password
- organization
- street-address
- address-line1
- address-line2
- address-line3
- address-level4
- address-level3
- address-level2
- address-level1
- country
- country-name
- postal-code
- cc-name
- cc-given-name
- cc-additional-name
- cc-family-name
- cc-number
- cc-exp
- cc-exp-month
- cc-exp-year
- cc-csc
- cc-type
- transaction-currency
- transaction-amount
- language
- bday
- bday-day
- bday-month
- bday-year
- sex
- url
- photo
- tel
- tel-country-code
- tel-national
- tel-area-code
- tel-local
- tel-local-prefix
- tel-local-suffix
- tel-extension
- email
- impp

Browser oder andere Client-Anwendungen können anhand des autocomplete-Werts einfach und selbstständig den Zweck des Eingabefelds bestimmen und gegebenenfalls durch entsprechende Symbole erweitern oder die Textbeschreibung des Eingabefelds durch verständlichere Werte ersetzen. Die maschinelle Bestimmung des Eingabefelds ermöglicht zudem das automatische Ausfüllen mit bereits bekannten Nutzerdaten.

Allgemein sorgt das Erfolgskriterium dafür, dass das Ausfüllen von Eingabeelementen für alle Nutzerinnen und Nutzer einfacher wird. Menschen mit Gedächtnisbeeinträchtigungen profitieren von einer automatischen Ausfüllung von Eingabefeldern durch den Browser, da diese Informationen so nicht gemerkt werden müssen. Hilfsprogramme können anhand des autocomplete-Attributs Eingabefelder durch beschreibende Symbole erweitern, was die Eingabefelder für manche Menschen einfacher identifizierbar macht. Menschen mit motorischen Beeinträchtigungen profitieren ebenfalls vom automatischen Ausfüllen des Browsers, da so der benötigte Aufwand zur Eingabe minimiert wird.[4]

Risiken

Eine falsche Anwendung von autocomplete-Attributen bei nicht-personenbezogenen Eingabefeldern kann eventuell dazu führen, dass Nutzerinnen und Nutzer unbewusst und unwillentlich persönliche Informationen preisgeben, die vom Browser einfach automatisch ausgefüllt wurden. Der ausfüllende Browser kann nicht beurteilen, ob im aktuellen Fall persönliche Daten überhaupt erforderlich sind und geht nicht sehr restriktiv mit den hinterlegten Nutzerdaten um.

2.2 Abgrenzung zu WCAG Erfolgskriterium 1.3.6

WCAG Erfolgskriterium 1.3.6

Das Erfolgskriterium 1.3.6 Identify Purpose (dt. Bestimmung des Zwecks) ist von Konformitätsstufe AAA und stellt somit ein sehr spezifisches Kriterium dar. Hierbei soll der Zweck von Inhalten, die "mit Auszeichnungssprachen (engl. markup languages, wie z.B. HTML) implementiert sind", "Bestandteile der Benutzerschnittstelle, Symbole (engl. icons) und Regionen"[10], maschinell bestimmt werden können. "Bestandteile der Benutzerschnittstelle" sind Steuerelemente, die mit einer klaren Funktion assoziiert werden, beispielsweise Knöpfe zum Absenden von Formularen oder ein Link. Regionen sind meist inhaltlich, zweckdienlich oder räumlich getrennte Bereiche, in HTML etwa "footer" und "form" oder andere ARIA-landmarks.

Beispielsweise können Icons auf einer Webseite entsprechend markiert werden, sodass Hilfsprogramme diese durch ein eigenes Set an Icons austauschen kann.[5]

Vergleich der Erfolgskriterien 1.3.5 und 1.3.6

Auch wenn beide Kriterien eine bessere Zugänglichkeit der Webinhalte durch maschinelle Identifizierung ermöglichen sollen, unterscheiden sie sich doch in ihrem konkreten Geltungsbereich: Kriterium 1.3.5 gilt für Eingabeelemente, während Kriterium 1.3.6 viel allgemeiner zum Beispiel Bereiche beschreibt. Die Anwendung von Erfolgskriterium 1.3.5 erfolgt über autocomplete-Attribute, Kriterium 1.3.6 hauptsächlich über ARIA-landmarks und HTML-Tags.

2.3 Aktuelle Anwendung des Erfolgskriteriums 1.3.5

Beim Schreiben dieser Arbeit haben wir viele öffentliche Webseiten auf autocomplete-Attribute untersucht. Diese Ergebnisse wurden zwar nicht quantitativ festgehalten, jedoch lässt sich sagen, dass eine großer Anteil an öffentlichen Webseiten bei Eingabefeldern für persönliche Daten keine autocomplete-Werte verwendet. Oft werden falsche Werte verwendet[1] oder eigene Begriffe, welche nicht von WCAG vorgesehen sind. Eine detaillierte Analyse und Auswertung von autocomplete-Attributen und ihrer Verwendung auf öffentlichen Webseiten war im vorgesehenen Zeitrahmen leider nicht machbar. Mithilfe der Funktionalität zum Erstellen eines autocomplete-basierten Datensatzes des Autocomplete-Check-Plugins, beschrieben in Kapitel 3.5, könnte dies in Zukunft aber gut realisiert werden.

2.3.1 Autofill-Funktion bei Chromium

Die Autofill-Komponente von Chromium nutzt zur Bestimmung der autocomplete-Klassen vier aufeinander folgende Verfahren.[3] Im ersten Schritt wird mithilfe von Heuristiken lokal im Browser eine mögliche Klasse bestimmt. Dabei kommt sowohl Pattern-Matching mithilfe von vordefinierten Regex-Ausdrücken zum Einsatz, als auch Verfahren, die auf maschinellem Lernen basieren. Im zweiten Schritt wird eine anonymisierte Repräsentation des zu klassifizierenden Eingabefelds an einen Google-Server gesendet, welcher das Ergebnis der Klassifikation zurückschickt. Unterscheidet sich dieses von der lokalen Klassifikation, überstimmt das Cloud-basierte Ergebnis das lokale Ergebnis. Aufgrund der proprietären Natur dieses Google-Dienstes lässt sich nur mutmaßen, wie genau die Klassifizierung innerhalb des Google-Servers abläuft, jedoch liegt die Vermutung nahe, dass Google aufgrund der hohen Popularität seines Chromium-basierten Browsers Chrome, der die Cloud-basierte Klassifikation standardmäßig aktiviert hat, auf einen gigantischen Datensatz an vergleichbaren Eingabefeldern oder gar Einträge zur aktuell besuchten Webseite, zurückgreifen kann, um die Klassifizierung

[1] zum Beispiel “passwort” statt “current-passwort”.

zu unterstützen. Dieser Schritt lässt sich optional in den Einstellungen deaktivieren. Als drittes Verfahren wird auf Vorhandensein des autocomplete-Attributs im Eingabefeld geprüft. Ist dieses vorhanden und hat einen validen Wert, überstimmt dieser Wert die beiden vorherigen Schritte. Im Experiment haben wir jedoch festgestellt, dass bei offensichtlich falschen autocomplete-Werten die Autofill-Funktion Chromiums deaktiviert wird. Nach der Klassifizierung aller Eingabefelder erfolgt der letzte Schritt, bei dem die Anordnung der Eingabefelder untereinander beurteilt wird. So werden unter anderem Fehler ausgeschlossen, bei dem "address-line1" auf "address-line2" folgt. Das Ergebnis der Klassifikation kann nun verwendet werden, um die Felder mit hinterlegten Zahlungsinformationen, Adressdaten oder Zugangsdaten zu füllen. Mit Klick auf ein leeres Eingabefeld erscheint die Option, die gespeicherten Daten auszuwählen. Die passenden Felder werden entsprechend gefüllt.

Es fällt auf, dass Chromium in der Klassifikation zwar für jeden autocomplete-Wert eine äquivalente Klasse besitzt, jedoch eine umfangreichere Liste an Klassen hat. Chromium führt hier ganze 70 Werte, während WCAG in den Web Content Accessibility Guidelines nur 53 benennt. Dies liegt teils daran, dass Chromium manchmal detailliertere Kategorien verwendet. Chrome unterscheidet zum Beispiel zwischen dem Kreditkarten-Ablaufdatum, wenn dieses zwei- oder vierstellig ist oder führt verschiedene Kategorien für unterschiedliche Datumsschreibweisen. Allerdings benutzt Chromium auch Klassen, die gar nicht in der HTML-Spezifikation vorgesehen sind, beispielsweise "passport" und "travel-destination".

2.4 Vergleichbare Anwendungen

Anforderungen

Eine manuelle Prüfung auf autocomplete-Attribute mit Blick in den Quellcode ist sehr mühsam. Alle gängigen Desktop-Browser-Anwendungen haben integrierte Entwicklerwerkzeuge, wie beispielsweise die Chrome DevTools im Chrome-Browser, umgangssprachlich auch Inspektor genannt. Dieser ermöglicht es zum Beispiel recht einfach per Mausklick, das zu untersuchende Feld auszuwählen. Im parallel angezeigten Quellcode wird das entsprechende Element direkt gesondert hervorgehoben, sodass die Attribute gut abgelesen werden können. Um eine gesamte Webseite auf autocomplete-Attribute zu überprüfen, muss dieser Schritt nun pro Feld wiederholt werden. Je Feld muss der Prüfer oder die Prüferin zudem beurteilen, welcher autocomplete-Wert infrage kommen könnte. Dieser Vorgang soll als absolute Mindestherausforderung für eine Anwendung, die auf das Erfolgskriterium 1.3.5 prüft, angenommen werden. Dies beinhaltet, dass die Anwendung aus dem Browser heraus aufrufbar sein soll, um das Prüfverfahren nicht zu verkomplizieren. Quelloffene Tools sind zwar wünschenswert, um die genaue Funktionsweise nachvollziehen zu können, aber kein Muss. Eine ideale Anwendung könnte

eine vollautomatisierte Prüfung vornehmen, wodurch der Prüfaufwand zeitlich gesehen wesentlich verbessert wird. In Annäherung daran ist der bestmögliche Beistand beim Identifizieren und Hervorheben der Eingabefelder gewünscht, sowie Unterstützung bei der Frage, ob im konkreten Fall ein autocomplete-Attribut gefordert ist und wenn ja, welchen Wert dieses Feld haben sollte.

Obwohl das Erfolgskriterium 1.3.5 von Konformitätsstufe AA ist, haben wir kaum Tools gefunden, die konkret für dieses Erfolgskriterium entwickelt wurden. Daher wurden auch Anwendungen aufgenommen, die eine manuelle Prüfung von Attributen erleichtern.

Die erstellte Auswahl umfasst neben den Chrome DevTools, die hier als Vergleichswert mitgenannt werden, drei weitere Anwendungen, die alle quelloffen sind und in unterschiedlichster Weise auf das autocomplete-Attribut testen.

- Chrome DevTools (Inspektor)
- Web Developer von Chris Pederick [23]
- WAccess von Akhila Sri Manasa Venigalla und Sridhar Chimalakonda [25]
- Autocomplete Favlet von Jonathan Avila [1]
- axe DevTools-Plugin von Deque Systems, Inc. [9]

Im Folgenden wird ein kurzer Überblick über alle ausgesuchten Anwendungen vermittelt, sowie beschrieben, wie konkret die Überprüfung auf autocomplete-Attribute damit abläuft.

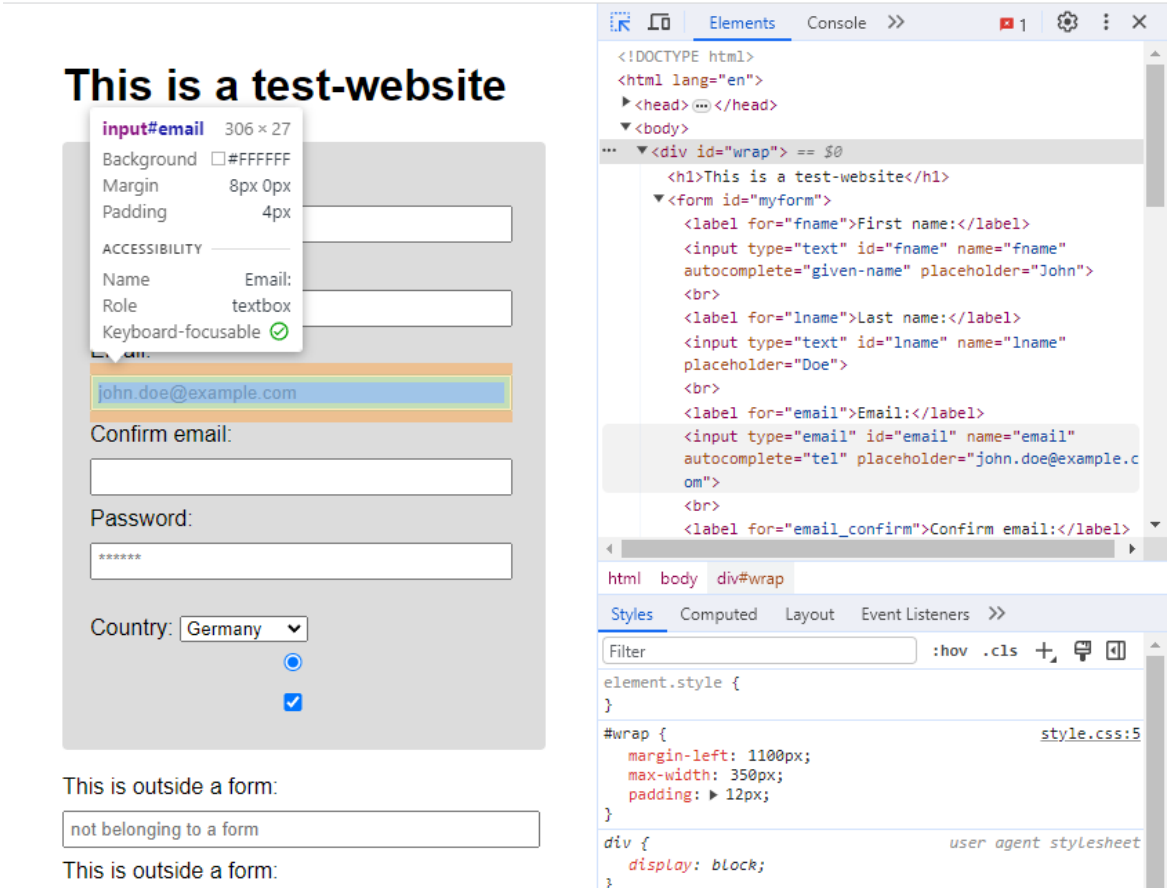
Chrome DevTools

Da sich bei den gängigen Browsern die Entwicklerwerkzeuge in ihrer Grundfunktion nicht groß unterscheiden, werden hier stellvertretend die Entwicklerwerkzeuge von Chrome untersucht. Der Chrome Inspektor bietet die Möglichkeit per Mausklick auszuwählen, sodass der Quellcode zu der entsprechenden Stelle springt und passend aufgefächert wird. So können direkt am HTML-Code alle Attribute abgelesen werden.

Unter dem Reiter “Elements” und dem Unterreiter “Classes & Attributes” können die Attribute von einem ausgewählten Element gesondert betrachtet werden, was eine bessere Lesbarkeit als bei reinem HTML-Code ermöglicht.

Außerdem kann im Reiter “Elements” und dem Unterreiter “Properties” nach Text gefiltert werden. Gibt man dort “autocomplete” ein, wird direkt der Wert des autocomplete-Attributs angezeigt. Das Textfeld zum Filtern der Properties merkt sich die Eingabe, wenn ein anderes Eingabefeld markiert wird und muss so nicht neu ausgefüllt werden.

Abbildung 2.1: Die Chrome DevTools mit ausgewähltem Input-Element



Insgesamt ist die Überprüfung auf autocomplete-Attribute mit den DevTools sehr zeitaufwendig, erfordert viele verschiedene manuelle Eingaben und ist dabei nur begrenzt übersichtlich. Außerdem gibt die Erweiterung keine Hilfestellung bei der Entscheidung, ob das autocomplete-Attribut korrekt ist und überhaupt benötigt wird.

Abbildung 2.2: Die Klassen- und Attribut-Ansicht der Chrome DevTools

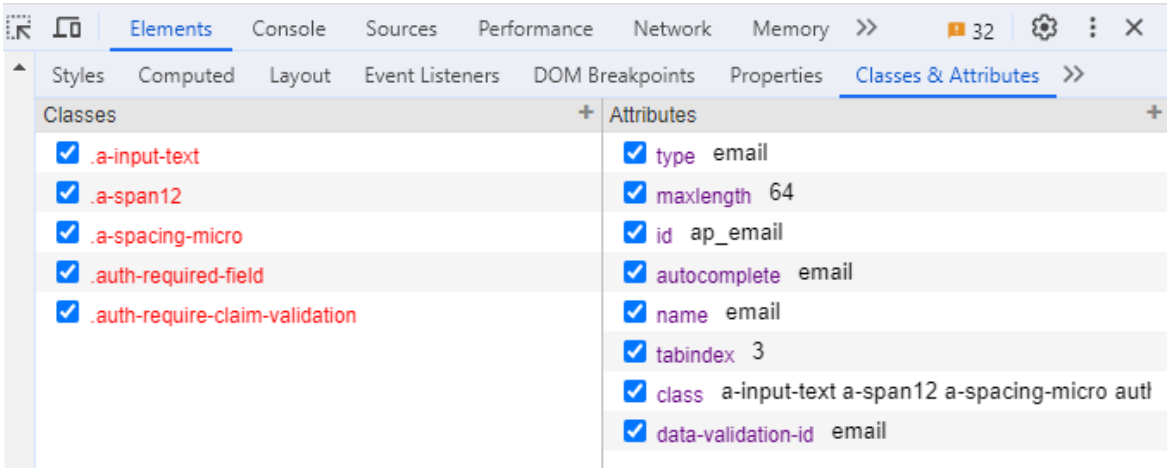
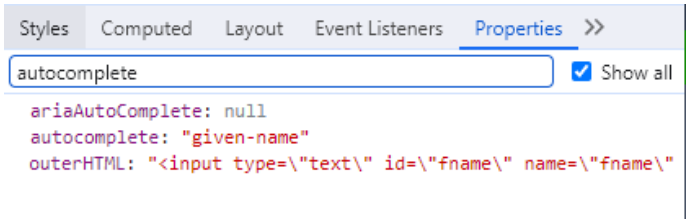


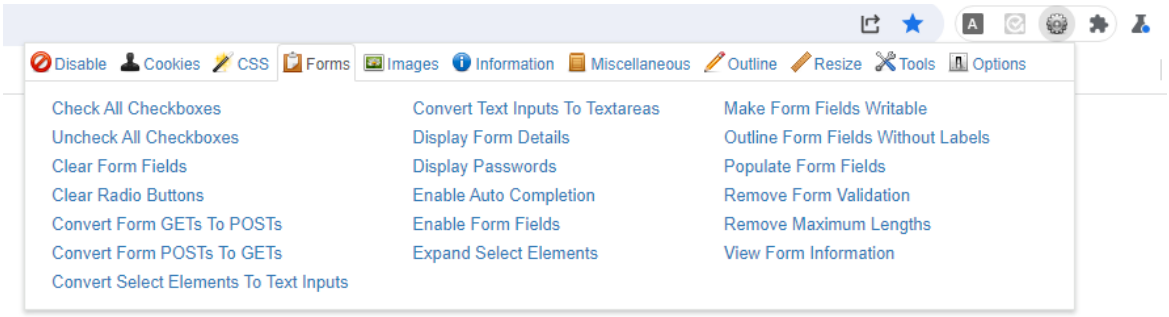
Abbildung 2.3: Die auf autocomplete gefilterte Properties-Ansicht der Chrome DevTools



Web Developer

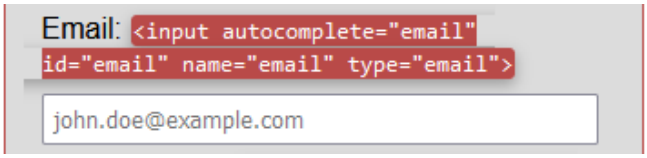
Web Developer ist eine Browser-Erweiterung für Chrome, Firefox und Opera, die mit über einer Millionen Nutzerinnen und Nutzer allein auf Chrome wohl eine der bekanntesten Browser-Erweiterungen für Web Entwickler ist. [24] Die Anwendung hat einen sehr umfangreicher Bestand an Helferfunktionen, deren Abdeckungsgebiet von CSS über Cookies und Bilder bis hin zu Formularen und vielem mehr geht. Bedient wird die Erweiterung über eine Werkzeugleiste, die mit einem Klick auf das Icon von Web Developer geöffnet werden kann.

Abbildung 2.4: Die Werkzeugleiste von Web Developer



Um mit Web Developer auf autocomplete-Attribute zu testen, kann man die Funktion "Display Form Details" benutzen, die vor jedem Eingabefeld auf der Webseite den jeweiligen HTML-Code als Textelement einfügt. Ist das autocomplete-Attribut vorhanden, kann dieses einfach abgelesen werden.

Abbildung 2.5: Ein Email-Feld nach aktivierter Display Form Details-Funktion

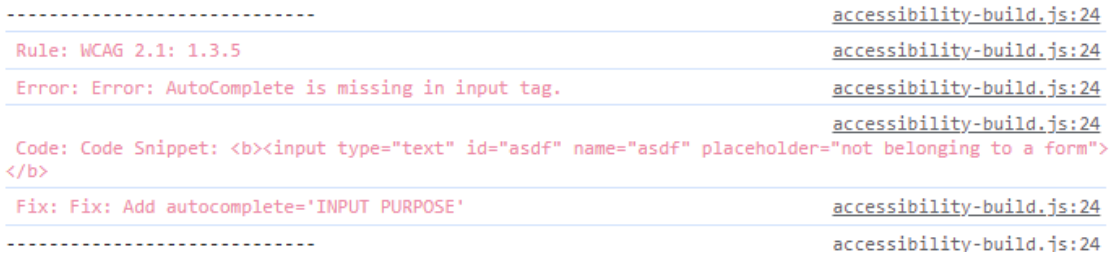


Auch wenn Web Developer zuverlässig alle Eingabefelder findet und markiert, und somit genau die richtigen Elemente für Kriterium 1.3.5 markiert, hat die Erweiterung dennoch viele Nachteile in Bezug auf das Testen des Erfolgskriteriums. Da der gesamte HTML-Code eines jeden Eingabeelements hinzugefügt wird, verliert man bei Feldern mit vielen Attributen schnell den Überblick und es kann gegebenenfalls lange dauern, bis alle Attribute überflogen und das autocomplete-Attribut gefunden wurde. Eine Analyse des autocomplete-Wertes oder eine Vorhersage fehlt gänzlich.

WAccess

WAccess ist eine Browser-Erweiterung, die gleich mehrere Erfolgskriterien verschiedener Konformitätsstufen von WCAG 2.0 und WCAG 2.1 auf Knopfdruck automatisiert testet. Über einen Eintrag im Kontextmenü (Rechtsklick-Optionen) wird der automatische Analyse-Prozess der aktuellen Webseite gestartet. Die Ergebnisse der Tests werden in der Konsole des Browsers ausgegeben. Beim autocomplete-Test wird geprüft, ob auf jedem Input-Element ein autocomplete-Attribut gesetzt ist, ansonsten wird das entsprechende Feld im Ergebnisbericht bemängelt.

Abbildung 2.6: Auszug aus dem Konsolen-Log von WAccess mit Prüfung auf autocomplete

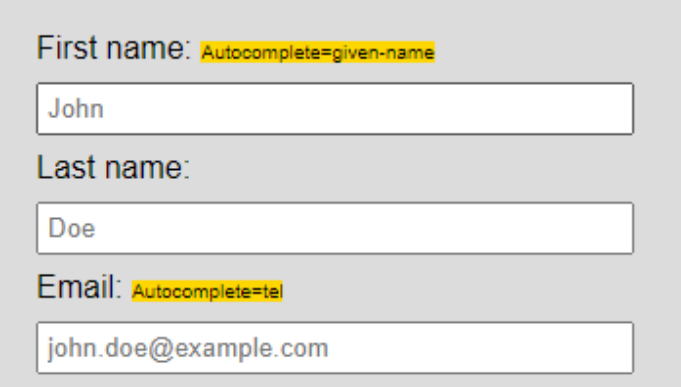


WAccess ist äußerst praktisch, wenn neben Erfolgskriterium 1.3.5 auch noch andere Kriterien geprüft werden sollen. Bei großen HTML-Seiten wird der dann generierte Bericht in der Konsole aufgrund seiner Länge schnell unübersichtlich. Zwar kann mit der Text-Suchfunktion zu Ergebnissen betreffend Kriterium 1.3.5 gesprungen werden, allerdings ist dennoch schlecht auf einen Blick ersichtlich, wie das Gesamtergebnis in Bezug auf die autocomplete-Attribut-Prüfung ausfällt. Eine große Schwachstelle ist, dass die bemängelten Felder nicht in der gerenderten HTML-Ansicht markiert werden. Im Ergebnislog wird der HTML-Code des gesamten Input-Felds ausgeschrieben. Die einzige Möglichkeit, den Feldern ihre visuellen Repräsentationen zuzuordnen, ist eine händische Suche über den Inspektor. Dieser Zuordnungsprozess relativiert die gewonnene Zeitersparnis durch die Automatisierung der Tests. Im Ergebnislog wird zur Behebung des Problems bei fehlenden autocomplete-Attributen zwar vorgeschlagen, das autocomplete-Attribut hinzuzufügen, jedoch wird kein Wert vorgeschlagen und eine differenzierte Analyse, ob das Attribut überhaupt benötigt wird, findet auch nicht statt. Ein weiterer Nachteil ist, dass lediglich nach Input-Elementen gesucht wird. Möglich sind jedoch auch Select- und Textarea-Elemente.

Autocomplete Favlet

Das Autocomplete Favlet ist ein Bookmarklet[2], das auf Knopfdruck alle vorhandenen autocomplete-Attribute in der aktuellen Webseite sucht, und falls vorhanden diese mit Wert als grafischen Marker vor das zugehörige Feld platziert. Über einen Alert wird nach Auslösen des Bookmarklets angezeigt, wie viele autocomplete-Attribute gefunden wurden.

Abbildung 2.7: autocomplete-Markierungen durch das Autocomplete Favlet



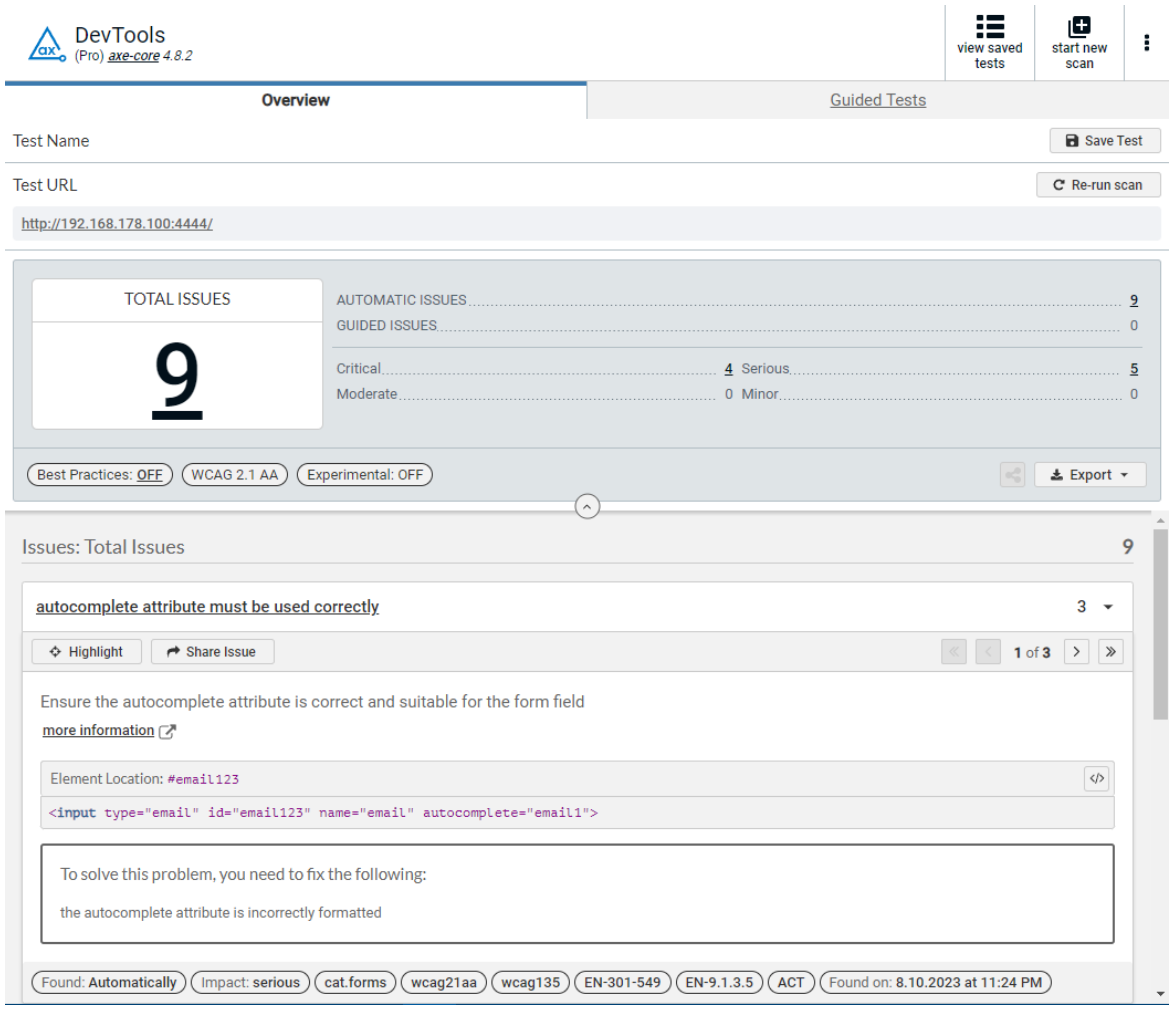
Zwar ist das Autocomplete Favlet recht schlicht, doch dafür äußerst effektiv. Die Marker verschaffen einen guten Überblick, welche Felder ein autocomplete-Attribut besitzen. Durch die interne Verwendung der Javascript-Funktion "querySelectorAll" liest das Bookmarklet nur laut HTML gültige autocomplete-Werte aus. Wäre dieser im HTML-Code beispielsweise "test", würde im Marker "undefined" stehen. Einerseits ist diese indirekte Prüfung nützlich, um zu prüfen, ob vorhandene Werte gültig sind, andererseits vermittelt die Ausgabe von "undefined" einen falschen Eindruck, wenn man den tatsächlichen Wert aus dem HTML-Code erwartet. Um diesen herauszufinden, muss dann der Inspektor verwendet werden. Außerdem wird nicht überprüft, ob autocomplete benötigt wird, oder ob der richtige Wert pro Feld verwendet wird. Das Autocomplete Favlet prüft auch nicht, ob es sich bei dem markierten Element um ein Eingabeelement handelt. Jedes HTML-Element, das ein autocomplete-Attribut besitzt, wird markiert. Um die generierten Marker wieder zu entfernen, muss die Webseite neu geladen werden, wodurch Usereingaben verloren gehen können. Wird ein Bookmarklet als Link zum Skript-Code im Internet angelegt, besteht zudem ein Sicherheitsrisiko. Da der Code jederzeit geändert werden kann, ohne dass der Browser darüber benachrichtigt wird, könnte theoretisch schädlicher Code eingeschleust werden.

[2]Ein Bookmarklet ist ein Script, das quasi wie ein Lesezeichen im Browser gespeichert wird. Auf Aufruf wird das Skript vom gespeicherten Pfad geöffnet und im Kontext der Webseite ausgeführt.

axe DevTools

Axe DevTools ist ein Plugin für die Browser Chrome, Firefox und Edge und basiert auf der quelloffenen Accessibility Engine (kurz axe) "axe-core" und wird von der Firma Deque Systems, Inc. entwickelt.[9] Das Plugin ist kostenlos verfügbar, kann in dieser Version aber nur einen gesamten Scan der Webseite auf mehrere Erfolgskriterien vornehmen. Eine kostenpflichtige Version erlaubt sogenannte "intelligent guided tests", bei denen Interaktiv einzelne Komponten einer Webseite getestet werden können, beispielsweise nur Tabellen, auf tabellenspezifische Tests. Neben den Browser-Plugins sind außerdem Handy-Apps für Android und iOS verfügbar. Das Plugin lässt sich über einen eigenen Bereich im Inspektor steuern, auf Knopfdruck kann dort der gesamte Webseiten-Scan gestartet werden. Dieser läuft vollautomatisch ab. Das Ergebnis wird im oberen Bereich des Berichts zusammengefasst, indem die Gesamtzahl an Problemen groß angezeigt wird. Die gefundenen Probleme werden nach Fehlern gruppiert, und mit detaillierter Beschreibung und Vorschlag zum Beheben angezeigt. Über einen Knopf kann das zugehörige HTML-Element hervorgehoben werden. Sind die detaillierte Problembeschreibungen aufgeklappt, ist die große Menge an dargestellten Informationen etwas unübersichtlich. (siehe Abbildung 2.8)

Abbildung 2.8: Testergebnis eines Seiten-Scans mit den axe DevTools



Das axe DevTools-Plugin testet zwar auf das Erfolgskriterium 1.3.5, allerdings nur insofern, dass geprüft wird, ob vorhandene autocomplete-Werte gültige Werte sind. Jedoch nicht geprüft wird, ob ein autocomplete-Wert benötigt wird, oder ob der richtige Wert verwendet wurde. Außerdem wird kein möglicher Wert vom Plugin vorgeschlagen, dafür aber ein Link angezeigt, unter dem die korrekte Verwendung der autocomplete-Attributs erklärt wird.

Vergleich der Anwendungsauswahl

Basierend auf den Anforderungen haben wir hier elf Kategorien definiert, in denen die verschiedenen Prüf-Tools beurteilt und daraufhin tabellarisch zum besseren Vergleich gegenübergestellt werden sollen. Die Kategorien beschreiben das Prüfverhalten der Anwendungen, wie leicht diese zu bedienen und zu verstehen sind, automatische Analysen der autocomplete-Werte auf deren Richtigkeit und korrekte Verwendung. Die Ergebnisse der Kategorie "Übersichtlichkeit" basieren auf der Beurteilung der Autoren und sind somit nicht streng objektiv. Die individuellen Ergebnisse in dieser Kategorie sind somit mit Vorsicht zu beachten. Im Vergleich zueinander entsteht aber ein guter Eindruck, ob eine Anwendung eine eher bessere oder schlechtere Übersichtlichkeit als ein anderes Tool hat.

Die folgende Tabelle vergleicht die Anwendungsauswahl in den elf Kategorien. Ergänzend zum Vergleich wurde zusätzlich die im Zuge dieser Arbeit erstellte Autocomplete-Check-Plugin aufgeführt. Für eine bessere Lesbarkeit wurden Kurzformen je Anwendung vergeben.

- Autocomplete Favlet: “AcF”
- Web Developer: “Dev”
- WAccess: “WAc”
- Chrome DevTools: “CDT”
- axe DevTools: “axe”
- Autocomplete-Check-Plugin “AcC”

	CDT	Dev	WAc	AcF	axe	AcC
prüft lokale HTML-Dateien	ja	ja	ja	ja	nein	ja
verändert den DOM nicht nachhaltig	ja	ja	ja	nein	ja	ja
automatische Markierung[3]	nein	ja	ja	ja	ja	ja
Übersichtlichkeit	schlecht	mäßig	sehr schlecht	sehr gut	gut	sehr gut
Klicks zum Ausführen	2	2-4[4]	2-3[5]	2	2-4[6]	0-1[7]
überprüfte Inhalte[8]	alle	i	i	alle	i,s,t	i,s,t
anpassbar	ja	ja	nein	nein	ja	ja
bemängelt fehlende AC-Werte	nein	nein	ja	nein[9]	nein	ja
AC-Wert Prüfung	nein	nein	nein	bedingt[10]	bedingt[10]	ja
AC-Vorschläge	nein	nein	nein	nein	nein	ja

Tabelle 2.1: Feature-Vergleich der ausgewählten Anwendungen

[3] höherer Grad der Automatisierung, der Felder selbst hervorhebt oder überprüft
[4] je nachdem, ob der Inspektor bereits geöffnet ist
[5] je nachdem, ob der Forms-Tab zuletzt im Popup ausgewählt war
[6] je nachdem, ob der Inspektor bereits geöffnet ist und wie breit das Inspektor-Fenster ist
[7] je nachdem, ob die Erweiterung beim Seitenaufruf aktiviert ist
[8] i: Inputs, s: Selects, t: Textareas
[9] Nicht markierte Felder müssen selbst identifiziert werden. Diese haben keinen Marker erhalten.
[10] Prüfung, ob der vorhandene autocomplete-Wert ein gültiger Wert ist, jedoch nicht, ob an dieser Stelle der richtige Wert verwendet wird

Anhand der Gegenüberstellung ist gut erkennbar, dass alle bisher verfügbaren, dedizierten Prüf-Tools die Grundanforderungen erfüllen und Feldattribute entsprechend kennzeichnen. Die Anwendungen sind generell schnell und einfach zu starten. Jedoch sind die Ergebnisse oft unübersichtlich und erfordern zusätzliche menschliche Beurteilung. Nur ein Tool markt das Fehlen eines autocomplete-Attributs aktiv an. Nur axe DevTools überprüft hinreichend auf zulässige HTML-Element-Typen. In zwei Fällen werden Select- und Textarea-Elemente gänzlich außen vor gelassen. Eine komplette Prüfung auf korrekte Verwendung des autocomplete-Attributs und die Richtigkeit des autocomplete-Wertes wird von keinem Tool durchgeführt. Ebenso gänzlich fehlt die Funktion, passende autocomplete-Werte vorgeschlagen zu bekommen.

Da gerade bei den letzten Kategorien, diese größtenteils nicht bis gar nicht von den verglichenen Anwendungen erfüllt werden, soll mit der Entwicklung des im Rahmen dieser Bachelorarbeit-Thesis vorgelegten Autocomplete-Check-Plugins diese Lücke geschlossen werden, um Prüferinnen und Prüfer bestmöglich beim Testen auf Erfolgskriterium 1.3.5 zu unterstützen. Zudem soll Autocomplete-Check in allen anderen Kategorien bestmöglich abschneiden, um den Testvorgang so schnell und angenehm wie möglich zu machen.

2.5 Aufbau von Chrome-Erweiterungen gemäß der Chrome-API

Das hier entwickelte Autocomplete-Check-Plugin verwendet die Chrome-API als Grundlage für die Programmstruktur. Somit deckt es neben Chrome indirekt auch Chromium-basierte Browser wie zum Beispiel Microsoft Edge oder Opera ab. Da Firefox eine weitgehend kompatible API verwendet, kann für eine Firefox-Erweiterung größtenteils der selbe Programmcode verwendet werden.

Chrome-Erweiterungen bestehen aus fünf grundlegenden Bausteinen: einer Manifest-Datei, einem Service-Worker, Content-Skripte, sowie einem PopUp und einer Einstellungs-Seite.

Manifest

Die manifest.json Datei ist der Einstiegspunkt der Erweiterung. Darin finden sich Informationen über die Erweiterung selbst, wie der Erweiterungs-Name und die Version der Erweiterung, gegebenenfalls die Sprache der Erweiterung, eine Kurzbeschreibung und die Urheberschaft. Im Manifest müssen Berechtigungen, welche die Erweiterung beim Browser anfragt, deklariert werden, wie zum Beispiel der Zugriff auf bestimmte Browser-Tabs oder die Möglichkeit, Dateien im Speicher des Browsers zu hinterlegen. All diese Daten können bereits vor der Installation einen guten Überblick über die Erweiterung bieten. Außerdem verweist das Manifest auf die Code-Dateien der Content-Skripte und andere Programm-Bausteine und verknüpft außerdem eine Icon-Datei als Logo.

Service-Worker

Der Service-Worker oder auch das Hintergrundskript läuft ab Installation der Erweiterung im Hintergrund des Browsers und reagiert auf verschiedene Events. Das kann zum Beispiel die Installation der Erweiterung sein, ein Klick auf das Icon der Erweiterung in der Lesezeichenliste, aber auch eine eingehende Nachricht von einem anderen Programmteil der Erweiterung. Der Service-Worker reagiert individuell auf eingehende Events, leitet diese eventuell an andere Programmbausteine weiter oder bearbeitet Daten.

Content-Skripte

Ein oder mehrere Content-Skripte sind Dateien, die im Kontext von Webseiten ausgeführt werden. Dies bedeutet, dass sie Zugriff auf den DOM der Webseite haben

und diesen lesen und verändern können. Mithilfe der Messaging-API können Content-Skripte mit anderen Programmteilen kommunizieren. Die Content-Skripte laufen in einer eigenen JavaScript-Umgebung, sodass keine Konflikte mit dem Code der aktuellen Webseite oder anderen Erweiterungen entstehen können. Diese klare Trennung dient auch sicherheitstechnischen Aspekten. Beispielsweise können schädliche Webseiten so keine persönlichen Daten aus einem Content-Script auslesen.

PopUp

Das PopUp ist ein eigenständiges Fenster, das durch Klick auf das Erweiterung-Icon im Browser geöffnet werden kann. Dieses überlagert einen Teil des Browser-Fensters und kann etwa ein Steuerungsmenü oder Informationen zur Erweiterung darstellen und mittels der Messaging-API Aktionen der Erweiterung auslösen.

Einstellungs-Seite

Die Einstellungs-Seite ist eine eigenständige HTML-Seite, mit der Einstellungen an der Erweiterung vorgenommen werden können. Im zugehörigen Skript kann das Verhalten und das Aussehen der Erweiterung oder andere denkbare Einstellungsmöglichkeiten beeinflusst werden. Beim Aufrufen der Einstellungs-Seite lädt die Erweiterung die aktuellen Werte je Einstellung aus dem Speicher und zeigt diese entsprechend an. Werden Änderungen vorgenommen und abschließend gespeichert, werden die neuen Werte im Browser-Speicher hinterlegt, sodass von Einstellungen betroffene Code-Abschnitte auf Grundlage der gespeicherten Einstellungswerte reagieren können.

Storage-API

Die Storage-API ermöglicht es der Erweiterung, Daten über das Schließen des Browsers hinaus zu speichern. Dies ist besonders nützlich für Benutzerdaten und Einstellungen. Die Daten werden in Form von Key-Value Paaren gespeichert und sind für alle Komponenten der Erweiterung zugreifbar. Es gibt drei gängige Speichermodelle, die es erlauben die Daten lokal, synchronisiert oder per Sitzung zu speichern. Standardmäßig dürfen vom lokalen Speicher 10MB belegt werden, über Berechtigungen im Manifest können aber auch Ausnahmen erfragt werden. Die Synchronisierung erlaubt es in Chrome angemeldeten Nutzerinnen und Nutzern, ihre Daten zwischen mehreren Browsern zu synchronisieren. Pro Erweiterung ist man hier jedoch auf 100KB begrenzt. Ist diese Option im Chrome-Browser deaktiviert, wird automatisch die lokale Speicherung verwendet. Sitzungsbasierter Speicher erlaubt es, Daten für die Dauer der Browsersitzung im Arbeitsspeicher zu hinterlegen – hauptsächlich, um globale Variablen abzubilden. Dies erleichtert die Kommunikation zwischen den Erweiterungs-Komponenten, da so nicht immer auf die Messaging-API zurückgegriffen werden muss.

Messaging-API

Da Content-Skripte im Kontext der aktuellen Webseite ausgeführt werden, haben sie keinen direkten Zugriff auf andere Erweiterungs-Bestandteile, wie zum Beispiel den parallel laufenden Service-Worker. Die Messaging-API erlaubt den bidirektionalen Austausch von Nachrichten zwischen den Komponenten. Diese werden als JSON-Objekt verschickt, können also nicht nur einfache Nachrichten, sondern auch ganze Datenstrukturen enthalten. Neben einfachen Nachrichten besteht auch die Möglichkeit, mit sogenannten langlebigen Verbindungen (engl. long-lived connections) einen längeren Austausch zu ermöglichen, wenn die Nachrichten dabei mehrmals hin und her gehen.

2.6 Verwendete Technologien

TypeScript

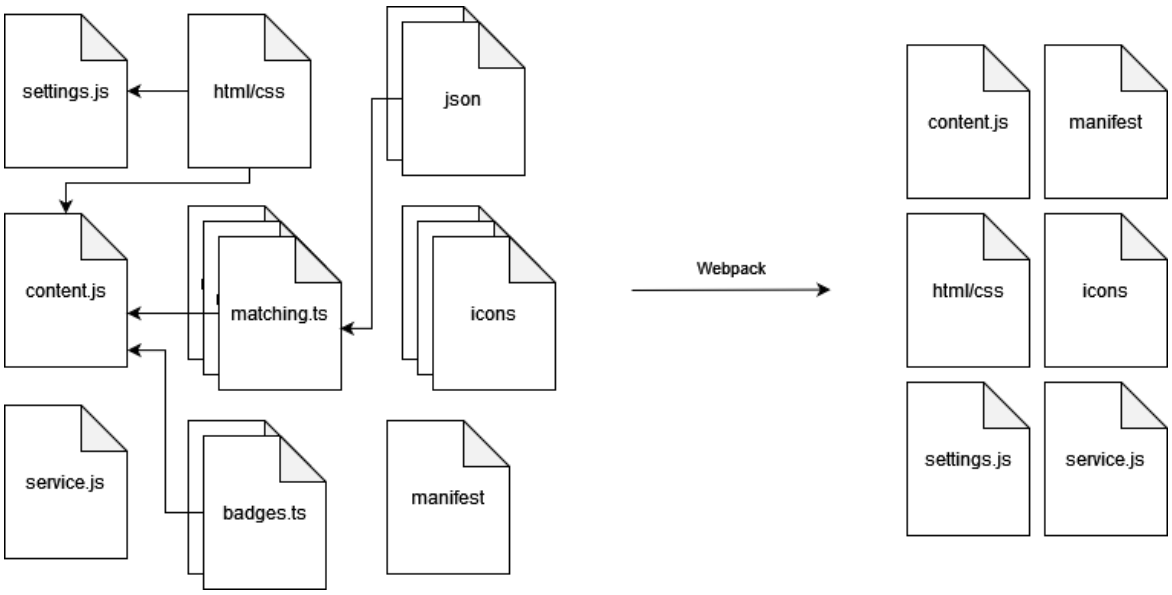
TypeScript ist eine freie und quelloffene Skriptsprache, die von Microsoft entwickelt wird und 2012 unter der Apache 2.0 Lizenz veröffentlicht wurde. Die Programmiersprache ist wesentlich von JavaScript beeinflusst und eine Obermenge davon, sodass gültiger JavaScript-Code automatisch gültiger TypeScript-Code ist. Dadurch kann die große Menge an verfügbaren JavaScript-Bibliotheken auch direkt in TypeScript verwendet werden. Ein großer Unterschied ist jedoch die statische Typisierung. Hierbei werden die Datentypen von Variablen, Methoden und Klassen bereits während der Kompilierung bestimmt. Dies geschieht anhand von Typ-Annotationen im Programmcode oder wird von den vorhandenen und übergebenen Typen abgeleitet. Moderne Entwicklungsumgebungen können so bereits beim Schreiben des Codes auf Fehler hinweisen, wenn etwa eine Variable möglicherweise auch ein anderer Typ, oder gegebenenfalls sogar undefiniert sein könnte. JavaScript verwendet im Gegensatz hierzu eine dynamische Typisierung. Die Typen-Bestimmung findet hauptsächlich beim Ausführen des Programms statt, sodass Fehler erst mit zusätzlichem Zeitaufwand oder möglicherweise gar nicht gefunden werden, sollte beim Testen der Anwendung ein bestimmter Fall nicht eintreten. Allerdings muss TypeScript-Code um in einer JavaScript-Applikation verwendet werden zu können, zu JavaScript-Code kompiliert beziehungsweise transpiliert werden, was einen zusätzlichen Schritt in der Bereitstellung der Software bedeutet. Außerdem können in größeren Software-Projekten die Typ-Annotationen große Mengen an zusätzlichem Code fordern.

Das hier vorgelegte Autocomplete-Check-Plugin verwendet ausschließlich TypeScript als Programmiersprache. Dies ist gerade bei großen Komponenten wie dem Matching-System äußerst hilfreich, um zum Beispiel die Datenstrukturen, die der Matching-Algorithmus verarbeitet, zu typisieren.

Webpack

Webpack ist ein freier und quelloffener Modul-Packer, der 2012 von Tobias Koppers auf Github unter der MIT-Lizenz veröffentlicht wurde. [28][16] Stand Oktober 2023 ist Webpack mit durchschnittlich 25 Millionen wöchentlichen Downloads auf dem Paket-Manager npm der meistgenutzte Modul-Packer. [21] Hauptanwendungszweck von Webpack ist das Zusammenführen mehrerer Javascript-Dateien zu einer Datei. Da das Laden vieler einzelner Skripte im Browser zu Performance-Einbußen führen kann, bietet es sich an, große Module, die aus mehreren Dateien bestehen, zu einer Datei zu vereinigen. Darüber hinaus macht Webpack es einfach, Software-Bibliotheken zu verwenden, da diese ebenfalls gepackt werden und nicht manuell der Anwednung beigefügt werden müssen. Zudem können mit Hilfe von Webpack Dateien transformiert werden, zum Beispiel die Übersetzung von TypeScript-Dateien zu Javascript. Aber auch andere Ressourcen wie Bild- oder HTML-Dateien können in den finalen Build inkludiert werden. Für das Autocomplete-Check-Plugin führt Webpack den TypeScript-Quellcode zu einzelnen Skripten zusammen und transformiert diese zu Javascript-Dateien. Jede Erweiterungs-Komponente, wie zum Beipiel der Service-Worker besteht somit nur noch aus einer einzelnen Datei, ohne das Referenzen auf externe Skripte oder Bibliotheken benötigt werden. Zudem kopiert Webpack die benötigten HTML- und CSS-Dateien, sowie die Manifest-Datei in den finalen Build-Ordner. Beim Bereitstellen der Erweiterung ergibt sich so eine enorme Zeitersparnis, da anstatt mehrerer einzelner Schritte und Hilfsprogramme lediglich ein Skript-Aufruf mit Webpack genügt, der nach wenigen Sekunden Laufzeit bereits den fertigen Erweiterungs-Ordner erstellt.

Abbildung 2.9: Transformationsprozess von Webpack



Jest

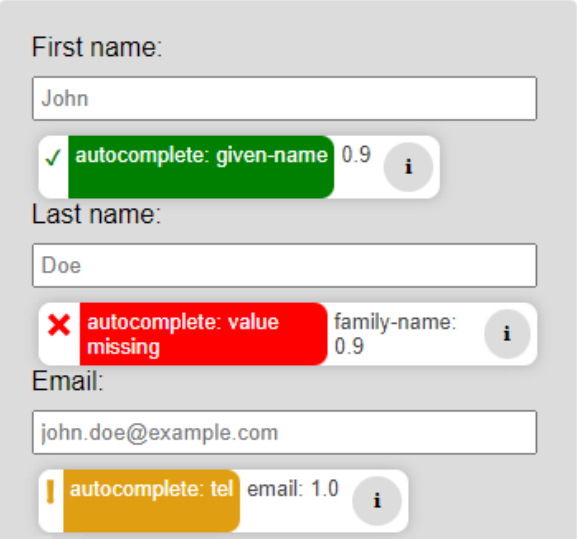
Jest ist ein Javascript-Test-Framework, das 2012 von Facebook unter der MIT-Lizenz veröffentlicht wurde.[19][17] Mit durchschnittlich 20 Millionen wöchentlichen Downloads auf npm (Stand Oktober 2023) ist es das mit Abstand meistgenutzte Test-Framework für Javascript. [22] Für das Autocomplete-Check-Plugin kommt Jest einerseits mit klassischen Unittests zum Einsatz, wie zum Beispiel bei wichtigen Kernkomponenten des Matching-Algorithmus. Zusätzlich wird Jest aber auch verwendet, um die korrekte Klassifikation der Erweiterung an lokal gespeichterte HTML-Seiten zu überprüfen. Hierzu werden in den Test-Dateien entsprechende Input-Felder der HTML-Seiten per Id identifiziert und manuell klassifiziert. Beim Durchlaufen der Tests wird dann das Topergebnis des Matching-Algorithmus, dem als Input das per Id bestimmte Feld übergeben wird, mit dem händisch vorgegebenen Wert verglichen. So kann automatisiert eine große Menge an Test-Webseiten, die gegebenenfalls Grenzfälle beinhalten, mit einer übersichtlichen Auflistung durchlaufen werden, welche Webseiten und einzelne Elemente korrekt klassifiziert wurden. So können zum Beispiel Feineinstellungen an der Gewichtung der einzelnen Tests im Matching-Algorithmus vorgenommen werden und deren Auswirkungen schnell und ohne menschliches Zutun getestet werden, sodass nicht für jede kleine Änderung ein händischer Durchlauf durch mehrere Webseiten mit der Erweiterung vorgenommen werden muss.

3 Praktischer Teil

3.1 Funktionalität des Plugins

Im Folgenden soll die Funktionsweise und Implementierung wichtiger Komponenten erklärt werden.

Abbildung 3.1: Beispielformular mit Markierungen durch das Autocomplete-Check-Plugin

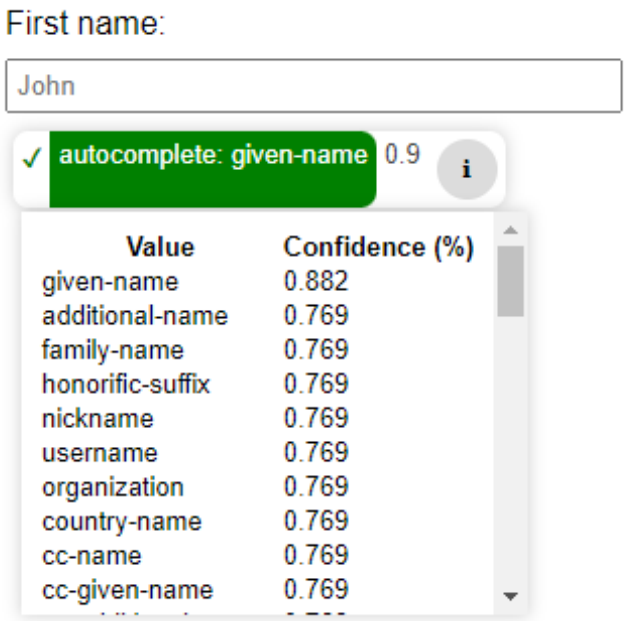


Das Autocomplete-Check-Plugin ermöglicht es Nutzerinnen und Nutzern, Eingabefelder auf autocomplete-Attribute zu testen. Auf Knopfdruck wird jedes Eingabefeld der aktuellen Webseite überprüft. Die Erweiterung ermittelt, ob auf dem jeweiligen Feld das autocomplete-Attribut gesetzt ist und falls ja, ob der korrekte Wert verwendet wurde. Hierzu ermittelt der sogenannte Matching-Algorithmus anhand verschiedener Kriterien maschinell einen Vorschlag für den autocomplete-Wert. Die korrekte Verwendung oder gegebenenfalls der Vorschlag der Erweiterung wird in farbig hervorgehobenen Markern unter jedem Eingabefeld angezeigt. Das untersuchte Input-Element wird mit dem Attribut "aria-describedby" versehen, sodass das autocomplete-Check-Plugin auch in Verbindung mit Screen-Readern verwendet werden kann. Nach Deaktivierung der Erweiterung werden alle Änderungen am DOM der aktuellen Webseite durch das

Autocomplete-Check-Plugin entfernt. Außerdem wird, sollte bei Eingabefeldern das "aria-describedby"-Attribut initial gesetzt gewesen sein, der ursprüngliche Wert wieder eingetragen. Dadurch kann die Webseite effektiv wieder in den Zustand vor der Benutzung der Erweiterung versetzt werden, sodass händische Eingaben bestehen bleiben und nicht etwa beim Neu-Laden der Seite verloren gehen.

Der Matching-Algorithmus, welcher in Kapitel 3.3 eigens genauer erklärt wird, liefert pro Eingabefeld einen Vorschlag für den Autocomplete-Wert, sowie eine Prozentzahl, wie zuverlässig das Ergebnis sein sollte. Durch Klick auf den Info-Knopf des Markers können außerdem tabellarisch die Wahrscheinlichkeiten für die übrigen Autocomplete-Werte eingesehen werden.

Abbildung 3.2: Alle Klassifizierungs-Ergebnisse eines Eingabefelds, sortiert nach Zuversicht

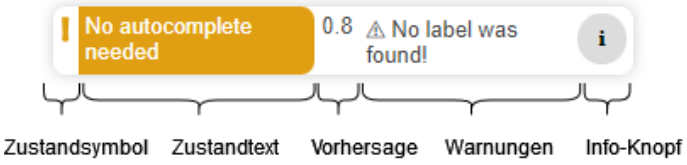


Die Erweiterung verfügt über eine Einstellungs-Seite, über die das Verhalten oder das Aussehen der Erweiterung angepasst und personalisiert werden kann.

Aufbau eines Markers

Ein Marker ist nach demselben Muster aufgebaut. Von links nach rechts finden sich ein Zustandssymbol und eine Zustandsbeschreibung. Diese Elemente sind je nach Analyseergebnis farbcodiert. Der Zustandstext gibt entweder den vorhandenen Autocomplete-Wert wieder, oder, falls dieser fehlt, eine Einschätzung, basierend auf dem Klassifizierungs-Ergebnis. Rechts davon wird der Zuversichtswert der Vorhersage angezeigt, gegebenenfalls mit ermitteltem Autocomplete-Wert. Danach folgen noch optionale Warnmeldungen und ein Info-Knopf zum Anzeigen aller Vorhersagen.

Abbildung 3.3: Die Regionen eines Markers, hier mit Beispielwerten



Marker Zustände

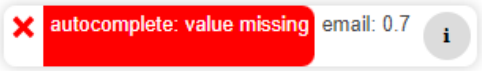
Die automatisch generierten Marker können die Zustände “Richtig”, “Falsch” und “Info” annehmen. Richtige Marker werden vergeben, wenn entweder der vorhandene autocomplete-Wert mit der Vorhersage des Matching-Algorithmus übereinstimmt (und diese über dem Entscheidungs-Schwellwert liegt) oder kein Autocomplete-Attribut gesetzt ist und auch keine Vorhersage mit genügend hohem Zuversichtswert ermittelt wurde. Hat ein autocomplete-Attribut mehrere Werte[1] genügt eine Übereinstimmung mit einem dieser Werte für die Vergabe des Richtig-Zustands.[2] Wie in Abbildung 3.4 erkennbar, wird die Korrektheit nicht nur über ein Häkchen-Symbol, sondern auch durch eine grüne Einfärbung symbolisiert. Der Zuversichtswert der höchstbewerteten Vorhersage wird ebenfalls angezeigt, um die automatisch getroffene Entscheidung zu belegen. Ist kein Autocomplete-Attribut nötig, beträgt der angezeigte Zuversichtswert $1 - maxZuversicht$.

Abbildung 3.4: Beispiele für richtige Marker

[1] durch Leerzeichen separierte Wörter
[2] Beispiel: autocomplete=“email hallo”, Vorhersage ist “email”, ergibt den Richtig-Zustand

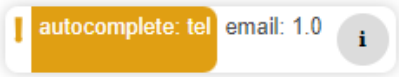
Der Falsch-Zustand wird nur vergeben, wenn für ein Eingabefeld ein Wert über der Entscheidungs-Schwelle vorhergesagt wurde, das Feld allerdings kein Autocomplete-Attribut besitzt. In diesem Fall kann davon ausgegangen werden, dass an dieser Stelle das Attribut benötigt wird. Der vorhergesagte Autocomplete-Wert wird samt zugehöriger Wahrscheinlichkeit im Marker angezeigt.

Abbildung 3.5: Ein Beispiel für einen Fehler-Marker



Hat ein Eingabefeld einen anderen Wert als die Vorhersage des Matching-Algorithmus, wird anstatt des Falsch-Zustandes der Info-Zustand vergeben. In diesem Fall muss der Tester oder die Testerin eine eigenständige Entscheidung treffen, welcher der Werte korrekt ist. Ein Info-Zustand vermittelt eher den Eindruck, dass es zusätzlicher Interaktion bedarf, als ein Falsch-Zustand, der als endgültig empfunden wird.

Abbildung 3.6: Ein Info-Marker mit unterschiedlichen Autocomplete-Werten

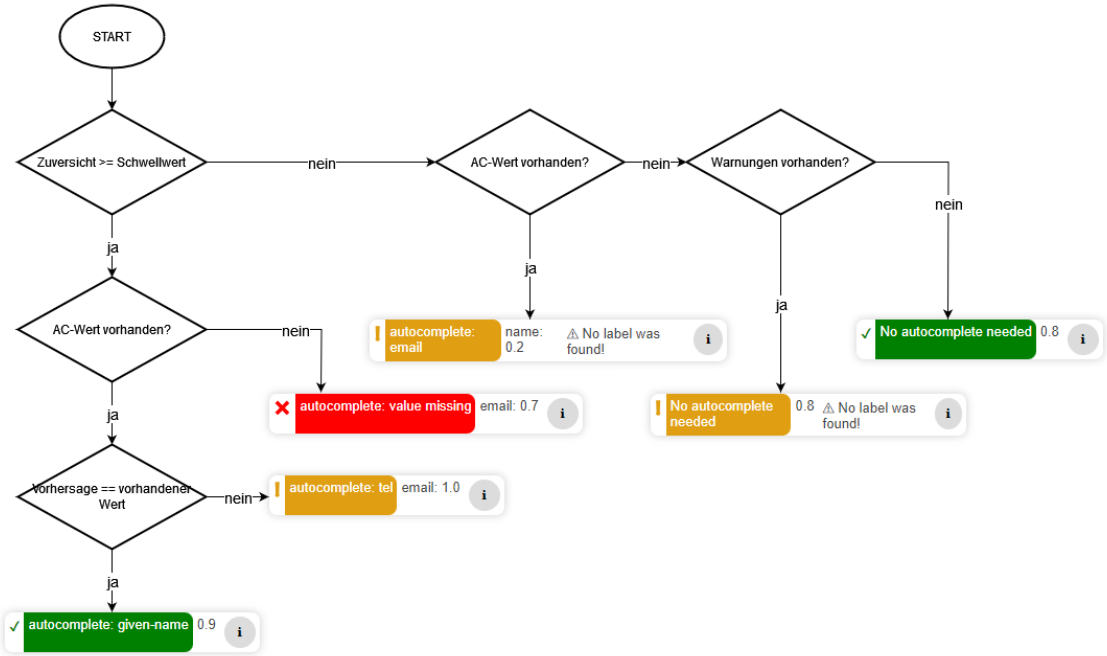


Der Info-Zustand wird auch verwendet, wenn keine Vorhersage über dem Schwellwert ermittelt wurde, aber ein autocomplete-Wert im Feld vorhanden ist, oder wenn kein autocomplete-Attribut gesetzt ist, keine Vorhersage getroffen wurde, aber Warnmeldungen existieren.

Abbildung 3.7: Beispiele für mögliche Info-Marker

Die folgende Abbildung 3.9 visualisiert noch einmal, welche Marker wann vergeben werden.

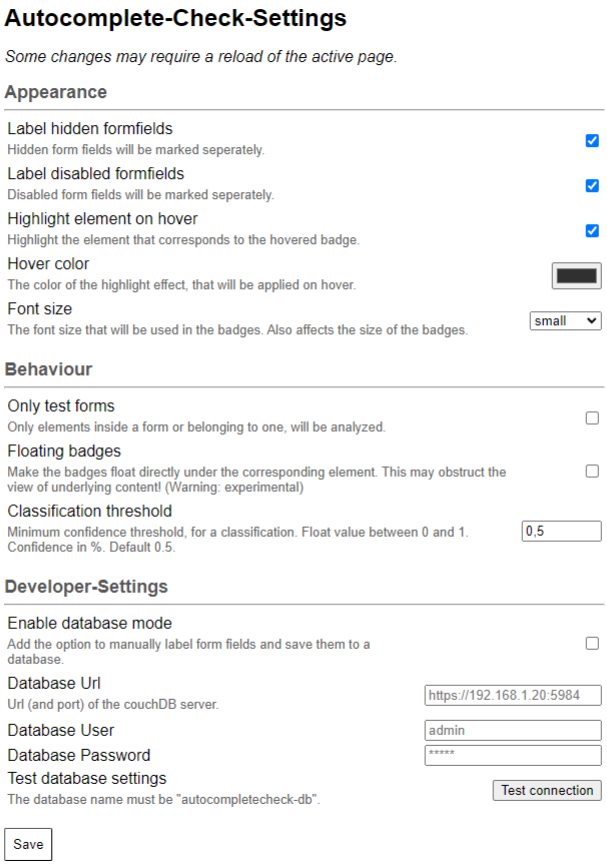
Abbildung 3.8: Der Entscheidungsprozess, welcher Info-Zustand vergeben werden soll, mit Beispielmarkern



Einstellungs-Seite

Das Autocomplete-Check-Plugin hat eine eigene Einstellungs-Seite, über die Anpassungen an der Erweiterung vorgenommen werden können. Diese kann in der Detailansicht zum Autocomplete-Check-Plugin über den Eintrag “Optionen” aufgerufen werden oder durch Rechtsklick auf das Plugin-Icon, wo sich ebenfalls ein Link mit dem gleichen Namen befindet.

Abbildung 3.9: Die Einstellungs-Seite des Autocomplete-Check-Plugins



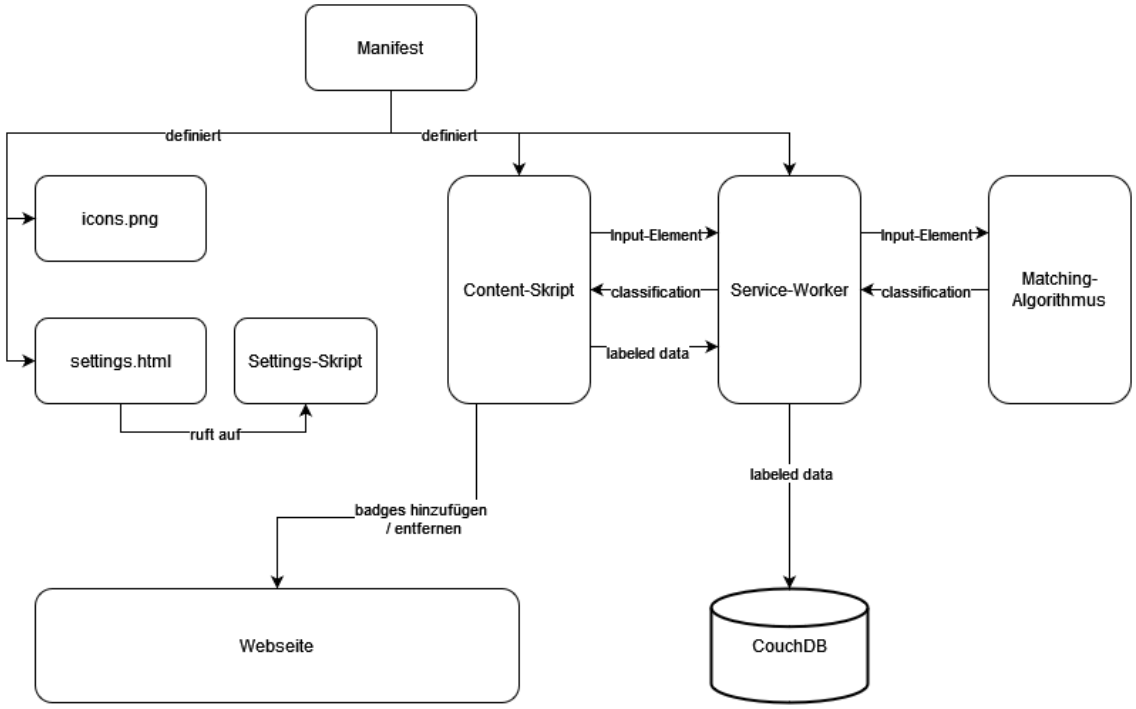
Dort kann die Anzeige des Plugins verändert werden, aber auch das Verhalten in Bezug auf das Prüfen von autocomplete-Attributen. Beispielsweise kann festgelegt werden, dass nur Eingabe-Elemente die zu einem Formular gehören, geprüft und markiert werden sollen. Des Weiteren kann zum Beispiel auch der Klassifizierungs-Schwellwert festgelegt werden, ab dem eine Vorhersage des Matching-Algorithmus als positiv aufgefasst wird.

Nicht zuletzt können der “database mode” aktiviert werden und die Zugangsdaten der Datenbank angegeben werden.

3.2 Aufbau und Architektur

Das Autocomplete-Check-Plugin ist in die Plugin-typischen Skripte pro Bereich gegliedert. Der Matching-Algorithmus ist eine eigenständige Komponente, die keinen Zugriff auf den DOM hat und lediglich ein Objekt, welches das zu analysierende Feld beschreibt, übergeben bekommt.

Abbildung 3.10: Aufbau des Autocomplete-Check-Plugins



Auf ein PopUp wurde in der finalen Version des Plugins verzichtet, da eine Aktivierung des Plugins, zum Beispiel über einen Knopf im PopUp, eine zusätzliche Aktion erfordert hätte. Solange über das PopUp nicht auch kurzfristige Einstellungen geändert werden können, bietet das PopUp keinen Mehrwert und wurde somit deaktiviert.

Der Service-Worker steuert das gesamte Plugin. Er detektiert Klicks auf das Plugin-Icon und fordert darauf das Content-Skript auf, die Markierungen zu erstellen. Die geparsten Matching-Items[3] werden von dort aus an den Service-Worker übergeben, der diese dem Matching-Algorithmus vermittelt und auch schließlich die fertigen Vorhersagen an das Content-Script sendet.

Damit die Datenbank Verbindung nicht im Kontext der überprüften Webseite aufgebaut wird, werden die gelabelten Daten für die Datenbank ebenfalls an den Service-Worker übergeben, der diese an die Datenbank schickt.

[3]siehe Kapitel 3.3

3.3 Matching-Algorithmus

Der Matching-Algorithmus nutzt Heuristiken um ein Feld schnell, basierend auf Erfahrungswerten zu klassifizieren. Das Ergebnis ist allerdings nicht immer richtig, der Algorithmus gibt hierfür Zuversichtswerte zwischen null und eins an, die angeben wie zuversichtlich der Algorithmus ist, dass das analysierte Feld dieser Klasse angehört. Der Matching-Algorithmus ermittelt für ein Eingabefeld von Typ Input-Element, Select-Element oder Textarea-Element passende autocomplete-Werte. Im Folgenden werden die möglichen autocomplete-Werten als autocomplete-Klassen bezeichnet. Als Ergebnis gibt der Algorithmus eine Liste von infrage kommenden autocomplete-Klassen zurück. Jede Klasse hat einen entsprechenden Wahrscheinlichkeitswert.

Als Eingabe erhält der Algorithmus ein sogenanntes "Matching-Item". Dieses Objekt enthält bereits alle geparsten Werte, die das entsprechende Eingabefeld betreffen und die zur Klassifikation notwendig sind. Auch wenn bislang nur der Label-Text, der Placeholder-Text, der Feld-Typ, der Input-Typ, der Form-Typ[4], der Feld-Name und die Feld-Id als Grundlage zur Vorhersage genutzt werden, ist das Matching-Item deutlich ausführlicher, sodass in Zukunft eine detailliertere und präzisere Vorhersage auf dieser Grundlage einfacher zu implementieren ist. Für jeden Eingabewert wird nun ein eigener Test durchgeführt.

3.3.1 Matching-Tests

Aktuell gibt es sieben verschiedene Matching-Tests, die jeweils eine Eigenschaft des zu analysierenden Felds als Testgrundlage verwenden: "matchByLabel", "matchByPlaceholder", "matchByFieldType", "matchByInputType", "matchByFormType", "matchByName" und "matchById". Jeder Test gibt eine Liste an denjenigen Klassen samt Wahrscheinlichkeit zurück, die beim jeweiligen Test für das getestete Eingabefeld infrage kommen. Ein Matching-Testergebnis muss nicht zwangsweise so lang sein wie die Anzahl an möglichen Klassen.

inklusive und exklusive Tests

Ob Testklassen, die nicht im Matching-Testergebnis auftauchen, das Ergebnis negativ beeinflussen sollen, kann über die "matchingClassesInfluence.json"-Datei bestimmt werden. Jeder Test kann darin wahlweise als "inklusiv" oder "exklusiv" markiert werden. Bei einem inklusiven Test werden dem individuellen Test alle Klassen, die im Einzel-Testergebnis nicht ermittelt wurden und somit keinen Zuversichtswert haben, mit der Wahrscheinlichkeit 0% eingestuft. Exklusive-Tests füllen ihre Ergebnisklassen nicht entsprechend auf.

[4] Anmeldeformular oder Registrierungsformular

matchByLabel

Die matchByLabel-Funktion ist ein inklusiver Test und der am höchsten gewichtete Test[5]. Hierbei wird der Label-Text mit Keyword-Listen "gematcht". Diese sind sowohl auf deutsch als auch auf englisch und enthalten pro Klasse meist mehrere Begriffe, die mögliche Labelinhalte oder Teile von Labeln sein können. Pro Klasse werden nun der Label-Text und jeweils die englischen und deutschen Listen an die Substring-Suchfunktion[6] übergeben. Danach werden die Ergebnisse zusammengeführt. Ein vollständiges Match übertrumpft dabei ein teilweises Match der selben Klasse in der anderen Sprache. Kollidieren zwei Teilweise-Matches wird der Durchschnittswert gebildet.

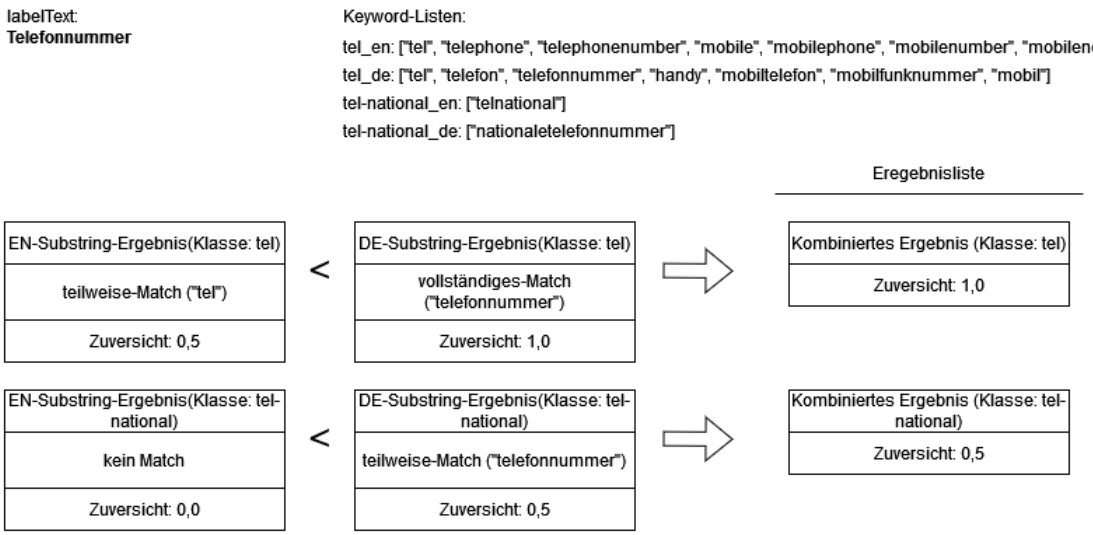
Abbildung 3.11: Auszug aus der deutschen Label-Keyword-Liste zu namensbezogenen Klassen

```
"byLabel_DE":{
    "1": ["name", "vorundnachname"],
    "2": ["anrede", "titel", "prefix"],
    "3": ["vorname", "rufname"],
    "4": ["zweitname", "zwischenname", "mittelname"],
    "5": ["nachname", "familienname"],
    "6": ["namenszusatz", "zusatz", "suffix"],
    "7": ["alias", "künstlername", "spitzname"],
    "8": ["beruf", "titel", "jobbezeichnung"],
```

Abbildung 3.12 soll diesen Prozess veranschaulichen. Beispielhaft wird hier auf den Label-Text "Telefonnummer" gematcht. In der Grafik wird die Substring-Suchfunktion nur an zwei Klassen durchgeführt, "tel" und "tel-national". Normalerweise wird dies natürlich für alle Klassen durchgeführt und dient hier nur der Übersichtlichkeit. Hier enthält die Ergebnisliste nun zwei Einträge, jeweils mit einer Klasse und einem vorläufigen Zuversichtswert.

[5] siehe 3.3.3 Test-Gewichtung
[6] siehe 3.3.2 Substring-Suchfunktion

Abbildung 3.12: Vereinfachte Vorgehensweise der matchByLabel-Funktion am Beispiel von zwei Klassen



matchByPlaceholder

Die Funktionsweise der matchByPlaceholder-Funktion ist identisch zu "matchByLabel", als Eingabewert wird hier jedoch der placeholder-Text verwendet und gegen die Label-Keyword-Listen gematcht. Außerdem ist matchByPlaceholder eine exklusive Funktion.

matchByFieldType

In dieser Funktion wird das aktuell überprüfte Feld auf seinen HTML-Typ getestet. In der "autocomplete-dict.json"-Datei ist für jeden der drei möglichen Feld-Typen (Input-Element, Select-Element und Textarea-Element) definiert, welche Klassen jeweils als solches Feld implementierbar wären. Felder, bei denen mehr als 30 Klassen möglich sind, also beim Input-Element, wird ein Zuversichtswert von 0,75 vergeben. Die beiden anderen Felder vergeben einen Zuversichtswert von 1,0. MatchByFieldType ist eine inklusive Funktion.

matchByInputType

MatchByInputType funktioniert genauso wie "matchByFieldType", matcht jedoch auf die möglichen Input-Typen pro Input-Feld, etwa "text", "email" oder "number". Ist das getestete Feld kein Input-Type, wird der Test abgebrochen und leer zurück gegeben. Diese Matching-Funktion ist ebenfalls ein inklusiver Test.

matchByFormType

Die matchByFormType-Funktion testet die im aktuellen Formular vorhandenen Überschriften mithilfe der Substring-Suchfunktion auf Matches mit Keyword-Listen zu den Wörtern "login" und "signup", jeweils auf deutsch und englisch. Diese Funktion soll ermitteln, ob der Nutzer oder die Nutzerin sich gerade anmeldet, oder einen neuen Account anlegt. Diese Information hilft bei der Unterscheidun der Klassen "current-password" und "new-password". Basierend auf der Art der Substring-Matches und der Anzahl an Treffern für jeweils "login" und "signup" wird einem der beiden Passwort-Klassen die Wahrscheinlichkeit 1.0 und der anderen 0.5 zugewiesen. Dieser Test ist ein exklusiver Test.

matchByName und matchById

Diese beiden Funktionen sind in ihrer Funktion identisch zu "matchByLabel" und prüfen auch mit den Label-Keyword-Listen. Der Eingabewert ist jedoch der Feld-Name oder die Feld-Id. Diese Tests sind beide exklusiv.

3.3.2 Substring-Suchfunktion

Die Substring-Suchfunktion verlangt als Parameter eine Liste an Keywörtern, sowie einen Suchbegriff gegen den gematcht werden soll. Mögliche Ergebnisse sind ein Vollständiges-Match, ein Teilweise-Match, ein Rückwärts-Teilweises-Match oder kein Match. Bei allen durchgeführten Suchen werden die verglichenen Worte von Sonderzeichen bereinigt und klein geschrieben. Zunächst wird überprüft, ob der Suchbegriff komplett in der Liste vorhanden ist. Falls ja, wird "vollständig" zurückgegeben. Nun wird der Suchbegriff, falls möglich, an Leerzeichen aufgespalten. Dies passiert etwa, wenn der Suchbegriff aus mehreren Wörtern besteht[7]. Jeder Suchbegriff-Teil wird nun gegen die Listen gematcht. Um eine hinreichende Genauigkeit zu gewährleisten, müssen Suchbegriff-Teile mindestens drei Zeichen lang sein, ansonsten wird die Suche abgebrochen. Ist ein Suchbegriff-Teil ein Substring eines Keywortes aus der Liste, wird "teilweise" zurückgegeben. Ist ein Keywort aus der Liste ein Substring eines Suchbegriff-Teils, ergibt "rückwärts-teilweise". Bei keinem Treffer wird "keins" zurückgegeben.

3.3.3 Test-Gewichtung

Die gesammelten Resultate der Matching-Tests werden pro Test noch unterschiedlich gewichtet. In der "matchingClassesInfluence.json"-Datei ist jedem Test ein Gewicht als Zahl größer Null zugewiesen. Pro autocomplete-Klasse werden nun mit allen Testergebnissen eben jener Klasse, folgende Berechnungen durchgeführt, wobei n die Anzahl an individuellen Testergebnissen in der aktuellen Klasse ist:

$$\text{gewichteteSummeZuversichten} = \sum_{i=1}^{n} (\text{Zuversicht}_i \times \text{Gewicht}_i)$$

$$\text{SummeGewichte} = \sum_{i=1}^{n} \text{Zuversicht}_i$$

$$\text{GesamtZuversichtGewichtet} = \frac{\text{gewichteteSummeZuversichten}}{\text{SummeGewichte}}$$

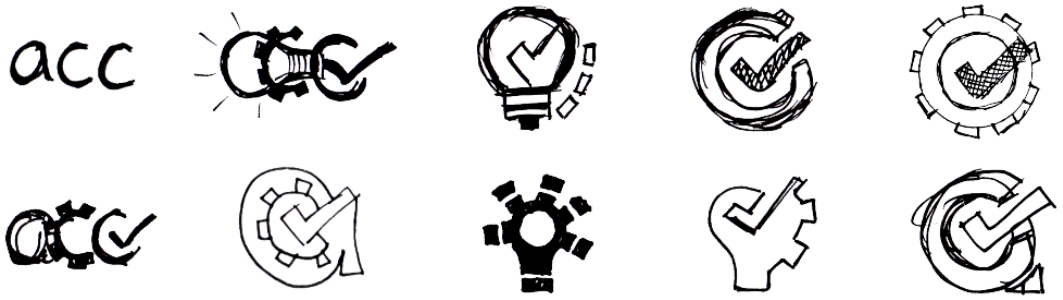
[7] Beispiel: "Straße und Hausnummer" würde in drei Wörter aufgespalten werden.

Der Wert aller “GesamtZuversichtGewichtet” wird jeweils mit der zugehörigen Klasse an das Content-Skript übergeben.

3.4 Logo-Design

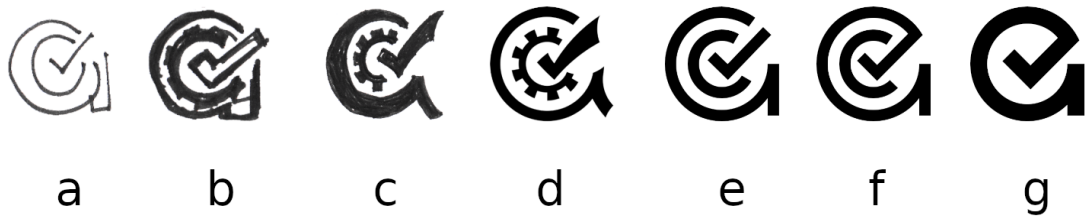
Um den Wiedererkennungswert des Autocomplete-Check-Plugins zu steigern, damit das Icon besser unter mehreren Plugins schnell gefunden werden kann, wurde ein eigenes Logo für die Erweiterung entworfen. Als Leitideen wurden die Anfangsbuchstaben “ACC”, eine Glühbirne für die Pseudointelligenz des Matching-Algorithmus, sowie ein Zahnrad für die Automatisierung und ein Häkchen für das “Check” im Namen des Plugins. Außerdem wird ein Häkchen mit positiven Eigenschaften, wie “korrekt” oder “richtig” assoziiert. Viele der Designideen waren jedoch zu detailliert und filigran, um als Logo und Icon verwendet zu werden.

Abbildung 3.13: Unterschiedliche Designkonzepte für ein Logo



Letztendlich haben wir uns für ein Design entschieden, bei dem ein kleingeschriebenes A ein kleingeschriebenes C umschließt. Ein Häkchen ist mittig über dem C platziert, sodass es so aussieht, als ob das Häkchen ein rundes Eingabefeld “ankreuzt”. Diese Grundidee wurde mehrfach angepasst und weiterentwickelt. Im ersten konkreten Entwurf (siehe Abbildung 3.15, Figur c) wurde ein handgeschriebener Style gewählt. Das C stellt ein halbes Zahnrad dar. Die computergezeichnete Version d offenbarte allerdings Schwächen des Designs: Design d ist zu kleinteilig und verwendet zu viele unterschiedliche Strichstärken. In den Iterationen wurden daraufhin einheitliche Linienbreiten sowie ein symetrischer Ansatz ausprobiert. Wenn ein Browser-Plugin an die Schnellzugriffsleiste angeheftet wird, ist das Logo dabei effektiv meist nur wenige Millimeter groß. In dieser Größe traten bei Iteration e und f visuelle Artefakte auf, sodass das Logo erneut vereinfacht und eine noch breitere Linienstärke verwendet wurde. In der finalen Version g wird das C nur noch negativ durch den Freiraum im Logo angedeutet. Um das Logo weiter zu vereinfachen, wurde das Häkchen an die Umrandungslinie angeschlossen.

Abbildung 3.14: Iterationen des Logos



Für das Plugin-Icon haben wir eine Version des Logos gewählt, bei dem der Hintergrund gefüllt ist und das Logo weiß bleibt. Durch eine unterschiedliche Färbung kann angezeigt werden, ob die Erweiterung aktiv ist. Hellgrau steht für ein inaktives, Grün für ein aktives Plugin. Die Farben wurden so gewählt, dass bei allen Arten von Farbfehlsichtigkeiten der Kontrast zur Unterscheidung ausreicht. Die massivere Version des Logos sorgt für eine bessere Sichtbarkeit des Plugin-Status, da so mehr Fläche gefärbt ist.

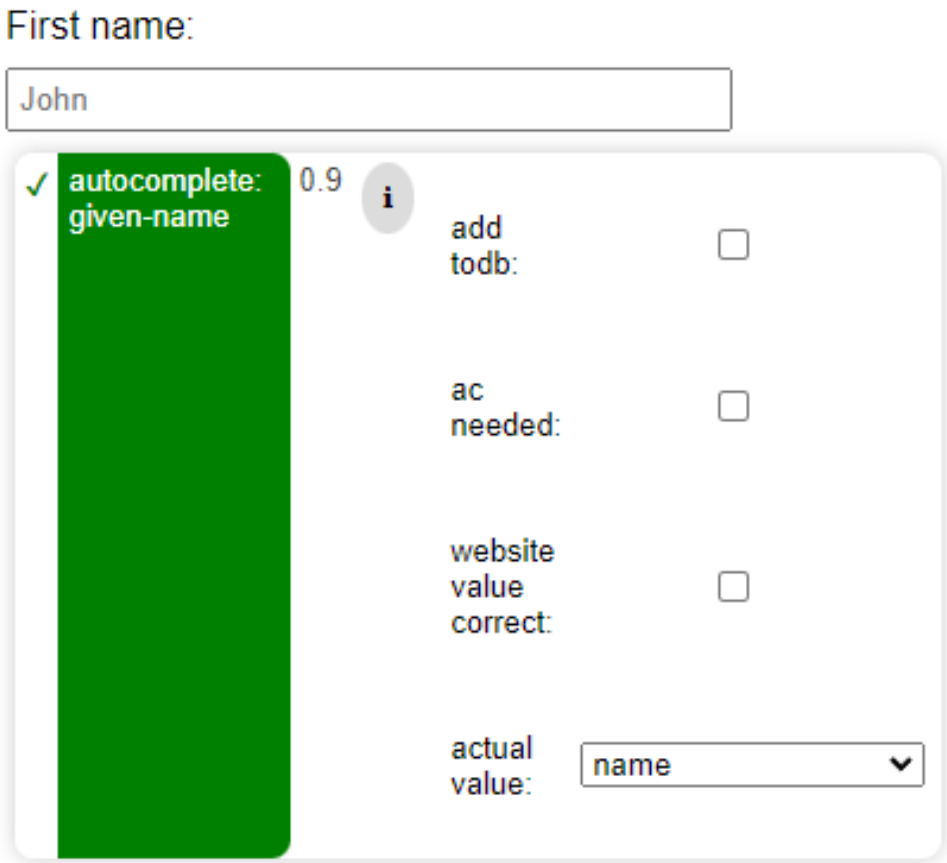
Abbildung 3.15: Die finalen Logos und Plugin-Icons

3.5 Datenbank-Anbindung

In den Einstellungen des Autocomplete-Check-Plugins lässt sich der sogenannte “database mode” aktivieren. Dieser ergänzt die Erweiterung um die Funktionalität, die für das Plugin relevanten Eingabefelder zusätzlich händisch zu klassifizieren und an eine Datenbank zu senden. Das wiederum ermöglicht das Erstellen eines autocomplete-bezogenen Datensatzes, anhand dessen man eine Auswertung über die Verwendung des autocomplete-Tags im Internet vornehmen könnte. Der Datensatz würde sich aber auch als Grundlage für Ansätze eignen, die auf maschinellem Lernen basieren. Klassifikationsverfahren wie zum Beispiel “Random Forrest” benötigen häufig eine große Anzahl an Datenpunkten sowie einen möglichst großen Datensatz. Datenpunkte sind hierbei beschreibende Informationen zum Eingabefeld wie Name, Id, und Label, aber auch Parameter, die bislang keinen Einfluss in den Matching-Algorithmus finden, wie beispielsweise das “maxlength”-Attribut. Das Autocomplete-Check-Plugin soll die Erstellung eines solchen Datensatzes vereinfachen, indem alle Datenpunkte maschinell

ermittelt werden. Das händische Labeln wird durch den Vorschlag des Matching-Algorithmus unterstützt, sodass schnellere Entscheidungen möglich sein sollen.

Abbildung 3.16: Ein Marker mit Label-Möglichkeit für die Datenbank



Die Erweiterung fügt hierzu den Klassifikations-Markern Checkboxen hinzu, mit denen festgelegt werden kann, ob der autocomplete-Wert an dieser Stelle korrekt ist, korrekt verwendet wurde, welcher Wert eigentlich richtig ist und ob dieses Feld zur Datenbank hinzugefügt werden soll. In der linken oberen Ecke der Webseite wird ein Knopf hinzugefügt, der die Webseite überlagert und bei Aktivierung alle ausgewählten Eingabefelder mitsamt zusätzlicher Metadaten zur aktuellen Webseite in eine Datenbank einträgt. In den Optionen können hierfür die IP-Adresse der Datenbank mit Port sowie Benutzername und Passwort hinterlegt werden. Als Datenbank haben wir “CouchDB” gewählt, eine dokumentenbasierte Datenbank, die von der Apache Software Foundation unter der Apache Lizenz entwickelt wird.

4 Methodik

Im Folgenden wird das Vorgehen bei dieser Arbeit erläutert. Dies bezieht sich sowohl auf die Programmierung als auch auf die Recherche zum Theorie-Teil, sowie die Auswertung und Validierung. Die gewählten Methoden entsprechen üblichen Standards an Vorgehensweisen.

4.1 Verwendete Technologien

Die verwendeten Software-Bibliotheken sind aktuell jeweils die populärsten ihrer Art für diesen Anwendungszweck. Aus diesem Grund ist eine umfangreiche Primär- und Sekundär-Literatur vorhanden[20][26][27][18]. Dies ist bei Fehlern oder offenen Fragen in der Entwicklung äußerst praktisch und hilfreich. Erwähnenswert ist außerdem, dass alle verwendeten Bibliotheken sogenannte "development dependencies" sind und somit nur während der Entwicklung verwendet werden. Der Build des Autocomplete-Check-Plugins kommt gänzlich ohne externe Bibliotheken aus, was die Sotfware-Abhängigkeiten minimiert und Konflikte beim Updateprozess, zumindest durch Bibliotheken, verhindert.

TypeScript

Als Programmiersprache haben wir TypeScript gewählt, da die Sprache mit ihrer strengen Typisierung viele Software-Fehler bereits in der Entwicklungsumgebung bemängelt, sodass diese nicht erst durch Zufall beim Ausführen gefunden und nachvollzogen werden müssen. Außerdem konnten viele logikbasierte Programmierfehler, die in JavaScript möglich gewesen wären, erst gar nicht entstehen. In der Tat traten während der Entwicklung des Plugins zur Laufzeit keine kritischen Fehler auf, die die Ausführung des Plugins abbrachen. Die wenigen Laufzeit-Fehler entstanden alle im Zusammenhang mit dem Networking der Datenbank-Anbindung. Durch die Robustheit von JavaScript werden diese zwar geloggt, das Plugin läuft aber unbemerkt weiter.

Jest

Mit dem Test-Framework Jest konnten einige Softwarefehler in der Substring-Such-Funktion identifiziert werden, da dort nicht alle möglichen Fälle bedacht wurden. Außerdem wurde Jest verwendet, um Anpassungen an den Gewichten der Matching-Tests automatisiert zu prüfen, was eine enorme Zeitersparnis im Entwicklungsprozess war.

4.2 Testseite

Zur Validierung des Plugins während seiner Entwicklung und für Tests mit vergleichbaren Anwendungen haben wir eine HTML-Testseite erstellt, die neben einem gängigen Formular auch Sonderfälle beinhaltet, beispielsweise Inputelemente vom Typ "hidden". Ziel der Testwebseite war es, alle erdenklich möglichen Szenarien im Bezug auf das autocomplete-Attribut abzudecken, um bei der Validierung einen möglichst ausführlichen Test zu bieten.

4.3 Analyse der autofill-Funktionalität von Chromium

Um die genaue Funktionsweise der autofill-Funktionalität von Chromium erklären zu können, haben wir die komponentenspezifische Dokumentation im öffentlichen Git-Repository von Chromium ausgewertet. Da diese jedoch recht allgemein und oberflächlich gehalten ist, wurde zum Verstehen wichtiger Stellen der relevante Source-Code gelesen. Um bestimmte Aussagen der Dokumentation zur Vorgehensweise Chromiums bei der Klassifikation zu überprüfen, haben wir ein Experiment an einer eigens erstellten Testwebseite durchgeführt. Im Experiment wurde der von Google zur Verfügung gestellte Build des Chromium-Projekts – der Chrome-Browser verwendet. Da dieser bei lokal gespeicherten HTML-Seiten die autofill-Funktion nicht anwendet, haben wir auf einem Testserver einen Apache-Webserver eingerichtet, um die entsprechenden HTML-Dateien bereitzustellen und darauf zu testen. In Chrome haben wir dazu eine Beispiel-Adresse und Beispiel-Zahlungsmethoden hinterlegt. Eine Adresse beinhaltet bei Chrome die Felder "Land", "Name", "Organisation", "Adresse"[1], "Postleitzahl", "Stadt", "Telefon" und "Email". Klickt man bei der Testwebseite auf ein passendes Eingabefeld, bietet der Browser an, das gesamte Formular passend auszufüllen. Das tatsächliche Verhalten des Chrome-Browsers wurde bei Abweichungen zur Dokumentation in der Ausführung erwähnt.

[1] Straße und Hausnummer, sowie Zusätze

4.4 Sammlung und Analyse vergleichbarer Anwendungen

Erstellen einer Liste an vergleichbaren Anwendungen

Auf der Suche nach Anwendungen, die auf autocomplete-Attribute prüfen, haben wir zunächst den Chrome-Webstore sowie die Firefox Add-on-Bibliothek durchsucht. Konkreter Fokus wurde auf Entwicklertools und accessibility-bezogene Plugins gelegt. Anhand der Beschreibung wurde entschieden, ob diese Erweiterung in die Vorauswahl aufgenommen werden sollte. Alle Plugins der Vorauswahl wurden daraufhin händisch auf der Test-Webseite ausprobiert und verworfen, sofern keine Unterstützung in Bezug auf das autocomplete-Attribut festgestellt werden konnte. In einer Besprechung mit einem Betreuer wurde der Hinweis auf ein Tool gegeben, das vor einiger Zeit im Internet verfügbar gewesen sein soll. Einziger Hinweis war jedoch ein nicht funktionierender Link. Mithilfe der Wayback-Machine des Internet Archives konnte eine gespeicherte Version der referenzierten Webseite eingesehen werden, worüber schließlich das GitHub-Repository des Autocomplete Favlets ausfindig gemacht werden konnte. Auf GitHub wurden Repositories nach Suchbegriffen wie “autocomplete”, “wcag” und “1.3.5” durchsucht. Dabei konnte das WAccess-Plugin gefunden werden.

Vergleich der Anwendungen

Nach dem Definieren von Mindest- und Idealen-Anforderungen an ein Prüftool, das auf WCAG Kriterium 1.3.5 testet, haben wir basierend darauf Bewertungskategorien erstellt. Zusätzlich wurde der Chrome-Inspektor in die Beurteilung aufgenommen, da dieser als nahezu vollständig ausschließlich händisches Prüftool die untersten Anforderungen verkörpert. Jede Anwendung der erstellten Liste wurde nun manuell in allen Bewertungskategorien getestet und nach Resultat bewertet. Aus dem tabellarischen Vergleich wurden Lücken identifiziert, welche den Bedarf am Autocomplete-Check-Plugin aufzeigen.

4.5 Betreuer-Test

Zur Überprüfung des Autocomplete-Check-Plugins wurden durch die Betreuer dieser Arbeit gegen Ende der Bearbeitungszeit drei Webseiten aus dem öffentlichen Internet vorgegeben. Die Auswahl der Webseiten war im Vornherein nicht bekannt, um zu vermeiden, dass der Matching-Algorithmus bewusst oder unbewusst an den Test-Seiten angepasst oder darauf zugeschnitten wird. Folgende Webseiten wurden von den Betreuern ausgewählt:

- eine Bewerbungsseite auf dem Karriereportal des Landschaftsverband Reinlands (kurz LVR)
- die Registrierungsseite der indischen Dating-Seite Jeevansathi, diese Webseite ist in englischer Sprache
- das Kontaktformular der Hochschule der Medien Stuttgart (kurz HdM)

Mit einem Chrome-Browser, der das Autocomplete-Check-Plugin[2] installiert hatte, haben wir die Test-Seiten, die durch die Betreuer vorgegeben wurden, mit dem Autocomplete-Check-Plugin überprüft. Die visuelle Markierung der Webseiten wurde jeweils als Screenshot zur Auswertung festgehalten. Je Eingabefeld wurde ermittelt, ob die Klassifikation des Matching-Algorithmus dem eigentlich erforderlichen autocomplete-Wert entspricht[3]. Die Ergebnisse wurden in einer Matrixform gesammelt, die zwar an eine Konfusionsmatrix einer binären Klassifikation erinnert, jedoch streng genommen nicht-binär ist, da die Bewertungsklassen der Vorhersage[4] nicht mit denen der tatsächlich erforderlichen Werte übereinstimmen, da beim Vergleich zusätzlich geprüft wurde, ob der Wert der Vorhersage korrekt ist. Der Vergleich ist somit nicht binär. Da ebenfalls zwei Fehlermöglichkeiten existieren, ist die Auswertung zur binären Klassifikation recht ähnlich.

4.6 Open-Source Veröffentlichung

Mit Abgabe dieser Arbeit wird das Autocomplete-Check-Plugin unter der MIT-Lizenz auf Github veröffentlicht[5] und somit der Allgemeinheit zur Verfügung gestellt. Die MIT-Lizenz erlaubt die kommerzielle und private Nutzung sowie die Verbreitung und Bearbeitung des Plugins, solange Lizenz und Copyright erwähnt werden. Die MIT-Lizenz schützt die Autoren, indem jegliche Garantie- und Haftungsansprüche ausgeschlossen werden. Die Veröffentlichung auf GitHub ermöglicht, dass andere Personen über Issues Fehlerberichte und Verbesserungsvorschläge oder eventuell sogar Quellcode-Änderungen per Pull-Request einreichen können. Da das Autocomplete-Check-Plugin erst nach der Fertigstellung dieser Arbeit quellöffentlich gemacht wird, sind die Vorteile einer Open-Source-Anwendung noch nicht in der Entwicklung spürbar gewesen. Die Veröffentlichung soll hier dennoch erwähnt sein, da diese einen zukünftigen Entwicklungsprozess maßgeblich beeinflussen kann. Der Plugin-Stand zum Abschluss dieser Arbeit wird auf GitHub mit dem Release "1.0.0" gekennzeichnet. Noch offene Implementierungs-Ideen wurden den Issues hinzugefügt.

[2] Durchführung am 24.09.2023
[3] Oder dessen Nichtvorhandensein
[4] autocomplete-Attribut benötigt / nicht benötigt
[5] Die URL des öffentlichen Repoistorys lautet: "https://github.com/PhilippRecke/Autocomplete-Check"

5 Ergebnisse und Validierung

5.1 Betreuer-Test

Um die Funktionsweise des Autocomplete-Check-Plugins, insbesondere in Hinblick auf die automatischen Vorhersagen zu testen, wurde ein Test durchgeführt, bei dem das Plugin an vorher unbekannten Webseiten getestet wurde. Die Webseiten wurden durch die Betreuer zur Verfügung gestellt.

5.1.1 Seitenbeschreibungen

Karriereportal des LVR

Zum Testen wurde eine beliebige Stellenausschreibung herangezogen. Der zum Testen vorgegebene Bereich "Persönliche Daten" ist bei jeder Anzeige identisch. Der Abschnitt lässt sich in die Überthemen Personen-, Kontakt- und Adressdaten einteilen. Zusätzlich gibt es Eingabefelder um anzugeben, ob derzeit bereits ein Arbeitsverhältnis mit dem LVR besteht, ob der Bewerber oder die Bewerberin einen PKW-Führerschein besitzt sowie Angaben zur Staatsangehörigkeit und zur "Arbeitssprache". Von den insgesamt 24 sichtbaren Eingabefeldern sind die Hälfte normale Textfelder, dazu kommen einige Select-Elemente sowie drei Radio-Button-Gruppen. An erster Stelle im Formular sind sechs Input-Elemente vom Type "hidden" sowie zwei in der Mitte des Formulars, die für Metadaten verwendet werden und nicht sichtbar sind.

Jeevansathi Registrierung

Zur Registrierung bei Jeevansathi werden Email, Mobiltelefonnummer und ein Passwort verlangt. Dafür wurden ausschließlich Textfelder verwendet, die Mobilnummer hat für die Ländervorwahl ein seperates Feld. Zusätzlich muss ausgewählt werden, für wen das Profil erstellt werden soll, etwa für sich selbst, Verwandte oder Andere. Dieses Element verhält sich wie eine Radio-Button-Gruppe, ist jedoch nicht mit Eingabeelementen implementiert. Außerdem sind zwei Textinputs für Benutzername

und Passwort angelegt, die jedoch als Style-Attribut “display: none” besitzen und somit nicht sichtbar sind. Ebenfalls nicht sichtbar sind zwei Input-Elemente vom Typ “hidden”, die für formularspezifische-Metadaten verwendet werden.

Kontaktformular der HdM

Für eine Kontaktaufnahme zur HdM im entsprechenden Kontaktformular muss eine Email-Adresse in ein Textfeld angegeben werden. Ein Select-Element lässt auswählen, wer kontaktiert werden soll. In eine Textarea lässt sich eine mehrzeilige Nachricht eingeben. Zusätzlich ist ein Google-reCAPTCHA-Element eingebunden, jedoch ist die Checkbox nicht als Input-Element implementiert.

Testergebnisse

In der Analyse der Testergebnisse werden die Eingabefelder beachtet, die für den Nutzer oder die Nutzerin sichtbar sind. In allen Fällen hat das Autocomplete-Check-Plugin nicht sichtbare Felder gefunden und entsprechend markiert. Da hier jedoch keine Vorhersage des autocomplete-Werts nötig ist, werden diese Felder hier nicht berücksichtigt. Die Tests wurden mit den Standardeinstellungen durchgeführt, der Schwellwert bei der Klassifikation liegt somit bei 0,5.

Bewertung der Vorhersagen

Die Autocomplete-Check-Vorhersagen werden anhand eines Verfahrens, das der binären Klassifikation ähnelt, bewertet. Dabei werden den Vorhersagen die tatsächlichen Werte des Feldes gegenübergestellt. Der tatsächliche Wert bezieht sich nicht zwangsweise auf den innerhalb der Webseite gefundenen autocomplete-Wert, sondern auf den, der gemäß WCAG Erfolgskriterium 1.3.5 vorgeschrieben ist. Als Fehler in der Klassifikation gelten Felder, die eine falsche Vorhersage haben, oder keine Vorhersage mit ausreichend hohem Schwellwert besitzen, tatsächlich aber ein autocomplete-Attribut benötigen, oder aber wenn eine positive Vorhersage gemacht wurde, aber kein autocomplete-Wert gefordert ist. Im Folgenden wird eine positive Vorhersage als solche bezeichnet, wenn die Erweiterung eine Vorhersage trifft, die über dem Schwellwert liegt. Ist diese niedriger als der Schwellwert, ist die Vorhersage negativ. Ein positiver tatsächlicher Wert existiert dann, wenn ein Eingabefeld einen autocomplete-Wert benötigt, wenn nicht ist der tatsächliche Wert als negativ zu werten. Da das Autocomplete-Check-Plugin als Ergebniswerte neben richtig und falsch auch einen Info-Zustand hat und somit nicht streng binär klassifiziert, wird hier definiert, wie die Info-Ergebnisse bewertet werden. Diese wäre meistens eigentlich binar[1] klassifizierbar,

[1] In Bezug darauf, ob das das Plugin ein autocomplete-Attribut fordert, oder nciht

jedoch wird der Info-Zustand verwendet um auf Ausnahmefälle hinzuweisen oder um bei Unklarheiten die Aufmerksamkeit des Nutzers oder der Nutzerin zu gewinnen.

Info-Ergebnisse werden als positive Vorhersage gewertet, wenn:

- eine Vorhersage über dem Schwellwert liegt, aber nicht dem angegebenen autocomplete-Wert entspricht, dafür aber dem tatsächlichen Wert.

Info-Ergebnisse werden als negative Vorhersage gewertet, wenn:

- kein autocomplete-Wert vorhanden ist und die Vorhersagen den definierten Schwellwert nicht überschreiten, jedoch Warnmeldungen exisitieren[2], die das Ergebnis beeinflussen könnten.
- die Vorhersage unterhalb des Schwellwerts ist, jedoch ein autocomplete-Wert gesetzt ist.

[2] z.B. Kein Label vorhanden, Element ist hidden

Auswertung LVR

Eingabefeld	AC-Wert	tatsächlicher AC-Wert	Vorhersage	Zuv. (%)
Anrede	honorific-prefix	honorific-prefix	honorific-prefix	0,85
Akadem. Titel	honorific-prefix	honorific-prefix	honorific-prefix	0,7
Vorname	given-name	given-name	given-name	0,96
Nachname	family-name	family-name	family-name	0,96
Geburtsdatum	bday	bday	bday	0,77
Geschlecht	-	sex	sex	0,85
Schw.behinderung ja	-	-	-	0,78
Schw.behinderung nein	-	-	-	0,78
Email	email	email	email	0,87
Telefonnummer	tel	tel	tel	0,87
Buisness Netzwerk	-	-	-	0,69
Mitarbeiter LVR ja	-	-	-	0,78
Mitarbeiter LVR nein	-	-	-	0,78
Personalnr.	-	-	-	0,68
Sprache	language	language	language	0,51
Führerschein ja	-	-	-	0,78
Führerschein nein	-	-	-	0,78
Staatsangehörigkeit	off	-[3]	-	0,69
Str./Hausnr.	address-line1	address-line1	street-address	0,96
Plz.	postal-code	postal-code	postal-code	0,96
Ort	address-level1	address-level2[4]	address-level2	0,95
Land	country-name	country-name	country-name	0,85
Adresszusatz	-	-	honorific-suffix	0,59
Jobportal	-	-	streed-address	0,55

Tabelle 5.1: Ergebnis des LVR-Tests, Stand des Codes vom 24.09.2023

		Vergleich mit tatsächlichem Wert		
		korrekt bestimmt	falsch bestimmt	gesamt
Vorhersage	Positive	12	2	14
	Negative	1	9	10
	gesamt	13	11	24

Tabelle 5.2: Vergleichsmatrix LVR - Vorhersage vs. tatsächliche Werte

Insgesamt bewertet das Autocomplete-Check-Plugin mit nur drei Fehlern 87,5% der Eingabefelder mit einer durchschnittlichen Zuversicht von 81,4% richtig. Die durchschnittliche Zuversicht bei den Fehlern liegt bei 0,7 und damit deutlich über dem Schwellwert von 0,5. Ein Grund dafür ist die Fehlklassifikation des "Straße/Hausnummer"-Feldes. Hier wäre der korrekte Wert "address-line1", jedoch wurde "street-address" vorhergesagt. Beide Werte sind vom Inhalt letztendlich identisch, jedoch wird letzteres ausschließlich für mehrzeilige Eingabefelder, sprich Textareas, verwendet. In der Datengrundlage des Matching-Algorithmus war fälschlicherweise definiert, dass "street-address" auch als normales Input-Element möglich wäre. Die falscherweise bestimmte positiven Vorhersage des Feldes "Adresszusatz" liegt an einem Substring-Match auf die Begriffe "Namenszusatz" und "Zusatz". Bei der zweiten falsch bestimmten positiven Vorhersage wurde das Feld zur Angabe eines Jobportals fälschlicherweise als "street-address" klassifiziert. Grund war ein Substringmatch des Wortes "aus", aus dem Label des Eingabefeldes ("Bitte wählen Sie ein Veröffentlichungsportal aus"), welches mit dem Begriff "Hausnummer" aus der Datengrundlage einen Treffer ergab.

Auswertung Jeevansathi

Alle Felder wurden vom Autocomplete-Check-Plugin falsch bewertet. Die durchschnittliche Zuversicht liegt bei 62,75%. Grund ist, dass alle Label-Elemente kein "for"-Attribut besitzen. Dem autocomplete-Marker wurde jeweils ein Hinweis auf das Fehlen eines zugeordneten Labels hinzugefügt. Erwähnenswert ist jedoch, dass die höchstgewerteten Vorhersagen den tatsächlichen Wert oder einen damit verwandten Wert haben. "Email" und "tel-country-code" wurden jeweils mit 29% ermittelt, statt "tel-local" wurde mit 29% "tel" klassifiziert und "current-password" statt "new-password" mit 31% Wahrscheinlichkeit.

Eingabefeld	AC-Wert	tatsächlicher AC-Wert	Vorhersage	Zuv. (%)
Email	nope	email	-	0,61
Mobile No. (ISD)	nope	tel-country-code	-	0,61
Mobile No.	nope	tel-local	-	0,61
New Password	-	new-password	-	0,68

Tabelle 5.3: Ergebnis des Jeevansathi-Tests, Stand des Codes vom 24.09.2023

		Vergleich mit tatsächlichem Wert		
		korrekt bestimmt	falsch bestimmt	gesamt
Vorhersage	Positive	0	0	0
	Negative	4	0	4
	gesamt	4	0	4

Tabelle 5.4: Vergleichsmatrix Jeevansathi - Vorhersage vs. tatsächliche Werte

Eingabefeld	AC-Wert	tatsächlicher AC-Wert	Vorhersage	Zuv. (%)
Email	-	email	email	0,98
Kontaktstelle	-	-	-	0,86
Frage	-	-	-	0,86
reCaptcha[5]	-	-	-	0,86
Submit-Button[6]	-	-	-	0,89

Tabelle 5.5: Ergebnis des HdM-Tests, Stand des Codes vom 24.09.2023

Auswertung HdM

Das Kontaktformular des HdM wurde mit einer durchschnittlichen Zuversichtlichkeit von 88,82% durchweg korrekt bewertet. Zum Zeitpunkt des Tests wurden Elemente, die das Style-Attribut “hidden” besitzen, noch nicht ignoriert beziehungsweise extra gekennzeichnet.

[3] Autocomplete-Check kann kein “off” vorschlagen, stattdessen leere Werte bei Unterschreitung des Schwellwerts

[4] address-level1 wird als höchste Verwaltungsebene definiert. Beispiele in der WHATWG-Dokumentation sind die US-Staaten, oder Schweizer Kantone. In Deutschland entspräche das den Bundesländern, jedoch werden diese bei Adressangaben oft ausgelassen.

[5] Zum Test-Zeitpunkt wurden Elemente mit “style=hidden” noch nicht ignoriert.

[6] Zum Test-Zeitpunkt wurden Submit-Buttons noch nicht ignoriert.

		Vergleich mit tatsächlichem Wert		
		korrekt bestimmt	falsch bestimmt	gesamt
Vorhersage	Positive	1	0	1
	Negative	0	4	4
	gesamt	1	4	5

Tabelle 5.6: Vergleichsmatrix HdM - Vorhersage vs. tatsächliche Werte

		Tatsächlicher Wert		
		Positive	Negative	gesamt
Vorhersage	Positive	13	2	15
	Negative	0	4	4
	gesamt	1	4	5

Tabelle 5.7: Konfusionsmatrix gesamter Betreuer-Test - Vorhersage vs. tatsächliche Werte

5.2 Analyse der Klassifikations-Ergebnisse

Insgesamt wurden mit sieben fehlerhaft klassifizierten Feldern 21,21% der gesamten Eingabefelder aller Test-Webseiten falsch klassifiziert. Dass für Elemente keine Werte (überhalb des Schwellwerts) bestimmt wurden, trat nur dann auf, wenn keine Label-Elemente vorhanden waren, was mit einer entsprechenden Warnmeldung angemerkt wurde. Häufige Fehlerquellen sind ungenaue Substring-Vergleiche, fehlende Label-Elemente und verwandte autocomplete-Werte, die anstatt des richtigen Wertes vorhergesagt wurden.

Eine Verbesserungsmöglichkeit wäre der Umstieg vom Keyword-basierten Wortabgleich auf ein Regex-basiertes System. Mithilfe von Regex-Ausdrücken können auch Wildcard-Zeichen in das Suchpattern aufgenommen werden. Möglich wäre dadurch eine breitere Abdeckung an Suchbegriffen bei einer höheren Spezifität. Um das Problem nicht zugeordneter Label zu adressieren, sind mehrere Ansätze möglich. Denkbar wäre, im Parent-Element des Eingabefelds nach Labeln zu suchen. Sofern nur ein Label und kein weiteres Eingabefeld vorhanden ist, kann man mit hoher Wahrscheinlichkeit das Label dem Feld zuordnen. Dieser Schritt könnte rekursiv über die jeweiligen Parent-Elemente erfolgen, mit abnehmender Wahrscheinlichkeit für eine korrekte Zuordnung, je höher man sich im DOM bewegt. Außerdem könnten Labels anhand eines sich ähnelnden Namens oder der Id ausfindig gemacht werden. Sollte sich das Eingabeelement in einer Tabelle befinden, könnte in benachbarten Zellen nach Label-Elementen gesucht werden. Für verwandte autocomplete-Werte (Beispiel: Klassifizierung “cc-given-name” statt “given-name”) muss das jeweilige Eingabefeld im Kontext des gesamten Formulars oder der gesamten Webseite betrachtet werden. Bislang analysiert und klassifiziert das Autocomplete-Check-Plugin jedes Eingabeelement für sich, ohne die umliegenden

HTML-Elemente zu beachten. Die Genauigkeit mit der Elemente, die häufig beieinander und oft in der selben Reihenfolge sind, klassifiziert werden, könnte so deutlich erhöht werden. Gruppierungen wie Telefondaten, Adressdaten oder Zahlungsinformationen könnten davon profitieren. Dazu ist es nötig, einen Nachbetrachtungsschritt einzufügen, sodass nach der initialen Klassifizierung aller Eingabefelder die Felder untereinander abgeglichen werden und ihre Vorhersage gegebenenfalls entsprechend angepasst werden könnte. Außerdem wäre es sinnvoll, die Position der Eingabeelemente in einem Formular oder auf einer Webseite mit zu beachten. Ein Eingabefeld für den Namen ist in der Regel höher oder früher im Formular als zum Beispiel der Name auf der Kreditkarte.

5.3 Sonstige Ergebnisse aus dem Betreuer-Test

Beim Durchführen des Betreuer-Tests wurden außerdem bestimmte Limitationen des Autocomplete-Check-Plugins deutlich. Beispielsweise hat das Jeevansathi-Formular Eingabefelder, die erst auf menschlichen Input aufklappen oder sichtbar gemacht werden. Die versteckten Felder werden zwar gefunden und klassifiziert, bleiben aber ebenfalls unsichtbar, bis das Feld durch Mausklick aufgefächert wird. Hilfreich wäre es, außerhalb des oberste Parent-Elements, welches unsichtbar ist, einen Hinweismarker auf die Anzahl an versteckten Eingabefeldern einzublenden. Beim Testen der HdM-Webseite fiel auf, dass die Tabelle zum Anzeigen der individuellen Vorhersagen pro Feld unzureichend strikte CSS-Klassen hatte, ein paar Regeln der HdM-Webseite übernahm und so mit übergroßen Paddings und Margins unleserlich wurde. Nach dem Betreuer-Test wurden die CSS-Regeln des Content-Skripts entsprechend verschärft, sodass die erweiterungseigenen Regeln die der Webseiten überschreiben.

6 Zusammenfassung und Ausblick

Das WCAG 2.1 Erfolgskriterium 1.3.5 schreibt Eingabefeldern, die Nutzerdaten betreffen, bestimmte Werte vor, sodass eine maschinelle Bestimmung des Eingabezwecks möglich ist. Der Prüfschritt zum Überprüfen auf die Verwendung solcher Werte bei personenbezogenen Daten, ist von Konformitätsstufe AA und somit ein essenzieller Test, beim Überprüfen von Webseiten auf digitale Barrierefreiheit.

In dieser Arbeit wird das Erfolgskriterium 1.3.5 näher beleuchtet und analysiert. Bestehende Anwendungen, die auf das Kriterium prüfen, werden in einer Auflistung gesammelt und untereinander verglichen. Daraus ergibt sich, dass die vorhandenen Anwendungen unzureichend in ihrem Funktionsumfang und teils umständlich in ihrer Bedienung sind. Basierend auf den Defiziten der bestehenden Anwendungen und deren Schwachstellen wurde mit dem Autocomplete-Check-Plugin eine Browser-Erweiterung entwickelt, die Eingabeelemente auf Webseiten auf autocomplete-Attribute überprüft und entsprechend markiert. Das Plugin nutzt Heuristiken um eine Vorhersage darüber zu treffen, ob ein autocomplete-Attribut gefordert ist, und falls ja, welcher Wert benötigt wird. In der Validierung bescheinigt der Betreuer-Test dem Plugin eine hohe Genauigkeit, sofern eine HTML-konforme Verwendung der Label-Elemente gegeben ist. Das Autocomplete-Check-Plugin erweitert somit die bestehenden Anwendungen und ist durch sein Alleinstellungsmerkmal, Vorhersagen von autocomplete-Werten treffen zu können, bislang einmalig.

Das Autocomplete-Check-Plugin ist so strukturiert, dass der Matching-Algorithmus leicht um weitere Matching-Tests erweitert werden kann. Manche Feldeigenschaften, auf die getestet werden könnte, werden bereits geparst und können direkt übernommen werden. Neben dem Matching-Algorithmus wäre in Zukunft ein Verfahren, das auf maschinellem Lernen basiert, eine ideale Erweiterung für das Plugin. Dadurch könnten genauere Ergebnisse, sowie bessere Einschätzungen, ob es sich bei den analysierten Eingabefeldern um persönliche Nutzerdaten handelt, getroffen werden. Das Plugin bietet die Funktion, Webseiten händisch in Bezug auf autocomplete-Attribute zu labeln, und diese mit umfangreichen Metadaten zur Webseite und den Eingabefeldern in einer Datenbank zu speichern. Auf dieser Grundlage könnte ein Datensatz für autocomplete-Attribute erstellt werden, der sowohl das Trainieren von Entscheidungsmodellen auf Basis künstlicher Intelligenz, aber auch eine Auswertung und Analyse zur aktuellen Verwendung des autocomplete-Attributs im öffentlichen Internet zulassen würde.

Die Veröffentlichung des Plugins auf GitHub lädt dazu ein, das Autocomplete-Check-Plugin weiter zu verbessern.