Progressive Web Apps barrierefrei entwickeln

entwickler.de – entwickler.de Deine Wissensplattform

[…]Weiterlesen…
von Markus Lemcke

Öffentliche Stellen des Bundes sind seit September 2016 gesetzlich dazu verpflichtet, barrierefreie Apps einzusetzen [1]. Progressive Web Apps haben dabei eine großen Vorteil gegenüber nativen Apps: Bei einer nativen App gibt es für drei unterschiedliche Betriebssysteme drei unterschiedliche Richtlinien. Wenn PWAs barrierefrei nach der EN 301 549 [2] entwickelt werden, kann eine App auf drei unterschiedlichen Betriebssystemen eingesetzt werden und die Barrierefreiheit ist auf allen dreien gleich gut.

Seit dem 1. Mai 2002 gibt es das Behindertengleichstellungsgesetz (BGG). Dieses Gesetz wurde im Jahr 2016 überarbeitet und wurde im September 2016 neu verabschiedet. Seit September 2016 sind öffentliche Stellen des Bundes nach § 12a Barrierefreie Informationstechnik Absatz 1 zur Barrierefreiheit bei Apps verpflichtet. Entsprechend haben auch die drei großen Unternehmen Google [3], Apple [4] und Microsoft [5] Richtlinien zur barrierefreien Appentwicklung veröffentlicht.

Bei der Überlegung, wie viele Menschen von barrierefreier Appentwicklung profitieren, stellt man fest, dass es nicht nur 7,8 Millionen schwerbehinderte Menschen [6] betrifft, denn auch einigen der 18,3 Millionen Senioren [7] können barrierefreie Apps das Leben erleichtern.

Vorteile von PWAs

Es gibt native Apps und Progressive Web Apps (PWAs). Als native Apps werden Anwendungen auf mobilen Endgeräten bezeichnet, die speziell für das Betriebssystem des jeweiligen Endgeräts konzipiert und entwickelt wurden. Sie werden meist über die App Stores, die an das Betriebssystem gekoppelt sind, als kostenfreie oder auch kostenpflichtige Anwendungen vertrieben. PWAs hingegen werden mit HTML, CSS und JavaScript entwickelt, sind somit plattformunabhängig und können auf Desktop- und Mobile-Betriebssystemen eingesetzt werden.

Richtlinien, die dafür sorgen, dass Apps von allen Menschen mit unterschiedlichen körperlichen Einschränkungen bedient werden können, haben so viele Prüfungskriterien, wie notwendig sind, um alle körperlichen Beeinträchtigungen zu berücksichtigen. Die Anzahl der Prüfungsschritte bei den Richtlinien von Google, Apple und Microsoft sind unterschiedlich. Das bedeutet, wenn bei einer Richtlinie Prüfungsschritte fehlen, werden Menschen mit bestimmten körperlichen Beeinträchtigungen ausgrenzt.

PWAs sind plattformunabhängig, sie können also auf allen Betriebssystemen eingesetzt werden. Wenn eine PWA nach der EN 301 549 barrierefrei entwickelt wird, ist sie auf allen Betriebssystemen für alle Menschen mit körperlichen Beeinträchtigungen barrierefrei [8].

https://phpconference.com/session-qualification/ipc-webentwicklung/?layout=contentareafeed&widgetversion=1&utmtrackerversion=1&seriesId=oLGXGfBxxeHKh6rMj
Um es auf den Punkt zu bringen: Die Entwicklung von barrierefreien PWAs sorgt dafür, dass alle Menschen mit unterschiedlichen körperlichen Beeinträchtigungen eine App auf jedem Betriebssystem bedienen können.

Richtlinie EN 301 549

Der Standard für Barrierefreiheit von Webinhalten, den das W3C 1999 eingeführt hat und der seitdem global angewendet wird, sind die Web Content Accessibility Guidelines (WCAG). Diese internationale Richtlinie wird kontinuierlich weiterentwickelt und erfuhr 2018 mit den WCAG 2.1 [9] ihre vorerst letzte Aktualisierung.

Die WCAG in ihrer aktuellen Fassung ist die Grundlage für die europäische Norm EN 301 549. Allerdings geht es in der EN 301 549 nicht nur um Barrierefreiheit im Web, sie beinhaltet ebenso Barrierefreiheit bei

- Hardware
- Nicht-Web-Dokumente
- Software
- Dokumentation und unterstützende Dienste

Die EN 301 549 ist also die erste Richtlinie, die der Komplexität des Themas digitale Barrierefreiheit Rechnung trägt und ist deswegen umfassender.

Um den Unterschied zwischen EN 301 549 und WCAG 2.1 deutlich zu machen, hier ein Beispiel: Wenn eine Webseite in Frankreich barrierefrei gemacht werden soll, dann kommt die EN 301 549 zur Anwendung. Bei einer Webseite in Amerika kommt hingegen die WCAG 2.1 zur Anwendung.

Die EN ist eine Richtlinie für Webprogrammierung. Deswegen ist es keine gute Idee, native Apps, die mit Java oder Swift entwickelt werden, nach der EN 301 549 barrierefrei zu entwickeln. PWAs nach der EN 301 549 barrierefrei zu machen, passt perfekt, weil diese mit HTML, CSS und JavaScript entwickelt werden.

Einzelne Kriterien der EN 301 549 umgesetzt

In den folgenden Abschnitten schauen wir uns anhand von praktischen Anwendungsbeispielen an, wie bestimmte Prüfungsschritte umgesetzt werden können.

Screenreadertauglichkeit

Ein Screenreader ist eine Vorlesefunktion für blinde Menschen. Da PWAs plattformunabhängig sind, gibt **Tabelle 1** einen Überblick über Screenreader, die für unterschiedliche Betriebssysteme zur Verfügung stehen, und wie diese aktiviert werden können.

Screenreader	Betriebssystem	Menü
Sprachausgabe	Windows 11	Einstellungen | Barrierefrei-heit | Sprachausgabe [10]
NVDA	Windows 11	Downloadlink https://nvda.bhvd.de/
Talkback	Android 12	Einstellungen | Bedienungshilfen | Talkback [11]
Voice Over	IOS 15.5	Einstellungen | Bedienungshilfen | Voice Over
Voice Over	macOS 10.15	Systemeinstellungen | Bedienungshilfen | Voice Over [12]
Orca	Ubuntu 22.04	Einstellungen | Barrierefreiheit | Bildschirmleser [13]

Tabelle 1: Überblick über unterschiedliche Screenreader

Screenreadertauglichkeit bedeutet, dass eine App-Oberfläche so entwickelt ist, dass sie von Screenreadern vorgelesen werden kann. Um das responsive Webdesign besser verwirklichen zu

können, können Div-Container als Schaltflächen verwendet werden. Warum dies wichtig ist, wird im Abschnitt „Motorische Einschränkungen“ erklärt. Damit der Screenreader weiß, dass der Div-Container eine Schaltfläche ist, muss das *role*-Attribut [14] den Wert *button* bekommen. Um einen Text festzulegen, der von Screenreadern vorgelesen wird, muss das Attribut *aria-label* verwendet und ihm ein Wert zugewiesen werden. Hier ein HTML-Beispiel:

```
<div role="button" aria-label="Quiz starten">START</div>
```

Mit den Attributen *role* und *aria-label* wird der Div-Container zur screenreadertauglichen, responsiven Schalfläche.

PWAs können nicht nur in mobilen Betriebssystemen, Android und IOS, sondern auch auf Desktopbetriebssystemen (Windows, Ubuntu und macOS) ausgeführt werden. Diese Möglichkeit ist für Menschen interessant, bei denen aufgrund einer Behinderung eine motorische Einschränkung in den Händen vorhanden ist. Blinde Menschen können ebenso den Wunsch haben, eine PWA auf einem Computer oder Laptop bedienen zu können. Sobald eine PWA auf einem Computer ausgeführt wird, ist es wichtig, dass sie komplett ohne Maus, also nur per Tastatur bedienbar ist [15]. Deswegen sollten alle Bedienelemente per Tabulatortaste erreichbar sein. Dem Div-Container, der als Schaltfläche verwendet wird, wird deswegen das Attribut *tabindex* hinzugefügt. Hier ein HTML-Beispiel:

```
<div tabindex="0">START</div>
```

Nach dem Hinzufügen des Attributs sind die Div-Schalter per Tabulatortaste erreichbar und somit ist die Grundvoraussetzung des Tastaturbedienbarkeit erfüllt. Dazu gehört auch, dass wichtige Funktionen per Tastenkürzel ausgeführt werden können. Das hilft blinden und sehbehinderten Menschen. Tastenkürzel können mit JavaScript wie folgt realisiert werden:

```
document.addEventListener("keydown", (varevent) => {
```

```
  switch (varevent.key) {
    case "s":
      document.getElementById("btnStart").click();
      break;
    case "w":
      document.getElementById("btnWeiter").click();
      break;
  }
});
```

Der JavaScript-Code sorgt dafür, dass der Schalter Start durch Drücken des Buchstaben *s* und der Schalter Weiter mit dem Buchstaben *w* ausgeführt werden kann.

Sichtbarkeit des Tastaturfokus

Dieses Thema bekommt dann große Bedeutung, wenn eine PWA auf einem Computer oder Laptop ausgeführt wird. Menschen mit einer Sehbehinderung haben Probleme, zu erkennen, welches Bedienelement den Tastaturfokus hat. In Eingabefeldern wird der Textcursor oft als schmaler senkrechter Strich dargestellt. Das ist für Menschen mit einer Sehbehinderung sehr schwer zu erkennen. Microsoft hat im Betriebssystem Windows 11 hierfür eine Lösung. In Einstellungen | Barrierefreiheit | Textcursor kann bei Textcursor-Indikator [16] die Darstellung des Textcursors angepasst werden.

In PWAs kann der App-Entwickler dafür sorgen, dass aktive Bedienelemente die Farbe Gelb als Hintergrundfarbe zugewiesen bekommen. Wenn ein Bedienelement bei Aktivierung eine gelbe Hintergrundfarbe bekommt, ist das für Menschen mit Sehbehinderung sofort sichtbar. Das kann mit CSS sehr einfach gelöst werden:

```
#btnStart:focus{background-color: yellow; color: black;}
```

Barrierefreier Farbkontrast

Menschen mit einer Farbfehlsichtigkeit können nicht immer einer Farbe den richtigen Namen zuordnen. Ihnen fehlt das Gefühl, welche Farben zusammenpassen. Ein Text mit einer dunklen Schriftfarbe auf einer dunklen Hintergrundfarbe ist für sie ebenso wenig erkennbar wie Text mit einer hellen Schriftfarbe auf einer hellen Hintergrundfarbe. Für diese Menschen ist es wichtig, dass eine App-Oberfläche einen barrierefreien Farbkontrast zwischen Schriftfarbe und Hintergrundfarbe [17] hat. Die Überprüfung des Farbkontrasts auf Barrierefreiheit einer App-Oberfläche kann mit der kostenlosen Software Colour Contrast Analyser [18] vorgenommen werden.

Zur generellen Frage der Farbgestaltung von PWA-Oberflächen ist es auch möglich, sich Anregungen von Material Design [19] von Google zu holen.

Motorische Einschränkungen

Motorische Einschränkungen betreffen „kleine Bewegungen“ (Feinmotorik) und „große Bewegungen“ (Grobmotorik). Menschen mit motorischen Einschränkungen in den Händen können Probleme haben, zu kleine Schaltflächen anzutippen. Google führt deswegen in seinen Richtlinien zur barrierefreien Appentwicklung das Kriterium „Use large, simple controls“ auf [20]. Hier empfiehlt Google eine Mindestgröße von Schaltflächen von 48dpx48dp. Bei der Entwicklung von PWAs wird empfohlen, diese Mindestgröße umzusetzen, weil sie auch von dem Accessibility Scanner (so heißt das Überprüfungstool von Google) kontrolliert wird.

Menschen mit motorischen Einschränkungen können auf die Idee kommen, eine PWA auf einem Tablet, iPad oder Computer zu bedienen, mit der Hoffnung, dass dort die Schaltflächen größer dargestellt werden. Wie im Abschnitt Screenreadertauglichkeit

erklärt, ist das der Grund, warum die Schaltflächen nicht als JavaScript-Buttons

```
<button onclick="myFunction()">Click me</button>
```

sondern als Divs

```
<div role="button" aria-label="Quiz starten" tabindex="0">START</div>
```

definiert werden.

Mit Media Queries (gehört zu Cascading Style Sheets) kann dafür gesorgt werden, dass bei einer Displaygröße von 768 x 1024 die Schaltflächen größer dargestellt werden. Folgender CSS-Code zeigt, wie es geht:

```
@media only screen and (min-width: 768px){
  .schalter{
    height: 6.0rem;
  }
}
```

Es wird die Displaygröße 1024 × 768 abgefragt, die bei Tablets und iPads oft anzutreffen ist. Die Abfrage funktioniert ebenfalls bei Bildschirmauflösungen von Computern und Laptops. Wenn die Displaygröße zutrifft, werden die Div-Container, welche eine CSS-Klasse *Schalter* besitzen, in der Höhe auf 6.0 rem angepasst. Somit werden die Schaltflächen größer und Menschen mit motorischen Einschränkungen in den Händen haben es leichter, eine Schaltfläche mit dem Finger oder der Computermaus anzutippen.

Übernahme von Einstellungen des Betriebssystems

Bestimmte Personengruppen mit körperlichen Einschränkungen nehmen grundsätzliche Anpassungen im Betriebssystem vor mit der Erwartung, dass die Apps diese Einstellungen übernehmen. Sehbehinderte Menschen passen im Betriebssystem die

Schriftgröße an. Menschen mit einer Farbfehlsichtigkeit können im Betriebssystem den hohen Kontrast aktivieren.

PWAs sollen, wenn installiert, so plattformnah wie möglich aussehen und sich auch so verhalten. Deswegen ist dieses Thema etwas komplex. Für Menschen mit einer Sehbehinderung ist es wichtig, dass die App-Oberfläche vergrößert bzw. gezoomt werden kann. Wie dies auf unterschiedliche Betriebssysteme umgesetzt wird, zeigt Tabelle 2.

Betriebssystem	**App-Oberfläche zoomen**
Windows	Im App-Menü (3 senkrechte Punkte) gibt es einen Menüpunkt Zoomen
Ubuntu	Im App-Menü (3 senkrechte Punkte) gibt es einen Menüpunkt Zoomen
macOS	Im App-Menü (3 senkrechte Punkte) gibt es einen Menüpunkt Zoomen
Android	Einstellungen | Bedienungshilfen | Text und Anzeige | Anzeigegrösse
IOS	Bedienungshilfen | Zoom

Tabelle 2: Vergrößern der App-Oberfläche auf unterschiedlichen Betriebssystemen

An diesem Beispiel wird deutlich, dass es keine einheitliche Regel gibt. Bei drei Betriebssystemen wird das Zoomen in der App-Oberfläche umgesetzt und bei zwei Betriebssystemen wird die Einstellung im Betriebssystem übernommen. Dieses Wissen ist wichtig, wenn App-Entwickler bestimmte Barrierefreiheitsfunktionen auf unterschiedlichen Betriebssystemen testen möchten.

Progressive Web Apps auf Barrierefreiheit validieren

Nach der App-Entwicklung mit Barrierefreiheit im Blick muss

zum Schluss herausgefunden werden, ob alles tatsächlich wie gewünscht funktioniert. Es gibt zwei grundsätzliche Methoden, PWAs auf Barrierefreiheit zu testen: ein automatisierter Test oder ein Test von Hand aufgrund der Richtlinien EN 301 549.

Zunächst wird der automatisierte Test mit dem Accessibility Scanner [21] betrachtet. Dieser ist für Smart-phones und Tablets ab Android 6.0 geeignet [22]. Auf einem Tablet mit Android 12 ist er erfreulicherweise schon vorinstalliert. Mit dem Accessibility Scanner können native Apps und PWAs auf Barrierefreiheit überprüft werden. Aktiviert wird er in Einstellungen | Bedienungshilfen | Accessibility Scanner. Zunächst wird eine Erlaubnis benötigt, dass der Scanner über anderen Apps eingeblendet werden darf. In den Einstellungen können das Textkontrastverhältnis, das Bildkontrastverhältnis und die Größe des Berührungsbereichs angepasst werden.

Es gibt zwei Möglichkeiten, mit dem Accessibility Scanner eine PWA auf Barrierefreiheit zu überprüfen: „Aufnehmen“ und „Snapshot“. „Aufnehmen“ macht jedes Mal ein Screenshot, wenn sich die App-Oberfläche ändert. Mit dieser Methode kann man sehr schnell eine komplette PWA auf Barrierefreiheit überprüfen. Bei „Snapshot“ kann der App-Entwickler festlegen, wann er die App-Oberfläche überprüfen möchte. Wenn der Accessibility Scanner Fehler findet, dann sind die Fehlermeldungen leicht verständlich formuliert, sodass der App-Entwickler weiß, was er zur Behebung tun muss.

Eine weitere Möglichkeit, die Barrierefreiheit einer PWA zu überprüfen, ist das kostenlose Tool Google Lighthouse [23]. Google Lighthouse ist im Browser Google Chrome in den Entwickler-Tools zu finden. Folgende Schritte müssen umgesetzt werden, um eine PWA mit Google Lighthouse auf Barrierefreiheit zu überprüfen: Die PWA in Google Chrome öffnen. Die Entwickler-Tools öffnen über das Menü Weitere Tools | Entwickler-Tools oder mit der Tastenkombination STRG + UMSCHALT + I. Jetzt das Menü „>>“ aktivieren und dann das Menü Lighthouse auswählen. Hier den Modus auf Navigation (Default)

lassen. Bei Gerät die Option Mobil auswählen. Bei Categories die Option Bedienungshilfen auswählen. Jetzt mit Aktivieren des Schalters Analyze page load die Analyse auf Barrierefreiheit starten.

Wenn die Analyse beendet ist, wird ein Kreis angezeigt, in dem eine Zahl steht. Die Zahl 100 ist die beste Bewertung, die erreicht werden kann. Als Erstes werden Prüfungskriterien angezeigt, die nicht bestanden wurden. Weiter unten werden Elemente angezeigt, die manuell geprüft werden müssen. Noch weiter unten werden *Bestandene Prüfungen* und ganz unten *Nicht zutreffend* angezeigt. In der Kategorie *Zusätzliche Elemente zur manuellen Überprüfung* gibt es noch einen Link zur Seite „How To Do an Accessibility Review“ [24]. Hier gibt Google-Mitarbeiter Rob Dodson Tipps in Form eines YouTube-Videos und Text, wie eine Überprüfung auf Barrierefreiheit durchgeführt werden kann.

Die dritte Möglichkeit ist, die PWA nach den 98 Prüfungsschritten der EN 301 549 von Hand zu prüfen. Zusätzlich ist es hilfreich, die App von Menschen mit Behinderungen testen zu lassen. Ob beispielsweise eine PWA für blinde Menschen bedienbar ist, weiß ein blinder Mensch am besten.

Fazit

Das Entwickeln von barrierefreien PWAs ermöglicht es, barrierefreie Apps zu entwickeln, die auf allen Betriebssystemen den gleichen Standard in Sachen Barrierefreiheit zu Verfügung stellen. Das bedeutet, dass es keine Personengruppe gibt, die von der Nutzung einer App auf einem bestimmten Betriebssystem ausgeschlossen wird.

Markus Lemcke ist seit elf Jahren selbstständig im Bereich Barrierefreiheit von Webseiten, Software und Betriebssystemen. Er ist Dozent an Hochschulen und schreibt als Autor für Fachmagazine. Sein Schwerpunkt ist die barrierefreie Softwareentwicklung mit Java und C#.

Links & Literatur

[1] https://www.gesetze-im-internet.de/bgg/__12a.html

[2] https://www.etsi.org/deliver/etsi_en/301500_301599/301549/03.02.01_60/en_301549v030201p.pdf

[3] https://developer.android.com/guide/topics/ui/accessibility/apps

[4] https://developer.apple.com/library/archive/documentation/UserExperience/Conceptual/iPhoneAccessibility/Making_Application_Accessible/Making_Application_Accessible.html

[5] https://docs.microsoft.com/de-de/windows/apps/design/accessibility/developing-inclusive-windows-apps

[6] https://www.destatis.de/DE/Themen/Gesellschaft-Umwelt/Gesundheit/Behinderte-Menschen/_inhalt.html

[7]

https://www.destatis.de/DE/Themen/Querschnitt/Demografischer-Wandel/Aeltere-Menschen/bevoelkerung-ab-65-j.html

[8] https://www.bitvtest.de/bitv_test/das_testverfahren_im_detail/pruefschritte.html

[9] https://www.w3.org/TR/WCAG21/

[10] https://www.youtube.com/watch?v=vAwf4WFq1jg

[11] https://www.youtube.com/watch?v=ABdOTZUGZR0

[12] https://www.youtube.com/watch?v=6MjR498CyMM

[13] https://www.youtube.com/watch?v=HIKGTi8O9KQ

[14] https://webtest.bitv-test.de/index.php

[15] https://webtest.bitv-test.de/index.php?a=di&iid=267&s=n

[16] https://www.youtube.com/watch?v=cjl0gTkAVc8

[17] https://webtest.bitv-test.de/index.php?a=di&iid=260&s=n

[18] https://www.tpgi.com/color-contrast-checker/

[19] https://material.io/design/color/the-color-system.html

[20] https://developer.android.com/guide/topics/ui/accessibility/apps

[21] https://support.google.com/accessibility/android/answer/6376570?hl=de

[22] https://www.youtube.com/watch?v=GRV1kucMqIo

[23] https://developers.google.com/web/tools/lighthouse

[24] https://web.dev/how-to-review/