Video-Kurs pluralsight.com: Meeting Web Accessibility Guidelines (Section 508/ WCAG 2.1)

Web Conformance Guidelines

Es gibt zwei verschiedene Guidelines, an denen man sich orienteiren kann, man sollte sich für eine entscheiden.

Section 508 (United States)

Updated 2017, nur US
Zielt auf Websites, die von der Regierung (federal government) entwickelt oder benutzt werden
www.section508.gov

WCAG 2.1: Level A, AA, AAA (Word Wide Web Consortium)

Soll ein global akzeptierter Standard für Barrierefreiheit sein
78 Kriterien organisiert in 13 Guidelines unter den 4 Prinzipien POUR (Perceivable, Operable, Understandable, Robust -> Wahrnehmbar, Bedienbar, Verständlich, Stabil)

Wahrnehmbar: alle Benutzer müssen die angezeigte Information wahrnehmen können
Bedienbar: alle Benutzer müssen das Interface bedienen können
Verständlich: alle Benutzer die Information und die Bedienung des Interfaces verstehen können
Stabil: alle Benutzer müssen auf den Content zugreifen können, wenn sich die Technik weiterentwickelt

3 Konformations-Level: A, AA, AAA
A: Basis Level, AA: Größte und am meisten verbreitete Barrieren AAA: Höchstes Level

www.w3.org/WAI/standards-guidelines/wcag (Gute Grundlage, aber schwierig zu lesen und kann verwirrtend sein)

Welche Guideline?

- **WCAG 2.1 Level AA** (damit erledigt man automatisch auch Section 508)

HTML

- Gutes semantisches HTML unabdingbar

Document Structure und Landmarks

Seitenstruktur	**Inhaltsstruktur**
Maschinen lesbarer Code	Menschen lesbarer Code
Landmarks	Headlines

Wichtiger Code:

Start:
<!doctype html>

Richtiges Parsing ermöglichen durch: tags richtig öffnen und schließen, keine doppelten Element Attribute, keine doppelten ids

Sprache der Seite:
<html lang="de">

Wichtige Infos (auch wenn sie nicht als Fehler angezeigt werden, wenn sie fehlen):
<meta charset="utf-8">
<meta http-equiv="x-ua-compatible" content="ie=edge">

Titel
<title></title>

- Für jede Seite anders (auch gut für SEO)

Mobil:
<meta name="viewport" content="width=device-width, initial-scale=1.0, shrink-to-fit=no">

Relative Einheiten für das Sizing (rem)

- Damit User mit der für sie geeigneten Technik zoomen können. User müssen bis zu 200% zoomen können, ohne Funktion und Lesbarkeit zu verlieren: Text, Bilder und Funktionen dürfen nicht abgeschnitten werden

Body: Landmarks (header, main, footer, nav, aside)

Mehrere nav: aria-labels nötig

Inhaltsstruktur: h1 (eine, sollte das erste Element in <main> sein) bis h6. Nicht skippen.
Im Header sollte keine h1 sein.

Exkurs: Bedienen von Voice-Over (mac)

- Voiceover an: Command + F5

- Webrouter anschalten (zum Navigieren über Landmarks oder Headlines): Control + Option + U
- Shortcuts anschalten: (zum Navigieren über Headlines mit Keyboard Shortcuts) Control + Option + Command + H
- Rückwärts: Control + Option + Shift + Command + H

Listen

3 Typen mit verschiedenen Bedeutungen:
 Ordered list, <ul> Unordered list, <dl> Description list (z. B. Glossar oder sogar FAQ)

Warum Listen benutzen?

- Um die Semantik zu nutzen, damit Screenreader besser damit umgehen können und Listen finden, den Typ der Liste festzustellen und die Anzahl der Listeneinträge zu bekommen, damit sie wissen wo in der Liste man sich befindet

Die Seite braucht eine Logisch Struktur, und visuelle Hinweise via css müssen auch nicht sichtbar über Semantik übertragen werden (z. B. Headlines)

Navigation und Skip Links

- Konsistente Navigaton
- Mehrere Wege um Seiten / Content zu finden
- Bedeutungsvoller Link Content
- Konsistentes Interface
- Skip Links

Innerhalb von <nav>: Liste machen
Navi-Einträge: Plain Text nutzen, keine Bilder
Navi-Einträge sollten immer und überall an der gleichen Stelle stehen und in der gleichen Reihenfolge sein

Mehrere Wege, Seiten zu finden: Navi, Suche (form mit role="search"), Sitemap.html (verlinkt vom Footer aus)

Buttons und Image sollten immer gleich benannt werden, wenn sie die gleiche Funktion haben

Linktexte sprechend, wenn das nicht geht, für Screenreader Text einfügen (z. B. show-for-sr -> mit css verstecken)

Bypass blocks: Blöcke, die sich wiederholen, überspringen. -> Skip link zum main-content. Muss der allererste Link auf der Seite sein
Zusätzlich zum css für skiplinks ist dieses js gut (Cross-browser Support)

```
var skipLink = document.querySelector('.skip-link');
skipLink.addEventListener('click', function(e) {
```

```
  document.querySelector(skipLink.getAttribute('href')).focus();
});
```

Tables

- Zum Darstellen von Daten in Reihen und Spalten. Nicht für Layout!

Für Screenreader muss man sie aufbereiten:

<caption> (nicht unbedingt nötig, aber gut zur Beschreibung, wichtig bei komplexen Tabellen)
<thead> <tfoot> <tbody> vor allem bei komplexen Tabellen, aber schadet auch so nicht
<th>
<td>
Scope wichtig (scope=”col”, scope=”row”)
und headers (bei komplizierten Tabellen können ids und mehrere tbody und mehreren headers verwendet werden)

Formulare

Barrierefreie Formulare: Fehleridentifizierung, Farbe, Tastatur-Navigation / Fokus-Anzeige

- Farb-Kontrast Vordergrund zum Hintergrund ist wichtig (und umgebende Farben)
- Placeholder Texte sind ein Problem (wenn sie zu dunkel sind, hält man sie für vorausgefüllte Felder, Browser behandeln sie unterschiedlich, in manchen verschwinden sie sobald man ins Feld klickt und dann ist die Beschreibung, was ins Feld muss weg -> ohne Label weiß man dann nicht mehr was hier hin muss)
- Besser sind Labels
- Gruppierung wichtig
- Pflichtfelder kennzeichnen
- Fokus-Anzeige unbedingt drin lassen (man kann sie per css hübscher machen)
- Keyboard-Traps vermeiden
- Fehlermeldungen gut sichtbar und zuordenbar, und nicht nur mit Farbe kennzeichnen und Infos liefern, wie man den Fehler beheben kann

Man sollte die native HTML5 Browser Validierung abschalten, da diese „HTML5 validation ballons“ nicht barrierefrei sind, und stylen kann man sie auch nicht wirklich.
<form .... novalidate>

Text (<p>) sollte außerhalb von <form> stehen.

Labels:
<label for=“idinput“>Name (Required):</label><input id=”idinput” required>
Geht auch: explicit label (ohne for=)
<label><input></label>

Die richtigen types für inputs verwenden (text, email, tel)

Gruppieren mit <fieldset> + <legend> verwenden (auf jeden Fall Radio Buttons und Checkboxen)

Man kann visuelle Labels auch nur für Screenreader anzeigen, aber man sollte überlegen, ob sie für sehende Benutzer auch interessant/hilfreich sind.

Fehler-Validierung mit css und js und aria
z. B.
Fehlermeldung

Oder mit Blur (mit aria-live="assertive")

<label for="name" aria-live="assertive">Name (Required):Fehlermeldung</label><input>

- ➔ Die Fehlermeldung wird gleich vorgelesen, wenn man nichts eingibt

Section am Anfang der <form> für eine Liste der Fehler mit aria-live
<section id="errors" aria-live="assertive" tabindex="-1">

Medien

Bilder

- Kein Text auf Bildern verwenden
- Text-Alternativen für Bilder anbieten: alt attribute, dekorative Bilder müssen ein leeres alt Attribut haben: alt="" (wird im Screenreader dann nicht gelistet)
- Tipps für guten ALT Text: das Bild nicht buchstäblich beschreiben „Dies ist ein Bild das zeigt: ....". Besser: Beschreiben was die Bedeutung oder der Zweck des Bildes ist. Was visuell übermittelt wird, muss auch programmatisch übermittelt werden (Wahrnehmbarkeit)

Hintergrundbilder mit css

- Brauchen Text-Content: Beschreibung des Icons

SVG

- Role beim svg vergeben (role="img")
- <title> mit id benutzen
- <desc> bei längerem Content

- Aria-labelledby benutzen und den title oder die desc zu referenzieren

Audio

- Textalternative muss da sein (transcript des audio): Bezahlter Service, Spracherkennungs-Apps (Google Docs Voice Typing, Windows Speech Recognition, Apple Dictation), Manuell
- Tipps: Namen der Sprecher einfügen, Alles beschreiben, auch sounds (die ganze Geschichte beschreiben)
- Wie? Inline, Link zum Transcript

Video

- Textalternative wie bei Audio Dateien
- Captions anbieten (Open: immer sichtbar, in Videos eingebettet / Closed: kann vom User an- und abgeschaltet werden)
- YouTube bietet Auto-subtitle an (Guter Startpunkt)
- Manueller Eintrag
- Video Captioning Services
- Audio Beschreibung: Alle nicht sichtbaren Hinweise beschreiben, z. B. Donner und Blitz vor dem Fenster.

Weiteres

- Kein Autoplay (pause, stop, hide)
- Kein Flackern
- Iframes sollten title attribute haben

Responsive Web Design und Barrierefreiheit

Context wechseln

- Sollte nicht unerwartet sein, z. B. wechslen der Seite durch ein Dropdown Menü: besser mit Button zum Absenden, aber es sieht nicht gut aus. Besser: Menü Button den man aktiv aufmachen muss

Reihenfolge von Content / Fokus

- Wenn man die Seite kleiner schiebt und z. B. die Sidebar bei einem breakpoint woanders hinspringt muss die Reihenfolge beim durchtabben gleich bleiben, bzw. Sinn machen
- Visuelle Ordnung muss der DOM Ordnung ensprechen

Mobile Geräte

- Nicht auf eine Orientation beschränken
- Pointer Gestures: nicht nur diese benutzen, man braucht auch Buttons
- Pointer Cancellations: Man muss Aktionen rückgängig machen können (action bei click und tap, nicht touch-down)
- Motion Actuation: schütteln, Rotieren: kann nicht die einzelne Möglichkeit sein, Button anbieten

Zusätzliche Responsive Richtlinien

- Content barrierefrei verstecken, z. B. mit dem hidden attribute (display:none und visibility: hidden -> versteckt für alle)
- Relative Einheiten benutzen (em, rem, %)
- Text-Spacing (dies sollte mindestens vom user einstellbar sein, ohne dass die Seite nicht mehr benutzbar ist): Line-height ist 1,5x der Font-size, 2x font-size spacing nach Absätzen, Letter-spacing sollte mindestens .12x der font-size sein, Word-space sollte mindestens .16x der font-size sein
- Es sollte keine Scrollbars geben beim Vergrößern

Datum des Kurses: 28.4.20