

BACHELORARBEIT

Barrierefreiheit in React Native Apps am Department Medientechnik

vorgelegt am 30. September 2025
Alexander Colin Allen Farwer

Erstprüferin: Prof. Dr. Larissa Putzar
Zweitprüfer: Prof. Dr. Eike Langbehn

**HOCHSCHULE FÜR ANGEWANDTE
WISSENSCHAFTEN HAMBURG**
Department Medientechnik
Finkenau 35
22081 Hamburg

Zusammenfassung

Diese Arbeit untersucht die Entwicklung von barrierefreien mobilen Anwendungen mit dem React Native Framework. Die App „Whatch“, die zur Entdeckung von Filmen dient, wird im Rahmen dieser Arbeit barrierefrei gestaltet. Hintergrund ist das Barrierefreiheitsstärkungsgesetz, welches seit Sommer 2025 alle Apps größerer Unternehmen zur Barrierefreiheit verpflichtet.

Die zentralen Fragestellungen der Arbeit sind, wie gut sich React Native zur Erstellung barrierefreier Apps eignet und inwiefern eine nach den gesetzlich geforderten Richtlinien gestaltete Anwendung für blinde NutzerInnen tatsächlich bedienbar ist.

Die Evaluation basiert auf Nutzertests mit zwei blinden Testpersonen, die die App mithilfe von Screenreadern auf iOS und Android genutzt haben. Die Ergebnisse zeigen, dass die Anwendung insgesamt sehr gut bedienbar ist und sich im Vergleich zu vielen anderen Apps positiv hervorhebt. Gleichzeitig wird deutlich, dass React Native in Bezug auf Barrierefreiheit erhebliche Einschränkungen aufweist und häufig nur mit hohem Aufwand gute Ergebnisse erreicht werden können.

Abstract

This paper examines the development of accessible mobile applications using the React Native framework. The app "Whatch," which is used for getting movie suggestions, is made accessible in this work. Reason for this is the German Accessibility Improvement Act, which requires all apps from larger companies to be accessible starting summer 2025.

The central questions of the paper are how well React Native is suited for creating accessible apps and how usable an application designed according to the guidelines is for blind users.

The evaluation is based on user tests with two blind participants who used the app with the help of screen readers on iOS and Android. The results show that the application is overall very user-friendly and stands out compared to many other apps. At the same time, it is clear that React Native has significant limitations in terms of accessibility and that good results are often only achievable with considerable effort.

Inhaltsverzeichnis

Abbildungsverzeichnis	III
1. Einleitung	1
1.1. Motivation	1
1.2. Zielsetzung	1
1.3. Struktur der Arbeit	1
2. Grundlagen	3
2.1. Barrierefreiheitsgesetz	3
2.2. Web Content Accessibility Guidelines (WCAG) 2.1	3
2.3. EN 301 549	4
2.4. Anforderung des BFSG	4
2.5. React Native	5
2.6. Screenreader	6
2.7. Existierende App „Whatch“	6
2.8. Barrierefreiheit Testwerkzeuge	8
2.9. Stand der Barrierefreiheit in Android Apps	9
2.10. Entwicklung Barrierefreier mobiler Apps mit Cross-Platform Frameworks	10
3. Konzept und Planung	12
3.1. Anforderungen	12
3.1.1. EN 301 549	12
3.1.2. Plattformeigene Richtlinien	13
3.2. Schwerpunkte bei der Umsetzung der Barrierefreiheit	14
3.3. Probleme in der existierenden App	14
3.4. Vorgehensweise	15
4. Umsetzung	16
4.1. Navigation	16
4.1.1. Zurück Button	16
4.2. Kontrast	17
4.2.1. Anforderung	17
4.2.2. Problem	17

4.2.3.	Lösung	19
4.3.	Text- und Bedienelementgrößen	20
4.3.1.	Textgröße	20
4.3.2.	Bedienelementgrößen	21
4.3.3.	Suchleiste	24
4.4.	Screenreader Bedienelemente	24
4.4.1.	Buttons Navigationsleiste	24
4.4.2.	Schalter	25
4.4.3.	Status von Bedienelementen	27
4.4.4.	Streaminganbieter Icons	28
4.5.	Screenreader Navigation	29
4.5.1.	Filmdetails Seite	29
4.5.2.	Interface Änderungen	30
4.5.3.	Automatisch mehr Filme laden	31
4.5.4.	Headlines	32
4.6.	Kleinere Probleme	32
4.6.1.	Label für die Suchleiste	32
4.6.2.	Einstellungselemente nicht auswählbar	33
4.7.	Fazit der Umsetzung	33
5.	Abschließende Nutzertests	35
5.1.	Nutzertests zu Barrierefreiheit	35
5.2.	Testablauf	36
5.3.	Beobachtungen	36
5.4.	Zentrale Ergebnisse	36
5.5.	Nutzertest Fazit	37
6.	Evaluation	38
Literatur		39
Anhang		41
A.	Interviews	42
A.1.	Interview 1	42
A.2.	Interview 2	48

Abbildungsverzeichnis

2.1.	App „Whatch“ Screenshots vor der Überarbeitung	7
2.2.	App „Whatch“ Screenshots vor der Überarbeitung	7
2.3.	Apple Accessibility Inspector Meldung eines Barrierefreiheitsproblems	8
2.4.	Android Accessibility Scanner Meldung Radio Button nicht barrierefrei	9
4.1.	Screenshot: nur Zurück-Button	17
4.2.	Screenshot: kein Zurück-Button	17
4.3.	Screenshot: Buttons mit headerRight angezeigt	17
4.4.	Android Accessibility Scanner meldet Kontrastproblem	18
4.5.	Screenshot: Suchleiste	18
4.6.	Apple Accessibility Inspector meldet Kontrastproblem	19
4.7.	Screenshot: Textskalierung funktioniert nicht	21
4.8.	Screenshot: Funktionierende Textskalierung	21
4.9.	Screenshot: Schalter zu klein	22
4.10.	Screenshot: Radio Button zu klein	22
4.11.	Screenshot: Switch zu klein	22
4.12.	Vergleich iOS Schalter	23
4.13.	Screenshot: Streaming-Anbieter Icons	28
4.14.	Screenshot: Filmfakten Bulletpoints	28
4.15.	Screenshot: Einstellungsseite	33

1. Einleitung

1.1. Motivation

Im Sommer 2025 wurde das Barrierefreiheitsstärkungsgesetz (BFSG) in Kraft gesetzt. Es verpflichtet größere Unternehmen, ihre Webseiten und Software barrierefrei zu gestalten, um eine uneingeschränkte Nutzung für Menschen mit Behinderungen zu gewährleisten (Abteilung DG des Bundesministeriums des Innern im Auftrag des Beauftragten der Bundesregierung für Informationstechnik, [2025a](#)).

In dieser Arbeit soll die React Native App „Whatch“ für eine potenzielle Veröffentlichung untersucht und barrierefrei gestaltet werden. Die App vereinfacht das Suchen nach Filmen bei mehreren Streaminganbietern. Obwohl die Anwendung nicht unter das BFSG fällt und nicht zur Barrierefreiheit verpflichtet wäre, ist es dennoch sinnvoll, eine App für möglichst viele Nutzer bedienbar zu machen.

1.2. Zielsetzung

Das primäre Ziel dieser Arbeit ist, zu untersuchen, inwiefern sich React Native als Framework für die Entwicklung barrierefreier Anwendungen eignet. Zusätzlich soll ermittelt werden, wie gut eine Anwendung für blinde Nutzer zugänglich ist, wenn alle gesetzlichen Anforderungen an die Barrierefreiheit beachtet und implementiert werden. Um dies herauszufinden wird die App den gesetzlichen Anforderungen nach angepasst und anschließend von blinden Nutzern getestet.

1.3. Struktur der Arbeit

Die Arbeit ist in sechs Kapitel unterteilt. Zunächst werden die theoretischen Grundlagen, rechtlichen Anforderungen und die technischen Rahmenbedingungen bezüglich Barrierefreiheit erläutert. Es folgt das Vorgehen sowie die Planung der Implementierung. Anschließend werden die implementierten Änderungen an der Anwendung dokumentiert und erläutert. Im

nachfolgenden Schritt wird eine abschließende Bewertung durch Nutzertests durchgeführt, bevor die Arbeit mit einer Schlussfolgerung und einer Reflexion der Ergebnisse endet.

2. Grundlagen

2.1. Barrierefreiheitsgesetz

Das Barrierefreiheitsgesetz (BFSG) ist am 28. Juni 2025 in Kraft getreten und ist die Umsetzung des European Accessibility Act (EAA). Der EAA ist eine Richtlinie der Europäischen Union, die einheitliche Anforderungen an die Barrierefreiheit von Produkten und Dienstleistungen festlegt (Abteilung DG des Bundesministeriums des Innern im Auftrag des Beauftragten der Bundesregierung für Informationstechnik, [2025b](#)). Das BFSG setzt diese Vorgaben in deutsches Recht um und schreibt vor, dass alle Produkte und Dienstleistungen, die nach dem 28. Juni 2025 in der EU auf den Markt gebracht werden, die entsprechenden Barrierefreiheitsanforderungen erfüllen müssen.

Es besagt unter anderem, dass Webseiten und Apps von Unternehmen mit einem Jahresumsatz von über 2 Millionen Euro oder mindestens 10 beschäftigten Personen barrierefrei sein müssen (Abteilung DG des Bundesministeriums des Innern im Auftrag des Beauftragten der Bundesregierung für Informationstechnik, [2025a](#)).

2.2. Web Content Accessibility Guidelines (WCAG) 2.1

Die „Web Content Accessibility Guidelines“ (WCAG) sind technische Richtlinien, die vom World Wide Web Consortium (W3C) veröffentlicht werden. Sie beschreiben wie Webseiten für Menschen mit verschiedenen Behinderungen zugänglicher gestaltet werden können.

Die WCAG haben drei „conformance Level“, A (niedrigstes), AA und AAA (höchstes). Die Level bauen aufeinander auf. Das Kriterium „Contrast (Enhanced)“ des Levels AAA fordert beispielsweise ein höheres Kontrastverhältnis als das Kriterium „Contrast (Minimum)“ des Levels AA. Nach BFSG müssen alle Webseiten das Level AA erfüllen. Um dies zu erreichen müssen alle „A Kriterien“ und „AA Kriterien“ umgesetzt sein.

Die Richtlinien basieren auf vier grundlegenden Prinzipien, die Inhalte erfüllen müssen:

- perceivable / wahrnehmbar (z. B. Alternativtexte für Bilder)

- operable / bedienbar (z. B. alle Funktionen mit Tastatur bedienen zu können)
- understandable / verständlich (z. B. klare Beschriftung von Buttons)
- robust / robust (z. B. Inhalte werden korrekt von Hilfstechnologien erkannt)

Das bedeutet, dass alle Inhalte generell wahrnehmbar, bedienbar, verständlich und robust sein müssen, unabhängig von Fähigkeiten oder Beeinträchtigungen. Zum Beispiel müssen Inhalte mit Ton auch für Menschen wahrnehmbar sein, die nicht hören können. Und alle visuellen Inhalte müssen auch für blinde Menschen wahrnehmbar sein. Mit „robust“ ist gemeint robust gegenüber Hilfstechnologien. Inhalte sollen also zuverlässig und korrekt von verschiedenen Hilfstechnologien erkannt werden.

So soll sichergestellt werden, dass grundsätzlich alle Menschen Inhalte im Web nutzen können (Accessibility Guidelines Working Group (W3C WAI), [2025](#)).

2.3. EN 301 549

Die EN 301 549 ist eine harmonisierte europäische Norm, die Barrierefreiheitsanforderungen an Produkte und Dienstleistungen der Informations- und Kommunikationstechnik stellt.

Die Norm ist technologie-unabhängig formuliert und basiert auf den Anforderungen des Level AA der WCAG 2.1 (Bundesfachstelle Barrierefreiheit, [2025-04-25b](#)). Unter anderem verweist die Barrierefreie-Informationstechnik-Verordnung (BITV) 2.0 auf die EN 301 549, welche die Barrierefreiheitsanforderungen an die öffentliche Verwaltung des Bundes regelt (Bundesfachstelle Barrierefreiheit, [2025-04-25a](#)).

Die Barrierefreiheitsanforderungen für Apps stehen in der EN 301 549 unter Abschnitt 11 „Software“ und basieren auch zu großen Teilen auf den WCAG 2.1 Richtlinien („EN 301 549 V3.2.1 (2021-03): Accessibility requirements for ICT products and services“, [2021](#)).

Im Abschnitt [3.1](#) wird genauer auf die konkreten Anforderungen eingegangen.

2.4. Anforderung des BFSG

Die Anforderungen, die das BFSG an Produkte und Dienstleistungen stellt, sind in der europäischen Norm EN 301 549 zu finden. Allerdings ist diese nicht explizit im Gesetz genannt. Es wird lediglich in §4 auf „harmonisierten Normen oder Teilen davon, deren Fundstellen im Amtsblatt der Europäischen Union veröffentlicht worden sind“ verwiesen, wovon die EN 301 549 eine ist (§4 BFSG, [2021](#)).

Zudem wird in der Verordnung zum Barrierefreiheitsstärkungsgesetz (BFSGV) mehrmals die Anforderung gestellt, dass Produkte „auf konsistente und angemessene Weise wahrnehmbar, bedienbar, verständlich und robust gestaltet werden“ müssen (§ 12 Absatz 2h BFSGV, [2021](#)).

Dies sind genau die vier grundlegenden Prinzipien der WCAG und der EN 301 549. Somit wird indirekt auf die EN 301 549 als Anforderung verwiesen (Bundesfachstelle Barrierefreiheit, [n. d.](#)).

2.5. React Native

React Native ist ein open-source „cross-platform development Framework“ (CPDF). Es wird von Meta (ehemals Facebook) entwickelt. Mit React Native kann eine App mit einer Codebasis für iOS, Android und andere Plattformen exportiert werden.

React Native Apps nutzen native (plattform-eigene) User-Interface (UI) Elemente zur Darstellung des Interface und Javascript für die Logik. Dadurch sind die Apps deutlich performanter als solche, die Webviews für die UI benutzen (Meta Platforms, Inc, [n. d. b](#)). Die Gestaltung der UI funktioniert ähnlich wie in der Webentwicklung. Für viele HTML-Elemente hat React Native entsprechende Gegenstücke.

Durch die Verwendung von React Native kann viel Arbeit gespart werden, da eine App nur einmal für mehrere Plattformen programmiert werden muss. Allerdings hat sich bei der ursprünglichen Entwicklung gezeigt, dass Funktionen in React Native unzuverlässig sein können oder Workarounds brauchen.

Da React Native native UI Elemente nutzt, hängt die Barrierefreiheit stark vom Betriebssystem ab. React Native hat ein eigenes Application-Programming-Interface (API) über das die Accessibility APIs von Android und iOS angesprochen werden können (Meta Platforms, Inc, [n. d. a](#)).

React Native hat die Möglichkeit auch eigene native Komponenten zu erstellen. Dies ist nützlich um Funktionen zu nutzen für die React Native standardmäßig keine Unterstützung hat. Hierbei müssen Komponenten mit Java/Kotlin für Android und Swift/Objective-C für iOS programmiert werden. In dieser Arbeit werden keine eigenen nativen Komponenten erstellt, da hierfür zusätzliche Kenntnisse über die native App Entwicklung vonnöten wären.

Für diese Arbeit wird die React Native Version 0.76 verwendet. Als diese Arbeit begonnen wurde, war die Version 0.78 bereits veröffentlicht worden, welche jedoch keine wesentlichen Änderungen zur Barrierefreiheit enthalten soll (Meta Platforms, Inc, [n. d. c](#)).

2.6. Screenreader

Mit Screenreadern können visuelle Bildschirminhalte vorgelesen werden. So werden User Interfaces für Menschen mit Sehbeeinträchtigung zugänglich gemacht. iOS und Android haben Screenreader, die in das Betriebssystem eingebaut sind. Auf iOS nennt sich der Screenreader Voiceover, auf Android Talkback.

Screenreader werden auf Smartphones üblicherweise mit Touchscreen-Gesten bedient. Durch horizontales Wischen kann zum nächsten oder vorherigen Bedienelement navigiert werden und mit einem Doppeltippen wird ein Element betätigt. Bei der Auswahl eines Elements wird vorgelesen, um was für eine Art von Element es sich handelt, was es beinhaltet, was der aktuelle Zustand ist und wie es bedient werden kann. So können Anwendungen idealerweise komplett ohne eine visuelle Wahrnehmung verwendet werden.

Damit Elemente korrekt von Screenreadern erkannt werden können müssen ihnen bestimmte Werte zugewiesen werden. Beispielsweise die Eigenschaft `accessible`. Diese gibt an, ob ein Element auswählbar sein soll. Des Weiteren hat React Native unter anderem die folgenden Werte:

- `accessibilityLabel`: Beschreibung eines Elements, z. B. eine Bildbeschreibung.
- `accessibilityRole`: Art eines Elements, z. B. Button oder Schalter.
- `accessibilityState`: Der aktuelle Zustand, z. B. an/aus.
- `accessibilityActions`: Welche Aktionen dieses Element bereitstellt, z. B. eine Einstellung zu aktivieren.

Ein häufiges Problem in Apps ist, dass diese Werte nicht korrekt gesetzt sind. Wodurch Anwendungen teilweise oder komplett unbedienbar werden können (siehe Anhang Interviews).

2.7. Existierende App „Whatch“

Diese Arbeit basiert auf der App „Whatch“. Diese ist im Wintersemester 2023/24 im Rahmen des Projekt B an der HAW Hamburg, von Michelle Lamowski und Alexander Farwer, entstanden.

Die App soll die Suche nach Filmen erleichtern, wenn mehrere Streamingdienste gleichzeitig genutzt werden. Die abonnierten Streamingdienste können eingestellt werden (dritter Screenshot in Abbildung 2.2), wodurch die App nur Filme anzeigt, die bei den Diensten verfügbar sind (zweiter Screenshot in Abbildung 2.1). Dadurch muss nicht bei mehreren Anbietern einzeln gesucht werden.

Es kann in der App auch nach bestimmten Filmen gesucht werden, um herauszufinden, bei welchen Anbietern diese verfügbar sind (dritter Screenshot in Abbildung 2.1). Außerdem gibt es eine Watch-Liste und eine Seen-Liste zu der Filme hinzugefügt werden können (erster und zweiter Screenshot in Abbildung 2.2).

Bei der Erstellung der App wurde nicht auf die Barrierefreiheit geachtet, weshalb viele Elemente nicht mit einem Screenreader bedient werden konnten.

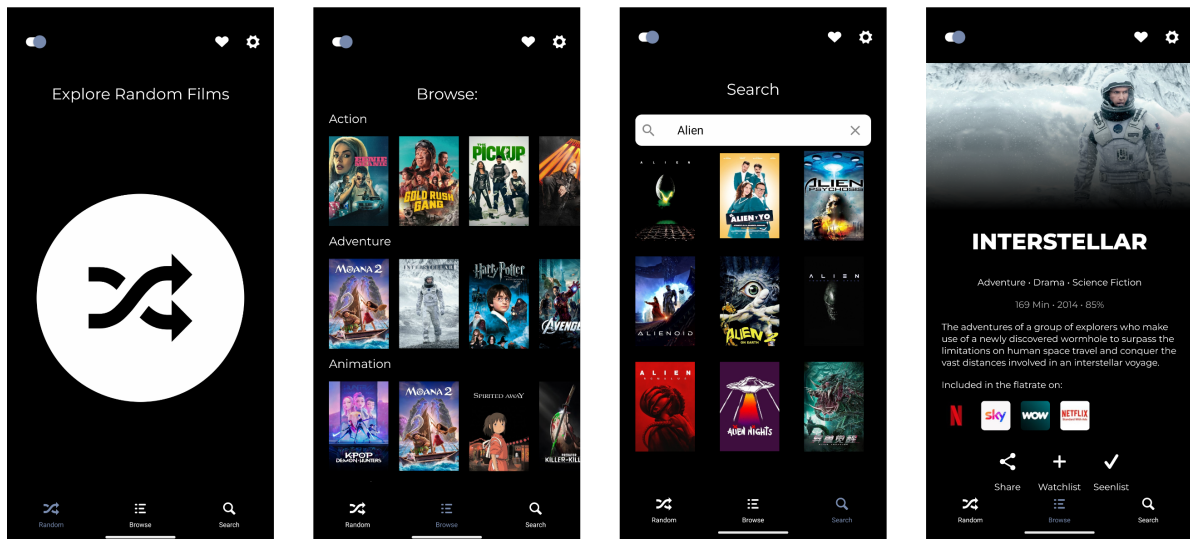


Abbildung 2.1.: App „Whatch“ Screenshots vor der Überarbeitung

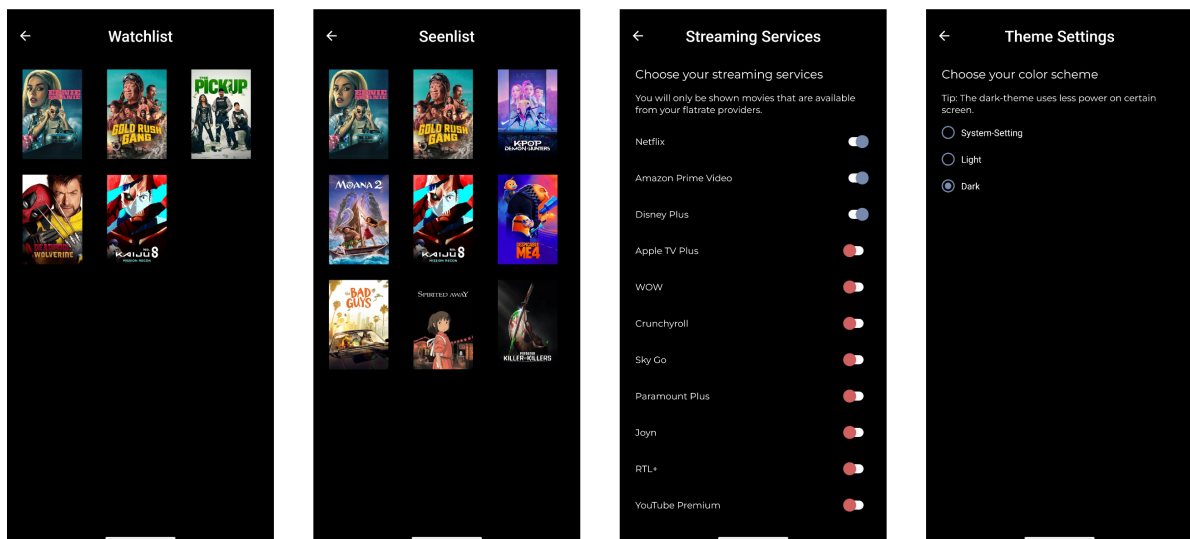


Abbildung 2.2.: App „Whatch“ Screenshots vor der Überarbeitung

2.8. Barrierefreiheit Testwerkzeuge

Um die Barrierefreiheit von Apps zu testen, haben Android und iOS eigene Tools.

Apple stellt für iOS den „Accessibility Inspector“ bereit. Dieser läuft auf macOS und kann mit iOS Geräten und Simulatoren verbunden werden, um Apps auf diesen Geräten zu testen. Mit dem Accessibility Inspector können einzelne UI Elemente untersucht oder ganze Seiten einer App in einem „Audit“ auf Probleme gescannt werden. Beispielsweise zeigt er auf, wenn Text zu klein ist oder - wie in Abbildung 2.3 - Text nicht skalierbar ist. Zusätzlich beinhaltet der Accessibility Inspector den „Color Contrast Calculator“ (Abbildung 4.4). Mit diesem kann der Kontrastwert zwischen Farben berechnet werden (Apple, n. d.).

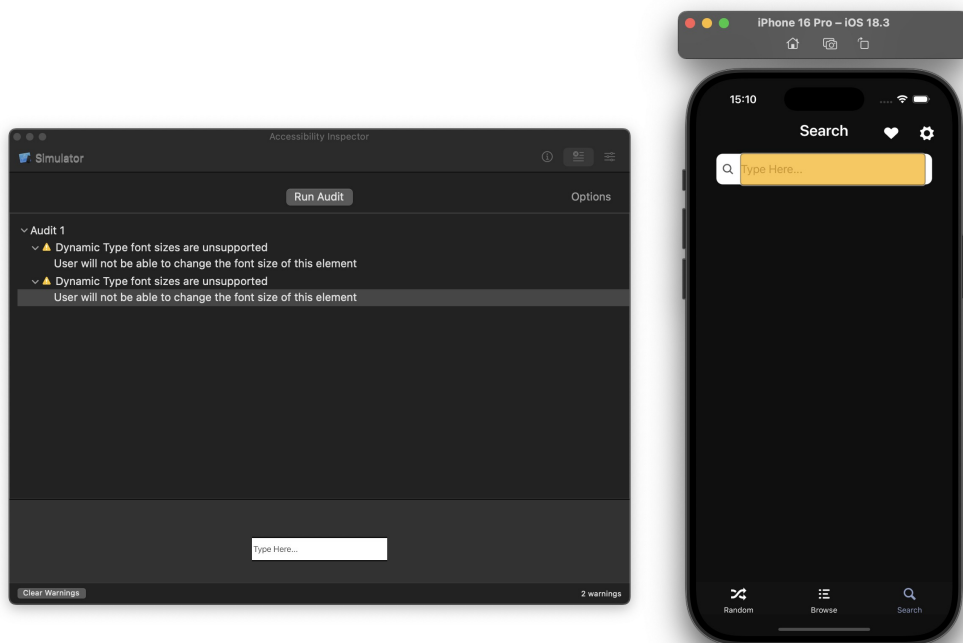


Abbildung 2.3.: Apple Accessibility Inspector Meldung eines Barrierefreiheitsproblems

Für Android gibt es die „Accessibility Scanner“ App von Google, die aus dem Play Store heruntergeladen werden kann. Mit dem Accessibility Scanner können „Snapshots“ und „Recordings“ ausgewertet werden. Snapshots sind einzelne Screenshots, auf denen markiert wird, welche Elemente nicht barrierefrei sind (Abbildung 2.4). Bei einem „Recording“ wird ein Interaktionsablauf durch mehrere Snapshots aufgenommen und diese Snapshots dann auf einmal ausgewertet (Google, n. d. a).

Beides sind nützliche Werkzeuge, allerdings können sie nur oberflächlichere Probleme finden,

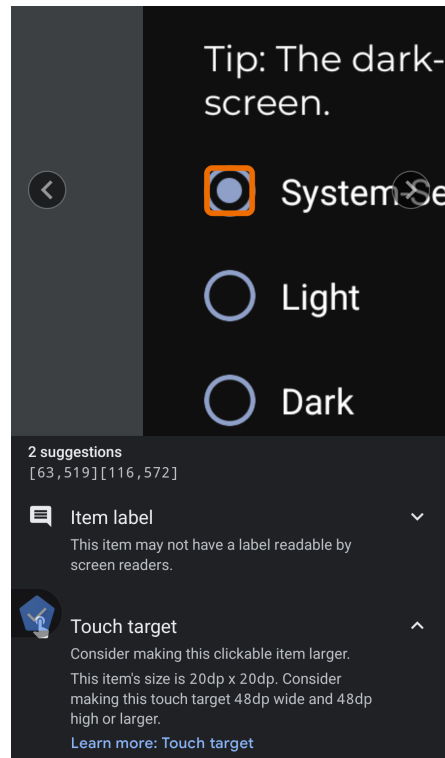


Abbildung 2.4.: Android Accessibility Scanner Meldung Radio Button nicht barrierefrei

wie zum Beispiel zu niedrige Farbkontraste oder zu kleine Buttons. Probleme wie schlechte Label oder umständliche User-Interfaces können sie nicht bewerten. Außerdem stellen sich gefundene Probleme teilweise als gar keine Probleme heraus. Der nicht skalierbare Text in Abbildung 2.3 konnte zum Beispiel doch in der Größe verändert werden. Daher muss die Barrierefreiheit auch immer von Hand getestet und bewertet werden.

2.9. Stand der Barrierefreiheit in Android Apps

In der Arbeit „The Current Status of Accessibility in Mobile Apps“ (2019) untersuchten Shunguo Yan und P. G. Ramachandran 479 beliebte Android-Apps aus dem Play Store. Sie stellten fest, dass nahezu alle Barrierefreiheitsprobleme hatten. Dies zeigt, dass trotz robuster Barrierefreiheitswerkzeuge in iOS und Android weiterhin deutliche Probleme in mobilen Apps bestehen. Es enthielten 94,8% der Apps eindeutige Verstöße gegen Barrierefreiheitsrichtlinien, 97,5% potenzielle Verstöße und 66,4% Warnungen.

Die Studie ergab, dass fünf UI Element-Typen (TextView, ImageView, View, Button und ImageButton) für ca. 92% der UI Elemente genutzt wurden und den Großteil der Probleme

verursachten. Die häufigsten Ursachen waren fehlende Fokusierbarkeit von Bedienelementen, unzureichende Label, zu niedrige Farbkontraste und zu kleine Schriftgrößen.

Zur Messung der Barrierefreiheit führten die AutorInnen neue Messwerte ein: die „Inaccessible Element Rate“ (IAER) und die „Accessibility Issue Rate“ (AIR). Die IAER ist der Anteil der UI Elemente mit Barrierefreiheitsproblemen aus allen Elementen einer App. IAR ist der Anteil der einzelnen Barrierefreiheitsverstöße aus allen möglichen Verstößen, die eine App haben könnte.

Durchschnittlich wiesen ca. 30% der UI Elemente einer App Barrieren auf (IAER) und 15% aller potenziellen Barrierefreiheitsprobleme blieben ungelöst (IAR). Diese Ergebnisse zeigen im Großen und Ganzen, dass Android Apps fast nie komplett barrierefrei sind (Yan & Ramachandran, 2019).

2.10. Entwicklung Barrierefreier mobiler Apps mit Cross-Platform Frameworks

Sergio Mascetti et al. (2021) haben in der Arbeit „Developing Accessible Mobile Applications with Cross-Platform Development Frameworks“ die Barrierefreiheits-APIs der CPDFs React Native und Xamarin untersucht. Mit Xamarin können, wie mit React Native, Apps für iOS und Android entwickelt werden. Es wurden die bereitgestellten APIs der Frameworks mit denen von iOS und Android verglichen.

Die Untersuchung umfasste 25 Funktionen, die für die Barrierefreiheit wichtig sind. Unter anderem Funktionen für die Lesbarkeit von UI-Elementen, Steuerung der Reihenfolge in der Screenreader den Fokus setzen, sowie die Ankündigung von Zustandsänderungen. Die Ergebnisse zeigten große Differenzen: React Native hatte nur für 13 von 25 Funktionen eine vollständige Unterstützung, während es bei Xamarin nur 7 von 25 waren.

Einige wichtige Funktionen, wie die Option, die Reihenfolge des Screenreader-Fokus zu bestimmen, fehlten in beiden Frameworks. Das hat zur Folge, dass manche Elemente für blinde NutzerInnen eventuell nicht zugänglich oder nur schwer verständlich sind.

Außerdem führten die AutorInnen einen Test zur Umsetzung einer Barrierefreiheitsfunktion durch. Dabei zeigte sich, dass die Implementierung einer fehlenden Funktion in React Native mit nativem Code über zehn Tage dauerte. Die Umsetzung integrierter Funktionen brauchte in der Regel nur Stunden. Dadurch wird deutlich, dass es sehr viel zusätzlichen Aufwand braucht, um Funktionen zu nutzen, die nicht direkt unterstützt werden.

Die Ergebnisse führten zu dem Schluss, dass CPDFs den Aufwand für plattformübergreifende App-Entwicklung grundsätzlich verringern, aber in Hinsicht auf Barrierefreiheit weniger

geeignet sind. Oft müssen EntwicklerInnen die nativen APIs von Android und iOS verwenden, um fehlende Funktionen zu implementieren, was eigentlich durch CPDFs vermieden werden soll (Mascetti et al., [2021](#)).

3. Konzept und Planung

Im nachfolgenden Kapitel soll die App Whatch barrierefrei weiterentwickelt werden. Zur Vorbereitung auf die Umsetzung werden in diesem Kapitel die konkreten Barrierefreiheitsanforderungen, Schwerpunkte, notwendigen Änderungen und das Vorgehen erläutert.

3.1. Anforderungen

3.1.1. EN 301 549

Die EN 301 549 verweist stark auf die WCAG 2.1 (Level AA), ergänzt aber auch spezifische Vorgaben für Software und mobile Apps. Die folgenden Punkte sind eine kurze Liste der wichtigsten Anforderungen, die teilweise auch für die anzupassende App „Whatch“ wichtig sind:

- 1. Wahrnehmbarkeit
 - Alle Inhalte, die kein Text sind, brauchen Alternativtexte (Abschnitt 11.1.1: Text alternatives).
 - Schriftgrößen müssen skalierbar sein (Abschnitt 11.1.4.4: Resize text).
 - Audio-/Video-Inhalte benötigen Untertitel oder Transkripte (Abschnitt 11.1.2: Time-based media).
 - Text darf nicht zu niedrige Farbkontraste haben ([2021](#), Abschnitt 11.1.4.3: Contrast (minimum)).
- 2. Bedienbarkeit
 - Alle Funktionen sind per Tastatur oder alternativer Eingabe (z. B. Schaltersteuerung, Sprachsteuerung) nutzbar (Abschnitt 11.2.1: Keyboard accessible).
 - Es darf keine zeitlich kritischen Eingaben geben ohne Anpassungsmöglichkeiten (z. B. Verlängern oder Abschalten von Timeouts) (Abschnitt 11.2.2: Enough time).

- Gesten (z. B. Wischen, Pinch-to-Zoom) müssen Alternativen haben (Abschnitt 11.2.5: Input modalities).
- Klare Fokus-Reihenfolge mit Unterstützungstechnologien: Elemente müssen logisch erreichbar sein (Abschnitt 11.2.4.3: Focus order).
- Inhalte müssen Header und Label haben, die sie beschreiben (Abschnitt 11.2.4.6: Headings and labels).
- 3. Verständlichkeit
 - Die Navigation und Bedienelemente sollen konsistent und berechenbar sein (Abschnitt 11.3.2.3).
 - Formulare und Eingabefelder brauchen Beschriftungen (Abschnitt 11.3.3.2: Labels or instructions).
- 4. Robustheit
 - Inhalte müssen von Unterstützungstechnologien wie Screenreadern erfassbar sein (z. B. durch Accessibility-Labels) (Abschnitt 11.4.1.2: Name, role, value).
 - Änderungen in der Benutzeroberfläche sind anzukündigen (z. B. durch Screenreader-Events) (Abschnitt 11.4.1.3: Status messages).
- Kompatibilität mit Unterstützungstechnologien
 - Software muss mit den Unterstützungstechnologien kompatibel sein, die das Betriebssystem bereitstellt (Abschnitt 11.5.2.3: Use of accessibility services).

Grob fordert die EN 301 549, so wie die WCAG, dass Software barrierefrei wahrnehmbar, vollständig bedienbar, verständlich aufgebaut und robust gegenüber Unterstützungstechnologien sein sollen.

3.1.2. Plattformeigene Richtlinien

Neben den gesetzlichen Vorgaben stellen Apple und Google auch eigene Richtlinien zur Verfügung. Apple hat in seinen „Human Interface Guidelines“ Richtlinien zur Barrierefreiheit festgelegt (Apple, 2025), während Google diese in den „Material Design Guidelines“ (Google, n. d. b) bereitstellt. Beide Richtlinien beinhalten Themen wie Farbwahl, Schriftgröße und Unterstützung von Barrierefreiheitstechnologien. Die von Apple und Google bereitgestellten Richtlinien sind jedoch nicht verpflichtend. Sie sollen lediglich das Entwickeln von barrierefreien und einheitlichen Apps vereinfachen.

3.2. Schwerpunkte bei der Umsetzung der Barrierefreiheit

Bei den Änderungen in der App liegt der Fokus vor allem auf allgemeinen Verbesserungen der UI sowie auf der Nutzbarkeit mit Screenreadern. Da die App keine Audioausgabe hat, fallen Anforderungen zu hörbaren Inhalten weg.

Außerdem ist die Bedienung einer App mit Schaltersteuerungen für Personen mit motorischen Einschränkungen ähnlich zur Navigation mit Screenreadern. Eine Verbesserung für Screenreader hilft deshalb auch dabei, die App für NutzerInnen mit motorischen Einschränkungen zugänglich zu machen.

3.3. Probleme in der existierenden App

Um die App barrierefrei zu gestalten, sind verschiedene Anpassungen notwendig. Die wichtigsten Probleme sind:

- **Unzureichende Farbkontraste:** Mehrere Textelemente und Symbole haben zu geringen Kontrast (mind. 4,5:1 für normalen Text, 3:1 für großen Text).
- **Zu kleine Textgrößen:** Teile der Texte sind zu klein dargestellt und daher schwer lesbar. Die EN 301 549 fordert selbst keine minimale Textgröße, Apples und Googles Richtlinien schon. Laut Googles Developer Guide sollte Text mindestens 12pt sein (Google, [2023](#)). Apples Human-Interface-Guidelines fordern 11pt (Apple, [2025](#)).
- **Zu kleine Bedienelemente:** Bestimmte Buttons sind nicht groß genug, um leicht bedient werden zu können. Die EN 301 549 fordert keine explizite Minimalgröße für Buttons, da die WCAG 2.1 Anforderung 2.5.5 „Target size“ nur auf Level AAA erforderlich ist. Diese fordert eine Mindestgröße von 44x44 CSS Pixeln. Google fordert eine Target Size von 48dp (Google, [2023](#)), und Apple 44pt (Apple, [2025](#)).
- **Nicht auswählbare Elemente:** Manche UI-Komponenten können von Screenreadern nicht fokussiert oder betätigt werden. Damit sind sie für NutzerInnen mit Sehbehinderungen nicht zugänglich.
- **Fehlende Labels für Screenreader:** Der Großteil der Bedienelemente verfügt nicht über Labels. Dies verhindert, dass Screenreader diese Elemente beschreiben können.
- **Keine dynamische Textanpassung:** Die App reagiert teilweise nicht korrekt auf System-Einstellungen für Schriftgrößen.

- **Fehlende Navigationsmöglichkeiten:** An einigen Stellen existiert kein klar erkennbarer „Zurück“-Button. Dies erschwert die Bedienung und widerspricht der Vorgabe konsistenter und vorhersehbarer Navigation.

Insgesamt war die App vor den Anpassungen nicht mit Barrierefreiheitstechnologien nutzbar und auch für Nutzer ohne Einschränkungen teilweise schwerer als notwendig zu bedienen.

3.4. Vorgehensweise

Bei der Weiterentwicklung der App wird zu jeder Änderung zunächst die zugrunde liegende Anforderung erklärt. Dabei erfolgte eine Zuordnung zu den relevanten Kriterien der EN 301 549 oder plattformeigenen Anforderungen. Im zweiten Schritt wird beschrieben, welches konkrete Problem in der App vorliegt und anschließend erläutert, wie das jeweilige Problem behoben wurde.

4. Umsetzung

In diesem Kapitel wird nun die praktische Umsetzung der Barrierefreiheit in der App Whatch beschrieben.

4.1. Navigation

4.1.1. Zurück Button

Anforderung

Nach der EN 301 549 ist es keine Anforderung an Software, dass die Bedienung einheitlich und logisch ist, es ist aber wünschenswert (2021, EN 301 549 Abschnitt 11.3.2.3 (Teil des Abschnitts Predictable)).

Problem

Üblicherweise haben Apps oben links immer einen Zurück-Button in der Menüleiste zum Navigieren (Abbildung 4.1). Die App sollte zusätzlich auch zwei Buttons in der Leiste haben (zu sehen in Abbildung 4.2 oben rechts). Um die Buttons an dieser Stelle zu positionieren, war es einfacher die „richtige“ Menüleiste auszublenden und stattdessen eine eigene zu erstellen. Wodurch der Zurück-Button nicht mehr angezeigt wurde. Zurück konnte dann nur noch durch die „Zurück-Wischgeste“ auf iOS und den universellen Zurück-Button auf Android navigiert werden.

Lösung

Damit alle Buttons angezeigt werden, muss die richtige Menüleiste wieder eingeblendet und die Funktion `headerRight` der Menüleiste genutzt werden, mit der zusätzliche Elemente rechts in der Leiste angezeigt werden können (Abbildung 4.3).

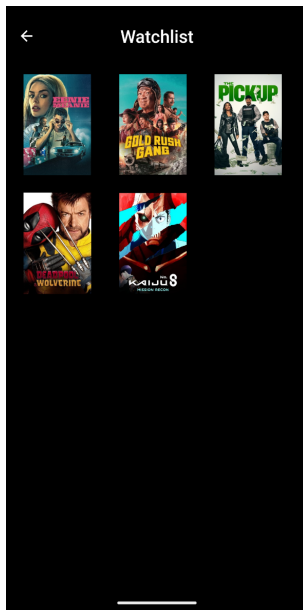


Abbildung 4.1.: Screenshot:
nur Zurück-
Button

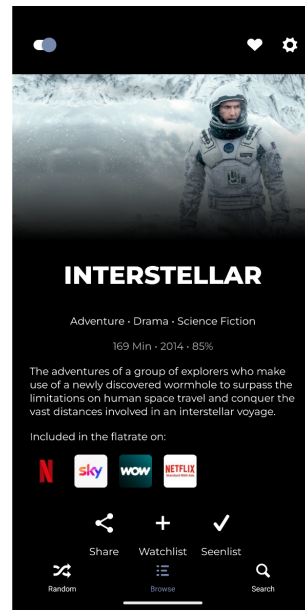


Abbildung 4.2.: Screenshot:
kein
Zurück-
Button

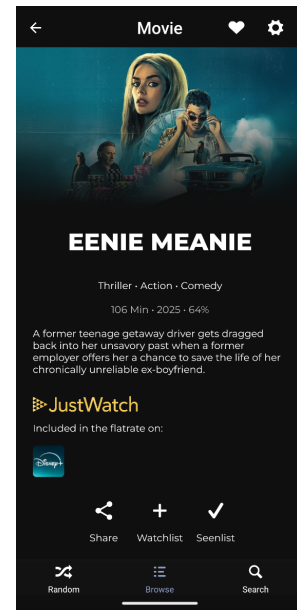


Abbildung 4.3.: Screenshot:
Buttons mit
headerRight
angezeigt

4.2. Kontrast

4.2.1. Anforderung

EN 301 549 Abschnitt 11.1.4.3 fordert, dass das WCAG 2.1 Erfolgskriterium 1.4.3: Contrast (Minimum) (Level AA) erfüllt sein muss. Inhalt dessen ist, dass Text ein Kontrastverhältnis von mindestens 4,5:1 zum Hintergrund haben muss. Ausnahmen sind großer Text, welcher nur ein Kontrastverhältnis von 3:1 haben muss, dekorativer Text und Text in Logos.

Damit Texte noch besser gelesen werden können, soll auch das WCAG Kriterium 1.4.6: Contrast (Enhanced) (Level AAA) erfüllt werden. Dieses verlangt ein Kontrastverhältnis von 7:1.

4.2.2. Problem

Die Farben der App werden zentral durch die Designfarben gesetzt. Es gibt ein helles (Light-theme) und ein dunkles (Darktheme) Design. In beiden hatte die blaue Akzentfarbe #798CB0 einen zu niedrigen Kontrast zum Hintergrund. Die Akzentfarbe wird zum Beispiel in der Navigationsleiste genutzt (Abbildung 4.4).

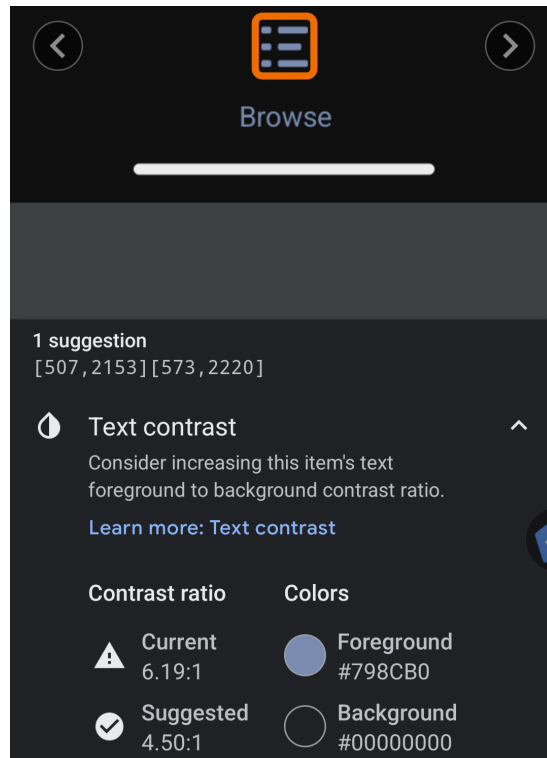


Abbildung 4.4.: Android Accessibility Scanner meldet Kontrastproblem

Vorherige Designfarben

■ Darktheme Hintergrund #0f0f0f

■ Darktheme Akzentfarbe #798CB0

Kontrastverhältnis: 5,65:1

□ Lighttheme Hintergrund #ffffff

■ Lighttheme Akzentfarbe #798CB0

Kontrastverhältnis 3,4:1

Zudem besaß auch die Suchleiste einen zu niedrigen Kontrast zwischen der Farbe der Suchleiste (Schwarz oder Weiß je nach Design) und der Platzhalterschrift. Die Platzhalterschrift hat einen Grauton, damit sie als solche zu erkennen ist (Abbildung 4.5).



Abbildung 4.5.: Screenshot: Suchleiste

4.2.3. Lösung

Damit die Akzentfarbe und der Platzhaltertext in beiden Designs ausreichend Kontrast zum Hintergrund haben, müssen sie je nach Design verschieden sein. Die Farben lassen sich gut mit Apples Color Contrast Calculator auswählen (Abbildung 4.6).



Abbildung 4.6.: Apple Accessibility Inspector meldet Kontrastproblem

Neue Designfarben:

■ Darktheme Hintergrund #0f0f0f

■ Darktheme Akzentfarbe #8FA0C9

Kontrastverhältnis: 7,34:1

□ Lighttheme Hintergrund #ffffff

■ Lighttheme Akzentfarbe #46587E

Kontrastverhältnis 7,1:1

Neue Farben in der Suchleiste:

□ Darktheme Suchleiste #feffff

■ Darktheme Platzhaltertext #595959

Kontrastverhältnis: 7,0:1

■ Lighttheme Suchleiste #0e0e0e
■ Lighttheme Platzhaltertext #a1a1a1
Kontrastverhältnis 7,4:1

4.3. Text- und Bedienelementgrößen

An einigen Stellen der App sind die Größen von Text und Bedienelementen zu klein.

4.3.1. Textgröße

Anforderung

Wie in Punkt 3.3 erläutert, geben Apple und Google eigene Empfehlungen zu Textgrößen, während die EN 301 549 keine Vorgaben dazu macht. Apple empfiehlt 11pt, Google 12pt. Außerdem muss Text in der Größe angepasst werden können. Dies wird auf Android und iOS in den Systemeinstellungen eingestellt und ändert die Schriftgrößen im ganzen Betriebssystem. Zu finden ist die Anforderung im WCAG 2.1 Erfolgskriterium 1.4.4: Resize Text (Level AA) (2025), welches nach EN 301 549 (Abschnitt 11.1.4.4: Resize text) erfüllt sein muss.

Problem

In der Navigationsleiste ist der Text 10pt groß und das Skalieren der Größe funktioniert nicht. Der Text ändert zwar seine Größe, allerdings wird er ab einer bestimmten Größe abgeschnitten, da sich die Höhe der Menüleiste nicht automatisch mit verändert (Abbildung 4.7).

Lösung

Der Text wurde auf Größe 12pt gesetzt. Damit der Text in der Menüleiste nicht abgeschnitten wird, muss sich die Höhe der Leiste zusammen mit der Textskalierung ändern. In React Native lässt sich der aktuelle Skalierungswert mit `PixelRatio.getFontScale` abrufen. Mit dem Wert kann die „normale“ Höhe der Menüleiste multipliziert werden und so an die Textgröße angepasst werden. Abbildung 4.8 zeigt die App mit deutlich größerer Textgröße.



Abbildung 4.7.: Screenshot:
Textskalierung
funktioniert nicht

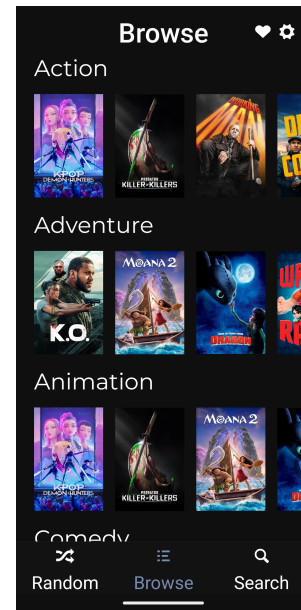


Abbildung 4.8.: Screenshot:
Funktionierende
Textskalierung

4.3.2. Bedienelementgrößen

Die meisten Buttons und Schalter sind nicht groß genug.

Anforderung

Wie in Punkt 3.3 erklärt, fordern die WCAG erst ab Level AAA eine minimale Größe für Bedienelemente, daher ist dies keine Vorgabe der EN 301 549. Gefordert sind Maße von mindestens 44x44 CSS Pixel (2025, WCAG 2.1 Success Criterion 2.5.5: Target Size (Level AAA)). Apples Empfehlungen sind 44 pt (2025, Human Interface Guidelines: Buttons), Google 48 dp (2023, Google Accessibility Guide: Design for motor skill).

Hierbei fällt auf, dass Google, Apple und die WCAG andere Einheiten verwenden. Googles dp und Apples pt werden von React Native gleich behandelt. Ein React Native Element mit einer Höhe von 48 ist auf iOS 48 pt hoch und 48 dp auf Android. Wie die CSS Pixel der Webentwicklung sich zu den pt und dp Einheiten verhalten, ist nicht in der EN 301 549 beschrieben. Dies ist nicht notwendig, da die europäische Norm keine konkreten Größenangaben für Apps angibt.

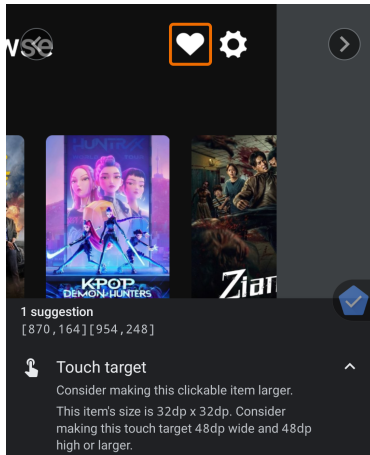


Abbildung 4.9.: Screenshot:
Schalter zu
klein

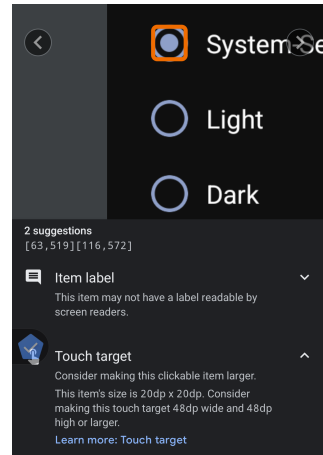


Abbildung 4.10.: Screen-
shot:
Radio
Button zu
klein

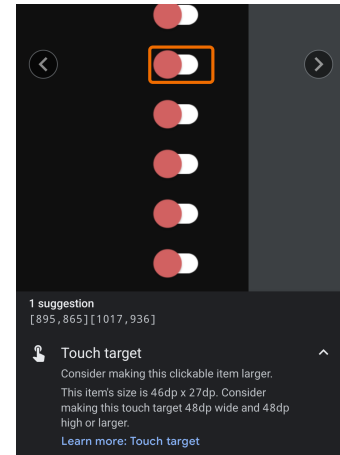


Abbildung 4.11.: Screen-
shot:
Switch zu
klein

Problem

Konkret sind die Buttons oben rechts in der Navigationsleiste (Abbildung 4.9), die Radio-buttons der Designeinstellungen (Abbildung 4.10), die Schalter für die Streaminganbieter (Abbildung 4.11) und die Suchleiste zu klein. Obwohl die Suchleiste sehr breit ist, ist die Höhe weniger als 44 px.

Lösung

Die Größe der Elemente wurde ursprünglich nach Ästhetik ausgesucht. Um dies beibehalten zu können wäre es wünschenswert, die antippbare Fläche unabhängig von der visuellen Größe verändern zu können. Hierfür gibt es in React Native die Eigenschaft `hitSlop`, welche auf Interaktionselemente angewendet werden kann. Leider hat `hitSlop` meistens nicht funktioniert oder es war nicht eindeutig, wie es richtig angewendet wird.

Die Suchleiste ist zum Beispiel eine fertige Komponente des React Native Elements Toolkits, welche aus vielen einzelnen Elementen besteht. Zur Vergrößerung der antippbaren Fläche müsste die `hitSlop` Eigenschaft des `TextInput`-Elements angepasst werden. In der Dokumentation von React Native Elements ist jedoch nicht beschrieben, ob und wie diese möglich ist. Daher wurde stattdessen die Höhe des kompletten `TextInput` Elements erhöht.

Für den Watchlist- und Einstellungsbutton konnte relativ einfach das verwendete `TouchableOpacity` Element größer gemacht werden, während die Größe vom Herz und Zahnrad Symbol gleich blieb, da diese Komponente selbst erstellt wurde.

Die Schalter für die Streaminganbieter in den Einstellungen waren am schwierigsten anzupassen. Laut React Native Dokumentation hat die `Switch` Komponente alle Eigenschaften der `View` Komponente. Demnach besitzt sie ebenso die Eigenschaft `hitSlop`. Leider hat das Ändern des `hitSlop` Wertes keinen Effekt auf die Bedienung des Schalters gehabt. Was funktioniert hat, ist die Höhe und Breite zu erhöhen, wodurch die anklickbare Fläche auf Android größer wurde, ohne dass der Schalter anders aussah. Auf iOS hatte dies jedoch keinen Effekt. Auf iOS ließ sich das Problem nur durch Vergrößern des gesamten Schalters lösen. Allerdings haben Schalter auf iOS eine feste Größe von etwa 31x51 pt, die überall gleich ist, wodurch sie bei einer Änderung etwas ungewöhnlich wirken.

Da keine von diesen Lösungen zufriedenstellend war, wurde verglichen, wie die Schalter in Apples eigenen Apps funktionieren. Das Ergebnis war, dass Apples Schalter, zum Beispiel in den Systemeinstellungen, selbst nicht die Anforderung erfüllen, mindestens 44 pt hoch und breit zu sein. In Abbildung 4.12 ist rechts ein Schalter aus den Systemeinstellungen zu sehen, links aus der App Watch. Beide haben nahezu gleiche Maße und die Schalter in den Einstellungen haben auch keine anklickbare Fläche, die über die Sichtbare hinaus geht.

Für Apple scheint es also in Ordnung zu sein, wenn Interaktionselemente in einer Richtung kleiner als 44 pt sind, solange die Fläche ähnlich groß ist. Dies steht jedoch nicht in Apples Anforderungen.

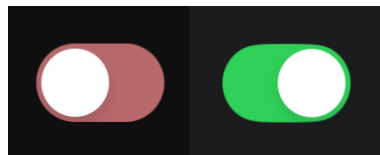


Abbildung 4.12.: Vergleich iOS Schalter

Zuletzt mussten die Radiobuttons der Designeinstellungen angepasst werden. Diese waren eine Komponente aus einer externen Komponentensammlung, wodurch die `hitSlop` Eigenschaft nicht zugänglich war. Um dies zu umgehen, konnten relativ einfach eigene Buttons nachgebaut werden, die gleich aussehen und funktionieren. Wieder hatte der `hitSlop` Wert jedoch keinen Effekt. Zur Lösung des Problems wurde stattdessen der übergeordnete Container, der sowohl den Radiobutton als auch die Beschriftung enthält, anklickbar gemacht, da diese Komponente die Größenanforderungen bereits erfüllte. So kann zusätzlich auch der Text für die Betätigung genutzt werden.

4.3.3. Suchleiste

Problem

Beim Schreiben in der Suchleiste wurde auf Android nach jedem Buchstaben der ganze Inhalt vorgelesen. Es gibt keine konkrete Anforderung für dieses Problem, da dies die Funktionalität nicht beeinträchtigt. Die Suchleiste wird dadurch lediglich benutzerunfreundlich. Dies ist ein weiteres Problem, das durch unterschiedliches Verhalten auf Android und iOS ausgelöst wird.

Ursache hierfür ist, dass die Suchleiste bei jeder Änderung automatisch nach dem eingegebenen Text suchen soll. Damit dies funktioniert, muss auf iOS der Inhalt der Suchleiste „controlled“ sein. Das bedeutet, der Inhalt der Suchleiste wird durch eine Variable gesteuert und diese wird bei jedem eingegebenen Buchstaben geändert. Auf Android hat dies den Effekt, dass der Screenreader bei jeder Änderung der Variable den Nutzer über diese Änderung informiert.

Lösung

Hierfür konnte keine andere Lösung gefunden werden, als zwei unterschiedliche Suchleisten zu erstellen. Für iOS bleibt die Leiste unverändert, für Android wurde die Variable entfernt, die den Inhalt steuert, da diese nur auf iOS notwendig ist.

4.4. Screenreader Bedienelemente

4.4.1. Buttons Navigationsleiste

Anforderung

Nach EN 301 549 Abschnitt 11.5.2.3: Use of accessibility services ([2021](#)) müssen Apps mit den Barrierefreiheitswerkzeugen einer Plattform kompatibel sein.

Problem

Die Buttons in der oberen Navigationsleiste (zu sehen in Abbildung [4.9](#)) können nicht mit Talkback auf Android betätigt werden.

Der Grund hierfür ist, dass die Buttons nicht die Standardfunktion `onPress` nutzen, um ihre Funktion auszulösen, sondern `onPressOut`. `onPressOut` funktioniert im üblichen Gebrauch genauso wie `onPress`. Diese Funktion musste genutzt werden, da die `onPress` Funktion nicht

funktioniert, wenn Buttons in einer Navigationsleiste eingebaut sind. Leider wird von Talkback die `onPressOut` Funktion aber nicht ausgelöst.

Dies ist ein bekanntes Problem, über welches bereits 2024 in einer Github Issue geschrieben wurde ([2024](#), stacyhigit: `onPress` with new architecture not called in navigation header). Allgemein scheint die `onPress` Funktion nie zu funktionieren, wenn sich ein Element in der Navigationsleiste befindet.

Lösung

Eine Lösung hierfür konnte erst durch langes Ausprobieren gefunden werden: Anstelle der `TouchableOpacity` Komponente von React Native wird die gleichnamige Komponente des Pakets `react-native-gesture-handler` verwendet. Diese bringt dieselbe Funktionalität mit sich und löst die `onPress` Funktion zuverlässig aus.

4.4.2. Schalter

Anforderung

Laut EN 301 549 Abschnitt 11.4.1.2: Name, role, value ([2021](#)) müssen von allen Elementen der Name und die Art des Elements durch Barrierefreiheitstechnologien erfasst und der aktuelle Zustand gesetzt werden können.

Problem

Die Schalter zum Einstellen der Streaminganbieter können nicht mit einem Screenreader genutzt werden. Auf iOS kann ein Anbieter nur im Ganzen – mit Namen und Schalter – ausgewählt werden. Da der Schalter nicht einzeln ausgewählt wird, ist es nicht möglich diesen zu betätigen.

Auf Android können die Schalter einzeln ausgewählt und betätigt werden. Wenn der Schalter ausgewählt ist, wird jedoch nicht vorgelesen, zu welchem Anbieter er gehört, da er kein Label hat.

Lösung

Damit ein Screenreader weiß, welche Elemente auswählbar sein sollen, müssen diese richtig konfiguriert sein. In React Native kann mit `accessible=false` oder `accessible=true` gekennzeichnet werden, ob ein Element auswählbar sein soll oder nicht. Außerdem gibt es noch viele weitere Werte, die steuern wie Elemente mit Screenreadern bedienbar sind.

Um die Schalter zugänglich zu machen, soll nur der übergeordnete Container, der den Schalter und den Anbieternamen beinhaltet, auswählbar sein. Über dieses Element soll auch der Schalter bedienbar sein.

Dafür müssen die Eigenschaften in Codeblock 4.1 gesetzt sein. Dadurch wirkt das Container-Element für Screenreader wie ein Schalter, der einen Zustand hat und betätigt werden kann.

Codeblock 4.1: Providerschalter View Accessibilityattributes

```
1 accessible={true}
2 accessibilityLabel={`Toggle provider ${providerLabel}`}
3 accessibilityRole="switch"
4 accessibilityState={{ checked: accessibilitySwitchState }}
5 accessibilityActions={[{ name: "activate", label: "Toggle" }]}
6 onAccessibilityAction={() => {
7   handleToggle();
8 }}
9 onAccessibilityTap={() => {
10   handleToggle();
11 }}
```

Zusätzlich müssen für den Schalter noch die Eigenschaften `accessible=false` und `importantForAccessibility="no"` gesetzt werden, damit dieser nicht auch einzeln ausgewählt werden kann. Die Eigenschaft `accessible=false` sollte laut React Native Dokumentation hierfür reichen. Oft funktionierte diese jedoch nicht wie erwartet und Elemente konnten dennoch ausgewählt werden.

Auch der Wert `onAccessibilityAction` funktionierte nicht wie erwartet. Der Wert sollte reichen, um Elementen eine mit Screenreadern ausführbare Funktion zu geben. Jedoch funktionierte dies nicht mit Voiceover auf iOS. Damit die Funktion auf iOS ausführbar ist, musste auch `onAccessibilityTap` gesetzt sein, was ein Wert explizit für iOS ist.

Alternativ wäre es auch möglich gewesen, ausschließlich den Schalter auswählbar zu machen und ihm ein eigenes Label zu geben. So hätte die Konfiguration des übergeordneten Containers als Schalter für Screenreader vermieden werden können.

4.4.3. Status von Bedienelementen

Anforderung

Der EN 301 549 Abschnitt 11.5.2.5: Object information ([2021](#)) besagt, dass der aktuelle Zustand von Bedienelementen mit Barrierefreiheitstechnologien erfassbar sein muss.

Problem

Wenn ein Bedienelement ausgewählt und betätigt wird, soll immer der aktuelle Zustand vorgelesen werden. Bei iOS wird jedoch teilweise nach dem Betätigen der vorherige Zustand vorgelesen statt dem neuen.

Dies ist mindestens seit 2023 ein bekanntes Problem (siehe Github Issue von samcarlinone, [2023](#)). Es tritt auf, wenn Elemente einen dynamischen accessibilityState Wert haben, der beim Betätigen geändert wird. Android wartet bis der Wert geändert wurde und liest den neuen vor. iOS kündigt den Wert direkt an, unabhängig davon, ob er schon geändert wurde oder nicht.

Lösung

Dieses Problem trat an verschiedenen Stellen auf und musste auf unterschiedliche Art gelöst werden, da die einfachste Lösung nur an einer Stelle funktioniert hat.

Für die Radiobuttons der Designeinstellungen reicht es, die dispatch Funktion mit einer 0ms Verzögerung im Hintergrund auszuführen. Die Funktion speichert die geänderte Einstellung. Warum dies funktioniert, ist nicht klar. Wie die dispatch Funktion ausgeführt wird, sollte eigentlich keinen Einfluss hierauf haben, solange der Zustandswert bereits vorher geändert wurde.

Für die Anbieter-Schalter und die Buttons zum Hinzufügen von Filmen zu Watch- und Seenlist funktionierte diese einfache Lösung leider nicht. Für diese musste die folgende aufwendige und nicht komplett zuverlässige Lösung erarbeitet werden, welche nur auf iOS ausgeführt wird.

Beim Betätigen der Elemente wird mit der Funktion announceForAccessibility(„Stop“) der Screenreader beim Vorlesen des falschen Wertes unterbrochen. Die Funktion ist eigentlich dafür da, um eigene Texte vorlesen zu lassen. Aus unerklärlichen Gründen wird beim Ausführen in dieser Situation weder der aktuell vorgelesene Wert noch der Text „Stop“ angekündigt und der Screenreader liest nichts vor.



Abbildung 4.13.: Screenshot:
Streaming-Anbieter
Icons

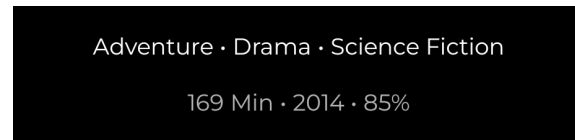


Abbildung 4.14.: Screenshot: Filmfakten
Bulletpoints

Daraufhin wird nach einer 500ms Verzögerung mit der gleichen Funktion `announceForAccessibility` der neue Wert des Schalters oder Buttons vorgelesen.

Diese Lösung ist aufwendig und unnötig komplex. Jedoch ist es der einzige Versuch der funktioniert hat, um den neuen Zustand der Elemente anzukündigen.

4.4.4. Streaminganbieter Icons

Anforderung

Nach EN 301 549 Abschnitt 11.1.1.1.1: Non-text content muss das WCAG Erfolgskriterium 1.1.1: Non-text Content (Level A) erfüllt sein. Diese besagt, dass alles, was kein Text ist, eine Beschreibung haben muss. Ausgenommen hiervon sind rein dekorative Elemente.

Problem

Die Streaminganbieter, bei denen ein Film verfügbar ist, sind jeweils dargestellt durch ein kleines Icon mit dem Logo des Anbieters (Abbildung 4.13). Diese brauchen wie alle Nicht-Text Inhalte ein Label.

Lösung

Um die Anforderung zu erfüllen, würde es reichen jedem Icon ein `accessibilityLabel` zu geben, durch den der Screenreader den Anbieter vorliest. Allerdings müsste jeder Anbieter einzeln ausgewählt werden, um alle vorgelesen zu bekommen.

Um dies zu vermeiden wurde – wie in Abschnitt 4.4.2 – der übergeordnete Container auswählbar gemacht und ihm ein zusätzlicher Text für den Screenreader gegeben. Sobald der Container ausgewählt wird, werden alle Anbieter auf einmal vorgelesen. Dies macht die App angenehmer zu nutzen.

Zusätzlich wurden auch den Elementen mit den Genres und Fakten eines Films (Abbildung 4.14) zusätzliche Beschriftungen für den Screenreader gegeben, damit unter anderem die Trennpunkte nicht mit vorgelesen werden.

4.5. Screenreader Navigation

4.5.1. Filmdetails Seite

Anforderung

Elemente müssen für einen Screenreader in einer sinnvollen und logischen Reihenfolge sein (2021, EN 301 549 Abschnitt 11.2.4.3: Focus order).

Problem

Damit beim Betrachten eines Films eine schnelle Navigation zum nächsten Film möglich ist, ist die Seite mit den Filmdetails (Abbildung 4.3) als eine horizontale Liste aufgebaut, die automatisch zu dem angeklickten Film scrollt. Dies ermöglicht beispielsweise, nach dem Öffnen des ersten Films der Watchliste durch die restlichen Filme zu gehen, ohne die Filmseite jedes Mal verlassen zu müssen. Dies funktioniert für sehende NutzerInnen gut.

Bei der Nutzung eines Screenreaders funktioniert die Navigation auf der Seite aufgrund der Liste jedoch nicht. Während des Öffnens der Seite ist die Seitenüberschrift „Movie“ ausgewählt. Wird dann zum Filmtitel navigiert, springt der Fokus zum ersten Film in der Liste, auch wenn ein anderer geöffnet wurde. Dass beim Navigieren durch eine Liste als Erstes das Element am Anfang ausgewählt wird, ist normalerweise auch das logische und erwartete Verhalten. In diesem Fall jedoch nicht.

Dieses Problem hätte bereits während der ursprünglichen Entwicklung der App beachtet werden sollen, da eine nachträgliche Umgestaltung der Seite ein großer Aufwand wäre. Es hätten zum Beispiel Buttons genutzt werden können, um durch die Filme zu wechseln, statt einer scrollbaren Liste.

Lösung

Um keine großen Änderungen an der App vornehmen zu müssen, wurde die Funktion, mit der man durch die Filme scrollen kann, bei der Nutzung mit einem Screenreader deaktiviert.

Dies wurde umgesetzt, indem für die Container aller Filme die folgenden Werte gesetzt wurden, außer für den Film, der tatsächlich angezeigt werden soll.

1. `accessible=false`
2. `importantForAccessibility=no-hide-descendants` (für Android)
3. `accessibilityElementsHidden=true` (für iOS)

Der Screenreader interpretiert es dann so, als ob der angeklickte Film der einzige ist, der angezeigt wird.

Hierbei hat sich jedoch ein neues Problem ergeben: Beim Öffnen eines Films auf iOS mit VoiceOver werden zwar die Informationen zum richtigen Film vorgelesen, auf dem Display ist jedoch der nachfolgende Film sichtbar. Dies fällt nicht auf, wenn ein vollständig blinder Nutzer die App nutzt, ist aber sonst irritierend.

Interessant ist, dass nur bei der Bedienung mit Voiceover der falsche Film angezeigt wird. Ohne Voiceover und auf Android passiert dies nicht, obwohl das Scrollen zum richtigen Film unabhängig vom Screenreader sein sollte. Eine Lösung hierfür konnte nicht gefunden werden.

4.5.2. Interface Änderungen

Anforderung

Nach EN 301 549 Abschnitt 11.5.2.15: Change notification müssen Änderungen des User-Interfaces angekündigt werden.

Problem

Die einzigen Stellen, an denen sich die UI unangekündigt verändert, sind, wenn die Daten über einen Film oder die Listen mit Filmen für die Browse-Seite geladen werden. Hier wird während des Ladens ein Ladesymbol angezeigt und danach der Inhalt. Diese Änderung wird über den Screenreader nicht angekündigt.

Lösung

Damit der Nutzer über diesen Vorgang informiert wird, ist während des Ladens der Ladekreis im Fokus. Sobald der Inhalt geladen ist, wird der Fokus auf das erste Element gesetzt. Für das automatische Fokussieren eines Elements mit dem Screenreader hat React Native die Funktion `setAccessibilityFocus()`. Generell ist diese sehr unzuverlässig, in diesem Fall hat sie aber funktioniert.

4.5.3. Automatisch mehr Filme laden

Anforderung

Nach EN 301 549 Abschnitt 11.5.2.3: Use of accessibility services müssen Apps mit Screenreadern bedienbar sein.

Problem

Auf der Browse-Seite werden anfangs nur 20 Filme pro Genre angezeigt und immer mit jeweils 20 weiteren ergänzt, sobald der letzte erreicht wird. Bei der Navigation mit einem Screenreader wird dies jedoch nicht automatisch ausgelöst.

Lösung

Um dies zu beheben, sollte erst der Ladeindikator am Ende der Liste mit dem Screenreader betätigt werden können, um mehr Filme zu laden. Allerdings ist, sobald die neuen Filme geladen waren, der Fokus des Screenreaders nicht auf den neuen Filmen gelandet. Es hat auch nicht funktioniert, den Fokus mit der Funktion `setAccessibilityFocus()` an die richtige Stelle zu setzen.

Durch Ausprobieren wurde herausgefunden, dass das automatische Laden mit Screenreadern funktioniert, wenn der Wert `onEndReachedThreshold` der Liste manuell auf zwei gesetzt wird. Der Wert bestimmt, wie nah ans Ende der Liste gescrollt werden muss bis automatisch neue Filme geladen werden. Laut Dokumentation soll der Standardwert bereits zwei sein, es funktioniert aber nur, wenn der Wert manuell auf zwei gesetzt wird (Meta Platforms, Inc, [n.d. a](#)).

4.5.4. Headlines

Anforderung

Das WCAG 2.1 Erfolgskriterium 1.3.1: Info and Relationships (Level A) von 2025 besagt, dass visuelle Strukturen der UI auch bei der Wahrnehmung durch Hilfsmittel erhalten bleiben müssen. Dafür müssen unter anderem Überschriften als richtige Headlines für den Screenreader angelegt sein.

Problem

Die Genre-Überschriften auf der Browse-Seite (Abbildung 4.8) sind für den Screenreader nicht als Überschriften zu erkennen, da diese nicht den Wert `Role=„heading“` zugewiesen haben. Dieser kennzeichnet Textelemente als Überschriften.

Üblicherweise gibt es im Web und in Textbearbeitungsprogrammen eigene Elemente für Überschriften mit Abstufungen wie H1, H2 und H3, welche automatisch als solche erkannt werden können. In React Native gibt es dies nicht. Für Titel und Überschriften muss ein normales Textelement angepasst und der Wert `Role=„heading“` gesetzt werden. Es ist ungewöhnlich, keine vorgefertigten Header-Elemente zu haben. Daher kann schnell vergessen werden, Textelemente extra zu markieren.

Dieses Problem wurde erst nach den Nutzertests behoben, da es erst bei diesen aufgefallen ist.

Lösung

Um das Problem zu lösen, muss nur der `Role=„heading“` Wert den Textelementen zugewiesen werden.

4.6. Kleinere Probleme

4.6.1. Label für die Suchleiste

Problem

Die Suchleiste braucht wie alle Elemente ein Screenreader-Label. Den Wert `accessibilityLabel` zu setzen, hat aber in diesem Fall nur dafür gesorgt, dass die Leiste ein Label auf iOS hat.

Auch nach längerer Recherche konnte der Suchleiste auf Android kein Label zugewiesen werden.

4.6.2. Einstellungselemente nicht auswählbar

Problem

Selten konnten nach längerer Nutzung der App die einzelnen Unterpunkte der Einstellungsseite (Abbildung 4.15) nicht mehr mit dem Screenreader ausgewählt werden. Für dieses Problem konnten weder eine Lösung noch ähnliche Fälle durch Internetrecherchen gefunden werden.

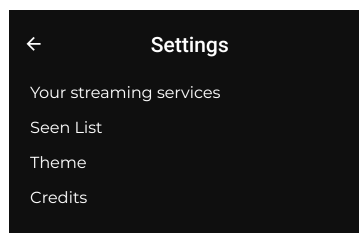


Abbildung 4.15.: Screenshot: Einstellungsseite

4.7. Fazit der Umsetzung

Bereits die ursprüngliche Entwicklung der Anwendung hat gezeigt, dass die Umsetzung gängiger Funktionen in React Native teilweise herausfordernd sein kann. Häufig sind Fehlermeldungen sehr vage und Bugs im Framework werden teilweise lange nicht behoben. Insgesamt ist die Entwicklung dennoch deutlich einfacher und schneller als zwei native Apps zu erstellen.

Bei der barrierefreien Gestaltung sind die Probleme deutlich größer. An vielen Stellen ist die Dokumentation, was Barrierefreiheit angeht, sehr ungenau und oft lässt sich auch wenig in Internetforen finden. Teilweise dauerte es länger, eine Funktion barrierefrei zu machen, als die ursprüngliche (nicht barrierefreie) Programmierung. Manche Probleme konnten gar nicht behoben werden.

Sehr frustrierend ist auch, dass Barrierefreiheitsfunktionen sich teilweise nicht so verhalten, wie in der Dokumentation beschrieben wird. Deshalb müssen oft Alternativen und Workarounds probiert werden, bis etwas funktioniert. Dies führt dazu, dass das Einschätzen vom Arbeitsaufwand deutlich schwerer ist.

Zusammenfassend lässt sich festhalten, dass die barrierefreie Entwicklung mit React Native erheblichen Mehraufwand mit sich bringt und an vielen Stellen durch unzuverlässige Funktionen und schlechte Dokumentation schwer gemacht wird.

5. Abschließende Nutzertests

5.1. Nutzertests zu Barrierefreiheit

Die Überprüfung der Barrierefreiheit umfasst mehr als nur die Beachtung technischer Vorgaben, da viele Anforderungen nur bei der tatsächlichen Nutzung getestet werden können. Automatische Tools wie Googles Accessibility Scanner oder Apples Accessibility Inspector überprüfen vor allem technische Vorgaben, wie zum Beispiel die Farbkontraste von Elementen oder das Vorhandensein von Labels. Ob die Inhalte sinnvoll und verständlich sind, lässt sich aber nur durch die tatsächliche Nutzung von Hilfsmitteln wie Screenreadern herausfinden.

Daher wurde die entwickelte App von zwei blinden NutzerInnen getestet, die als „ExpertInnen in eigener Sache“ Feedback geben konnten. Solche Nutzertests sind ein etablierter Teil des Überprüfes der Barrierefreiheit, da sie Barrieren sichtbar machen, die mit rein technischen Tests nicht gefunden werden würden (Hellbusch & Probiesch, 2011).

Die Tests sollten untersuchen, wie gut eine React Native App, die die gesetzlichen Anforderungen erfüllt, in Bezug auf die Barrierefreiheit nutzbar ist. Zudem sollte identifiziert werden, welche Dinge bereits gut umgesetzt wurden und an welchen Stellen es noch Verbesserungsbedarf gibt. Auch sollte eingeschätzt werden, wie gut die Anwendung im Vergleich zu anderen Apps wahrgenommen wird. Und es eröffnete die Möglichkeit einige offene Fragen, die beim Programmieren aufgekomen sind, zu klären:

1. Wie gut ist die Navigation der horizontalen Listen auf der Browse-Seite?
2. Wie würde es besser funktionieren auf der Detailseite zwischen den Filmen zu wechseln?
3. Ist es ein Problem, dass beim Öffnen der Watchliste nicht immer das erste Element auf der Seite den Screenreader-Fokus hat?

5.2. Testablauf

Es wurde eine Liste von Aufgaben erstellt, welche alle Funktionen der App abdeckt. Diese sollten die Testpersonen nacheinander erledigen. Danach haben die Teilnehmer in einem Interview Feedback gegeben. Außerdem wurden auch allgemeine Fragen zur Nutzung von Apps mit Screenreader gestellt. Zudem wurde erfragt, wie die Barrierefreiheit dieser App im Vergleich zu anderen steht.

Das erste Interview wurde mit einer Schülerin der siebten Klasse geführt, die ein iPhone nutzt und die App auf iOS testete. Das zweite Interview fand mit einem Schüler der 11. Klasse im Wirtschafts- und Informatikprofil statt. Er nutzt ein Android-Smartphone und hat auch Erfahrung mit iOS, so dass er die App auf beiden Plattformen testen konnte.

5.3. Beobachtungen

Während der Tests wurde beobachtet, wie die beiden Kandidaten die App mit VoiceOver und TalkBack nutzten. Aufgefallen ist dabei, dass Elemente häufig direkt durch Antippen ausgewählt wurden, anstatt sie Schritt für Schritt durchzugehen. Nur der zweite Kandidat nutzte die Möglichkeit durch Überschriften zu springen, um schneller zu Navigieren.

5.4. Zentrale Ergebnisse

Die beiden Tests haben gezeigt, dass die entwickelte Anwendung insgesamt sehr gut nutzbar ist. Alle Funktionen waren bedienbar und die App wurde als „viel barrierefreier als die meisten Apps von großen Unternehmen“ empfunden (Interview 2, Anhang A.2). In vielen anderen Apps sollen selbst einfache Dinge nicht umgesetzt werden, wenn zum Beispiel Schalter und Buttons keine Labels haben oder Elemente nicht richtig für Hilfstechnologien gekennzeichnet sind. Insbesondere Apps, die weniger bekannt sind, sollen häufig mehr Barrieren haben.

Einige Probleme konnten dennoch gefunden werden. Auf der Browse-Seite waren die Überschriften für die einzelnen Genres nicht als solche markiert, so dass ein schnelles Navigieren zu den Überschriften nicht möglich ist. Dieses Problem fiel jedoch nur einer der beiden Testpersonen auf. Zusätzlich wurde angemerkt, dass die horizontalen Listen auf der Browse-Seite umständlich zu navigieren sind.

Auch die Seite mit den Filmdetails wurde überwiegend positiv bewertet. Es wurde bestätigt, dass Buttons zum Blättern durch die Filme besser geeignet wären als eine scrollbare Liste. Die weiteren Seiten der App, wie die Zufallsseite, die Suchfunktion und die Watchliste wurden

auch als gut bedienbar eingeschätzt. Bei der Watchliste wurde darauf hingewiesen, dass der Screenreader-Fokus beim Öffnen der Seite zuverlässig am Anfang gesetzt werden sollte, um die Orientierung zu erleichtern.

Das Feedback der NutzerInnen bestätigt, dass die Barrierefreiheit in mobilen Anwendungen oft nicht vollständig umgesetzt wird. Die meisten Apps lassen sich jedoch mit etwas Erfahrung oder Probieren bedienen. Die größten Probleme sind, dass Apps häufig keine klare Struktur haben, Elemente nicht richtig gekennzeichnet sind und vor allem, dass manchmal Teile einer Anwendung gar nicht mit Hilfsmitteln bedienbar sind. Letzteres kann selbst bei Apps von großen Konzernen vorkommen. Es passiert immer wieder, dass bei der Einführung von neuen Funktionen und UI-Änderungen Barrieren unbeseitigt bleiben und erst in späteren Updates behoben werden. Abgesehen davon ist Barrierefreiheit in Anwendungen von großen Konzernen jedoch meist ordentlich umgesetzt.

5.5. Nutzertest Fazit

Die Ergebnisse der Nutzertests zeigen, dass eine Anwendung, die nur die gesetzlichen Anforderungen erfüllt, bereits überdurchschnittlich gut nutzbar ist. React Native hat sich in vielen Bereichen als umständlich für die Entwicklung barrierefreier Anwendungen erwiesen. Dennoch zeigte sich, dass mit genügend Zeit und Aufwand Apps entstehen können, die gut per Screenreader bedienbar sind.

6. Evaluation

Die durchgeführte Arbeit hat gezeigt, dass es grundsätzlich möglich ist, mit React Native barrierefreie Anwendungen zu entwickeln, die sogar überdurchschnittlich gut nutzbar sind. Entscheidend sind also weniger die verwendeten Tools und Frameworks, sondern eher die konsequente Umsetzung von Richtlinien und die Sicherstellung, dass tatsächlich alle Elemente bedienbar sind.

Gleichzeitig wurde deutlich, dass React Native nur eingeschränkt Unterstützung für Barrierefreiheitsfunktionen hat. Damit verbunden sind häufig unerwartete technische Probleme und höhere Planungsunsicherheit. Während React Native üblicherweise einen klaren Vorteil bietet, indem es die Entwicklung einer Anwendung für zwei Plattformen vereinfacht, relativiert sich dies im Kontext barrierefreier Apps. Die schlechte Unterstützung führt dazu, dass der Entwicklungsaufwand deutlich höher ist und potenziell viele eigene native Komponenten erstellt werden müssen.

Zur Entwicklung nativer Komponenten sind zusätzliche Kenntnisse über die native iOS und Android Entwicklung notwendig. Dadurch müssten EntwicklerInnen letztendlich drei verschiedene Arten der App-Entwicklung beherrschen – React Native, iOS und Android. Genau dies soll React Native eigentlich vermeiden.

Insgesamt ist festzuhalten, dass React Native dringend eine bessere Unterstützung für Barrierefreiheit benötigt, um weiterhin eine gute Option für die App-Entwicklung zu sein. Aufgrund der steigenden Bedeutung von Barrierefreiheit sollte dies möglichst schnell umgesetzt werden.

Darüber hinaus könnte es hilfreich sein, App Stores mit einer Metrik zur Barrierefreiheit auszustatten. Ähnlich wie bei den bereits existierenden Datenschutzübersichten im App Store und Play Store könnte durch automatische Verfahren (wie in Abschnitt [2.9](#) erläutert) und Nutzerbewertungen die Barrierefreiheit von Apps transparenter gemacht werden.

Literatur

- Abteilung DG des Bundesministeriums des Innern im Auftrag des Beauftragten der Bundesregierung für Informationstechnik. (2025a). *Barrierefreiheitsstärkungsgesetz (BFSG)*. Verfügbar 21. August 2025 unter <https://www.barrierefreiheit-dienstekonsolidierung.bund.de/Webs/PB/DE/gesetze-und-richtlinien/barrierefreiheitsstaerkungsgesetz/barrierefreiheitsstaerkungsgesetz-node.html>
- Abteilung DG des Bundesministeriums des Innern im Auftrag des Beauftragten der Bundesregierung für Informationstechnik. (2025b). *Lexikon: EAA*. Verfügbar 21. August 2025 unter <https://www.barrierefreiheit-dienstekonsolidierung.bund.de/Webs/PB/DE/service/lexikon/functions/bmi-lexikon.html?lv2=18267330&lv3=18581548#doc18581548>
- Accessibility Guidelines Working Group (W3C WAI). (2025). *Web Content Accessibility Guidelines (WCAG) 2.1: W3C Recommendation — 6 May 2025*. Verfügbar 21. August 2025 unter <https://www.w3.org/TR/WCAG21>
- Apple. (n. d.). *Accessibility Inspector*. Apple Inc. Verfügbar 23. August 2025 unter <https://developer.apple.com/documentation/accessibility/accessibility-inspector>
- Apple. (2025). *Human Interface Guidelines: MateThe HIG contains guidance and best practices that can help you design a great experience for any Apple platform*. Apple Inc. Verfügbar 30. August 2025 unter <https://developer.apple.com/design/human-interface-guidelines>
- Barrierefreiheitsstärkungsgesetz (BFSG)* [BGBl. I 2021, Nr. 46, S. 2970-2992]. (2021). Verfügbar 21. August 2025 unter <https://www.gesetze-im-internet.de/bfsg/>
- Bundesfachstelle Barrierefreiheit. (n. d.). *FAQ zum Barrierefreiheitsstärkungsgesetz (BFSG): Häufig gestellte Fragen und Antworten*. Verfügbar 26. August 2025 unter https://www.bundesfachstelle-barrierefreiheit.de/DE/Fachwissen/Produkte-und-Dienstleistungen/Barrierefreiheitsstaerkungsgesetz/FAQ/faq_node.html
- Bundesfachstelle Barrierefreiheit. (2025-04-25a). *Barrierefreie-Informationstechnik-Verordnung (BITV 2.0)*. Verfügbar 26. August 2025 unter <https://www.barrierefreiheit-dienstekonsolidierung.bund.de/Webs/PB/DE/gesetze-und-richtlinien/bitv2-0/bitv2-0-node.html>
- Bundesfachstelle Barrierefreiheit. (2025-04-25b). *Harmonisierte Europäische Norm (EN) 301 549: EN 301 549 ist die wichtigste Sammlung von einschlägigen Basrierefreiheitsanforderungen an die Informationstechnik (Web, Software, Hardware, mobile Anwendungen und Dokumente) der öffentlichen Stellen*. Verfügbar 26. August 2025 unter https://www.bundesfachstelle-barrierefreiheit.de/DE/Fachwissen/Produkte-und-Dienstleistungen/Barrierefreiheitsstaerkungsgesetz/FAQ/faq_node.html

- [//www.barrierefreiheit-dienstekonsolidierung.bund.de/Webs/PB/DE/gesetze-und-richtlinien/en301549/en301549-node.html](http://www.barrierefreiheit-dienstekonsolidierung.bund.de/Webs/PB/DE/gesetze-und-richtlinien/en301549/en301549-node.html)
- EN 301 549 V3.2.1 (2021-03): *Accessibility requirements for ICT products and services* [Harmonised European Standard]. (2021). Verfügbar 21. August 2025 unter https://www.etsi.org/deliver/etsi_en/301500_301599/301549/03.02.01_60/en_301549v030201p.pdf
- Google. (n. d. a). *Einführung in den Accessibility Scanner*. Google LLC. Verfügbar 23. August 2025 unter <https://support.google.com/accessibility/android/faq/6376582>
- Google. (n. d. b). *Material Design: Material Design 3 is Google's open-source design system for building beautiful, usable products*. Google LLC. Verfügbar 30. August 2025 unter <https://m3.material.io/>
- Google. (2023). *Android Developer Guides: Mobile Accessibility*. Google LLC. Verfügbar 1. Juli 2025 unter <https://developer.android.com/design/ui/mobile/guides/foundations/accessibility>
- Hellbusch, J. E., & Probiesch, K. (2011). *Barrierefreiheit verstehen und umsetzen: Webstandards für ein zugängliches und nutzbares Internet*. dpunkt. verlag.
- Mascetti, S., Ducci, M., Cantù, N., Pecis, P., & Ahmetovic, D. (2021). Developing Accessible Mobile Applications with Cross-Platform Development Frameworks. *Proceedings of the 23rd International ACM SIGACCESS Conference on Computers and Accessibility*. <https://doi.org/10.1145/3441852.3476469>
- Meta Platforms, Inc. (n. d. a). *React Native Documentation: Accessibility*. Meta Platforms, Inc. Verfügbar 27. August 2025 unter <https://reactnative.dev/docs/accessibility>
- Meta Platforms, Inc. (n. d. b). *React Native Frontpage*. Meta Platforms, Inc. Verfügbar 27. August 2025 unter <https://reactnative.dev/>
- Meta Platforms, Inc. (n. d. c). *React Native versions*. Meta Platforms, Inc. Verfügbar 3. September 2025 unter <https://reactnative.dev/versions>
- samcarlinone. (2023). Github Issue: iOS Accessibility Value Out of Sync [GitHub Issue #35774, accessed 2025-07-21]. Verfügbar 12. August 2025 unter <https://github.com/facebook/react-native/issues/35774>
- stacyhigit. (2024). Github Issue: onPress with new architecture not called in navigation header [GitHub Issue #29489, accessed 2025-08-23]. Verfügbar 19. Juli 2025 unter <https://github.com/expo/expo/issues/29489>
- Verordnung über die Barrierefreiheitsanforderungen für Produkte und Dienstleistungen nach dem Barrierefreiheitsstärkungsgesetz 1 (BFSGV)*. (2021). Verfügbar 21. August 2025 unter <https://www.gesetze-im-internet.de/bfsgv/>
- Yan, S., & Ramachandran, P. G. (2019). The Current Status of Accessibility in Mobile Apps. *ACM Trans. Access. Comput.*, 12(1). <https://doi.org/10.1145/3300176>

Anhang

Eigenständigkeitserklärung

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

Barrierefreiheit in React Native Apps

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

Hamburg, 30. September 2025

A. Interviews

Die nachfolgenden Transkripte beruhen auf den durchgeführten Interviews. Durch Anpassungen wurde die Lesbarkeit verbessert, aber die Inhalte und Aussagen entsprechen weiterhin den Originalaufnahmen.

Die Interviews wurden unmittelbar im Anschluss an die Tests geführt.

A.1. Interview 1

- Interviewer:** Hast du generell irgendwelches Feedback zu der App? Sachen, die dir jetzt schon einfallen?
- Teilnehmer:** Also an sich ist die App für mich relativ einfach zu bedienen. Da war jetzt nichts Großartiges, was schwierig sein könnte. Dass was wir hier alles ausprobiert haben.
- Interviewer:** Was findest du könnte man am ehesten verbessern? Ich war mir zum Beispiel sehr unsicher wie gut man die Entdecken-Seite bedienen kann, weil das so viele horizontale Listen sind, die nochmal in einer vertikalen Liste sind, und ich wusste nicht, wie man das strukturieren soll, damit das einfach ist, da durchzugehen.
- Teilnehmer:** Nee, das fand ich jetzt nicht so schwer. Also es dauert, davon mal ganz abgesehen, aber es ist nicht so schwierig.
- Interviewer:** Also wenn man ein bisschen rumprobiert, dann klappt das schon?
- Teilnehmer:** Ja genau, und wenn du nicht gucken kannst, musst du sowieso sehr oft rumprobieren.
- Interviewer:** Wie fandest du die Seite mit den Details über einen Film?

Teilnehmer: Relativ gut, würde ich mal sagen, weil dann hat man so ein bisschen einen Überblick, was man sich da gleich anschauen kann. Weil na ja, es gibt ja Leute, die das mit der Audiobeschreibung sich anhören, aber es gibt auch Leute wie mich und ich halte jetzt nicht so viel von der Audiobeschreibung, deswegen wäre es so ganz gut, wenn man so ungefähr wüsste, worum es geht, damit man sich vorbereiten kann.

Interviewer: Also ich meinte eher so das Interface. Zum Beispiel das Element, bei dem die Streaming-Anbieter vorgelesen werden, bevor ich das angepasst hatte, war das so dass jeder Streaming-Anbieter ein einzelnes Icon war, so dass man jedes Icon einzeln auswählen musste, damit es vorgelesen wird. Und ich hatte es jetzt so gemacht, dass dann das übergeordnete Element ausgewählt wird und dass alle Anbieter auf einmal vorgelesen werden.

Teilnehmer: Ja, das ist besser so.

Interviewer: Dann die Watchlist. Wie fandest du so die Struktur davon?

Teilnehmer: Auch ganz gut.

Interviewer: Ich hatte ein bisschen ein Problem damit als ich die barrierefrei machen wollte. Manchmal wenn man die Seite öffnet, war nicht den ersten Film ausgewählt, sondern teilweise der letzte Film oder die Überschrift oben. Ist das nervig, wenn das dann nicht der erste Film ist?

Teilnehmer: Also ich finde es besser, wenn die Überschrift ausgewählt ist. Und man sich dann selber durchklicken kann.

Interviewer: Ist das generell besser, wenn immer die Überschrift ausgewählt ist, damit man dann weiß wo man jetzt ist?

Teilnehmer: Ja, genau. Zu dem ersten Element kann man ja immer noch scrol-
len. Es wäre gut, wenn die Überschrift da ist, weil dann weißt du, habe ich das richtig jetzt geöffnet, oder bin ich woanders.

Interviewer: Wie fandest du die Einstellung von den einzelnen Streaming-Providern, die Schalter? Ich hab die so angepasst, dass der Schalter und die Beschriftung ein Element ist, das die Beschriftung und Einstellung von Schalter zusammen vorliest. Und man kann dann direkt den Schalter umschalten, wenn man doppeltippt.

Bei vielen anderen Apps ist mir aufgefallen, dass die Beschriftung und der Schalter getrennte Elemente sind und der Schalter selbst gar keine Beschriftung hat.

Teilnehmer: Ah ja, bei den meisten Apps steht da einfach nur „Taste“, also Voiceover liest dir das als „Taste“ vor, also da steht halt nicht, wofür der Schalter jetzt genau ist. Ich fand es besser, wenn der Schalter und der Text jetzt beide in einem sind.

Interviewer: Ist das meistens zusammen oder ist es meistens getrennt, dass es nur als Schalter vorgelesen wird?

Teilnehmer: Meistens getrennt.

Interviewer: Dann wollte ich noch fragen, wie wichtig ist das, dass man mit dem Screenreader ein Element nach dem anderen komplett so durch die Seiten der App gehen kann. Ich hatte gesehen, teilweise gehst du immer ein Element nach dem anderen vorwärts und zurück, aber teilweise wählst du auch direkt ein Element mit dem Finger aus. Ist das wichtig, dass man so einfach nach und nach da durchgehen kann oder ist das okay, wenn man etwas dann mal mit dem Finger direkt auswählen muss? Zum Beispiel, wenn man auf der Entdecken-Seite durch die Filme eines Genres geht, werden immer mehr Filme geladen bevor man das Ende erreicht. Dadurch erreicht man das Ende der Liste nie, und kommt auch nie zum nächsten Genre. Oder wäre es besser, wenn die Liste recht kurz ist und man dann zum nächsten Genre kommt.

Teilnehmer: Die zweite Version ist besser, dass man so von allem so ein bisschen was hat, weil sonst kommt man ja nie aus der Liste.

Interviewer: Also dann ist es vielleicht besser irgendwie einen Knopf zu haben, mit dem man noch mehr Filme laden kann und sonst geht man einfach weiter zur nächsten Kategorie.

Teilnehmer: Genau so ist es besser tatsächlich.

Interviewer: Wenn du die App einfach so heruntergeladen hättest, wärst du dann gut mit der zurechtgekommen?

Teilnehmer: Ich tatsächlich ja. Ich hab schon Schlimmeres bei Google erlebt. Die App ist harmlos im Vergleich. Alles super.

Interviewer: Dann hab ich noch so ein paar Fragen allgemein. Wie schwer ist das generell Apps zu benutzen, wenn man einfach irgendeine App im Store runterlädt? Geht das bei den meisten Apps so einfach oder ist das teilweise der Fall, dass das gar nicht funktioniert?

Teilnehmer: Also ich hatte noch nie Apps, bei denen es gar nicht funktioniert hat. Vielleicht hängt es auch davon ab, dass ich jetzt nicht so schwierige Apps mir runterlade, das sind eher so die ganz normalen, die fast jeder hat. Bisher ging es immer. Also du musst dich natürlich erstmal mit der App bekannt machen quasi und manchmal kann es sein, dass du ein bisschen verzweifelst, weil du so ein bisschen die ganze Zeit hin und her scrollen musst und so, aber an sich geht es dann irgendwann.

Interviewer: Wie schwer ist denn das so sich mit einer neuen App vertraut zu machen. So wie lange dauert das?

Teilnehmer: Ich gehe mal davon aus, das ist von Person zu Person unterschiedlich. Ich brauch meistens so die ersten 10 Minuten damit ich so ein bisschen mich damit bekannt machen kann, und nach und nach entdeckst du sowieso neue Sachen bei der App meistens. Du lernst nach und nach mehr dazu, das ist glaube ich bei den meisten auch so.

Interviewer: Aber man kriegt das schon hin, so nach 10 Minuten oder so kommt man ganz gut rein?

Teilnehmer: Ja, genau, die ersten Sachen kann man dann machen, kommt auch auf die App an.

Interviewer: Was würde dabei helfen eine neue App zu nutzen, um alle Features und Funktionen kennenzulernen?

Teilnehmer: Ja, die Struktur ist zum Beispiel sehr wichtig, dass alles nicht so durcheinander gemacht ist. Sondern eher so ein bisschen mehr Struktur halt. Damit das so ein bisschen alles seinen, wie sagt man, seinen richtigen Lauf bekommt, sag ich mal.

Interviewer: Also eine sinnvolle Struktur, dass alles Überschriften hat und die Sachen dazu passen.

Teilnehmer: Genau, ja.

Interviewer: Was sind so die größten Probleme, auf die man stößt in Apps, oder was nervt am meisten?

Teilnehmer: Das ist schwierig jetzt. Ich mein, diese Frage bekommst du nicht jeden Tag gestellt und irgendwann gewöhnst du dich auch an die Schwierigkeiten und dann sind die Schwierigkeiten gar nicht mehr so schwierig, weil du damit irgendwie umgehen musst. Deswegen muss man bei der Frage so ein bisschen länger überlegen. Aber ich glaub das größte Problem wäre zum Beispiel, wenn du eine App hast und die gar nicht richtig mit Voiceover funktioniert. Wenn zum Beispiel nicht alles angezeigt wird. Oder bei Instagram war es mal so, dass wenn man beim Anmelden einen Code per Mail gekriegt hat und man den eingegeben hat und auf weiter geklickt hat, konnte man dann nichts mehr machen und musste wieder zurückgehen. Das ist so glaube ich das größte Problem, das es manchmal bei Apps nicht direkt mit der Voiceover Funktion klappt.

Interviewer: Also generell, wenn Sachen überhaupt nicht mit Voiceover funktionieren?

Teilnehmer: Genau, nicht alles wird dir direkt angezeigt, wenn du mit Voiceover unterwegs bist. Und dann weißt du nicht was manche Sachen sind.

Interviewer: Ach so, die sind dann nicht richtig beschriftet?

Teilnehmer: Ja, erstens das und zweitens werden dir die meistens auch gar nicht angezeigt. Wenn, dann sind die nicht richtig beschriftet. Das ist das größte Problem, das ich entdeckt habe bisher.

Interviewer: Ah OK, also so Sachen, die dann gar nicht funktionieren, nicht beschriftet sind oder gar nicht sichtbar sind mit den Screenreader.

Teilnehmer: Ja.

Interviewer: Okay, und dann noch zum Schluss, wirkt das so, als ob die Barrierefreiheit oder jetzt in diesem Fall, das Nutzen mit Screenreader und Voiceover, bei Apps wirklich eine Priorität ist? Oder solange es irgendwie geht, reicht das. Ich hab jetzt aus deinen Antworten genommen, dass das bei den meisten Apps ja wahrscheinlich nicht wirklich so wirkt wie eine Priorität, oder? Dass

sich da nicht jemand wirklich groß Mühe gegeben hat. Also wieder zum Beispiel, dass Schalter oft kein Label haben.

Teilnehmer: Nee, so wirkt das bei den meisten Apps nicht.

Interviewer: Gibt es denn manche Apps, die so wirken, als ob sich wirklich jemand Mühe gegeben hat?

Teilnehmer: Also bei den bekannten Apps, die die meisten Leute benutzen, geht das irgendwie. Zum Beispiel wie YouTube oder WhatsApp oder so was. Da ist dann alles möglich, aber wo es ein bisschen schwieriger wird dann, sind glaub ich die Dienste, die weniger Leute benutzen. So kleinere Apps. Dann ist es für die meisten Firmen wahrscheinlich auch egal, solange es geht, weil die meisten Nutzer können ja gucken.

Interviewer: Ja, es wirkte für mich auch so als ich andere Apps getestet habe. Es waren zwar eigentlich alle benutzbar, aber es war selten mehr als das Minimum gemacht worden.

Teilnehmer: Ich glaub die meisten denken gar nicht daran.

Interviewer: Ok, dann vielen Dank, dass du mir geholfen hast.

A.2. Interview 2

- Interviewer:** Hast du generell noch Feedback außer, dass die Genres keine richtigen Header sind?
- Teilnehmer:** Nein, eigentlich nicht. Also ich finde so wie es ist gerade ziemlich gut.
- Interviewer:** So generell funktioniert also alles ok und das ist das einzige, was dir als schlecht aufgefallen ist?
- Teilnehmer:** Ja.
- Interviewer:** Zur der Browse-Seite wollte ich fragen, das ist ja eine vertikale Liste und dann sind da viele horizontale Listen drin. Ist das nervig zu navigieren, dass das so viele Listen in Listen sind?
- Teilnehmer:** Ne ich finde das gar nicht problematisch. Ich finde nur, dass man die Überschriften haben sollte, damit man schnell zu den Genres springen kann, sonst funktioniert das gut.
- Interviewer:** Wie fandest du die Seite, auf der die Details über einen Film sind?
- Teilnehmer:** Ich finde, sie ist ziemlich kompakt. Man findet alles, was man herausfinden will, es funktioniert alles ziemlich schnell, finde ich gut.
- Interviewer:** Ich hatte da nämlich auf der Seite zum Beispiel dem Element, in dem die Fakten über den Film stehen, einen eigenen Text für den Screenreader gegeben. Damit der nicht einfach den Text, der da steht, vorliest, denn da sind so Trennpunkte im Text und die braucht man ja nicht beim Vorlesen. Oder zum Beispiel das Element mit den Streaming-Providern ist ein übergeordnetes Element. Eigentlich ist das für jeden Anbieter ein kleines Icon und hätte ich denen nur einfach ein Label gegeben, dann müsste man jedes einzeln auswählen und jetzt funktioniert das so, dass alle auf einmal vorgelesen werden. Machen das viele andere Apps auch so, dass sie extra Texte für den Screenreader haben und nicht einfach den Text, den ein Element hat, vorlesen?
- Teilnehmer:** Apple macht das in ihren eigenen Apps ziemlich oft. Ja, das Ding ist, jeder macht irgendwie Sachen anders und meistens sind Apps nicht so barrierefrei wie deine jetzt. Aber je nachdem,

man muss sich überlegen, wie viele Informationen sind das, jetzt funktioniert das ziemlich gut, würde ich sagen. Aber überleg immer so, was möchte ich darstellen, wie viele Informationen sind das und wie wichtig sind sie.

Interviewer: OK, aber viele Apps machen sich nicht so die Mühe, dass sie extra Texte dafür haben?

Teilnehmer: Ne.

Interviewer: Würde ich die Funktion, dass man nach links und rechts durch die Filme gehen kann auch mit Screenreadern nutzbar machen wollen, sollte ich dann statt der Wischgeste eher einfach Knöpfe benutzen?

Teilnehmer: Ja, das wäre einfacher glaube ich, weil, ich kann das gar nicht so gut erklären, aber Voiceover funktioniert wie so eine Quehook. Also es erlaubt solche Wischbewegung nicht, weil es die selber braucht. Man kann zwar trotzdem scrollen aber Knöpfe wären einfacher.

Interviewer: Hast du irgendwelche Anmerkungen zur Random-Seite?

Teilnehmer: Ne, die ist ja nur ein Button und funktioniert gut.

Interviewer: Ist dir irgendetwas zur Suchseite aufgefallen?

Teilnehmer: Die ist auch richtig gut. Man hat einen Button den man anklickt und dann erscheint die Tastatur, man kann seine Sachen eintippen und die Suchergebnisse sind auch okay.

Interviewer: Wie fandest du die Watchliste und die Seenliste?

Teilnehmer: Die sind auch richtig einfach. Das sind ganz normale Listen.

Interviewer: Bei denen hatte ich das Problem, dass teilweise, wenn man die öffnet, nicht der erste Film ausgewählt war, sondern mal der letzte oder manchmal die Überschrift oben. Was wäre so das Beste, das ausgewählt sein sollte, der erste Film oder die Überschrift oben von der Seite?

Teilnehmer: Das ist gar nicht so wichtig. Wenn man ganz am Anfang fokussiert wird, dann funktioniert das gut. Also, wenn man ganz am Anfang der Seite startet.

Interviewer: Die andere Teilnehmerin meinte, dass sie findet der Titel sollte am besten ausgewählt sein, damit man weiß, wo man ist.

- Teilnehmer:** Ja das ist so das Standardverhalten für Apps, dass man ganz am Anfang platziert wird allgemein.
- Interviewer:** Die Providersettings waren auch gut meintest du. Dafür hatte ich auch eine Weile gebraucht die anzupassen. In vielen Apps, die ich getestet habe, ist mir aufgefallen, dass Schalter oft getrennt sind von der Beschriftung und keine Labels haben. Und wenn man den Schalter auswählt nicht angesagt wird, was er macht. Und dann muss man davon ausgehen, dass der Schalter zum Textelement von davor gehört.
- Teilnehmer:** Ja, das würde auch funktionieren. Also das wäre nicht so schlimm, aber für jemanden, der nicht so irgendwie technisch versiert ist, wäre das auf jeden Fall besser, wenn das Element ein Label hat.
- Interviewer:** Wie machen die meisten anderen Apps das, ist das meistens richtig beschriftet oder getrennt?
- Teilnehmer:** Mal so mal so, oft sind sie getrennt oder haben verschiedene Probleme, wo bestimmte Dinge nicht ganz richtig beschriftet sind. Und dann hat man so eine Checkbox, die keinen Namen hat.
- Interviewer:** Und dann muss man irgendwie das Label dazu finden?
- Teilnehmer:** Oder man weiß es einfach, was der Schalter macht. Wenn man etwas Erfahrung hat, dann weiß man so, ah, hier ist höchstwahrscheinlich das, was ich brauche. Aber wenn es so funktioniert, das ist auf jeden Fall besser.
- Interviewer:** Im Vergleich zu anderen Apps, wie barrierefrei würdest du diese einschätzen?
- Teilnehmer:** Also viel viel Barrierefreier als die meisten Apps von großen Unternehmen. Man erwartet, dass irgendwie die Apps zum Beispiel von Spotify oder YouTube richtig gut sind, aber die haben oft verschiedene Probleme, wo Interfaces irgendwie upgedatet werden und dann nicht mehr barrierefrei sind. Also es ist auf jeden Fall sehr schön, dass man einfach eine App hat, die funktioniert, wo man keine Probleme hat. Auf jeden Fall richtig gut.
- Interviewer:** Dann hab ich noch ein paar Fragen allgemein zu Apps. Wie schwer ist es generell so Apps blind zu nutzen. Wenn man

irgendeine App runterlädt, kann man die meistens einfach so benutzen?

Teilnehmer: Ja, wenn man sich ein bisschen auskennt, dann kriegt man das meistens hin. Heutzutage gibt es auch Image-Recognition und OCR. Man findet irgendwie einen Weg.

Interviewer: Und wie schwer ist es eine neue App zu lernen, wie lange dauert das so bis man da reinkommt?

Teilnehmer: Kommt darauf an wie groß die Apps ist, aber meistens so 10 bis 15 Minuten, man scrollt ein bisschen durch und ja.

Interviewer: Du hast auch schon gesagt, dass du so Features von Screenreadern benutzt wie Image-Recognition. Customized du den auch irgendwie, man kann ja zum Beispiel bestimmten Gesten eigene Funktionen zuweisen.

Teilnehmer: Nein, also man hat so Standardgesten und die reichen eigentlich.

Interviewer: Benutzt du auch dieses Feature, dass man sich mit AI Bilder beschreiben lassen kann?

Teilnehmer: Ja auf Sozialen Medien und so ist das sehr nützlich.

Interviewer: Und man kann sich ja auch die UI beschreiben lassen, nutzt du das?

Teilnehmer: Ja, also das ist momentan nicht so weit entwickelt, aber man kann es auf jeden Fall nutzen.

Interviewer: Was sind so generell in Apps die größten Probleme, die es so gibt?

Teilnehmer: Dass man keine richtigen Tags verwendet, keine Heading-Tags, keine richtigen Listen-Tags, dass Buttons nicht richtig beschriftet sind, so was. Dass Dinge einfach logisch aufgebaut sind. Dass man einfach nur so durch swipen kann und Sachen schnell findet. Und wenn alles richtig getagt ist, ist das einfacher.

Interviewer: Ah, also dass alles auch die richtigen Elemente nutzt, Headings sind Headings und Listen sind richtige Listen und so.

Teilnehmer: Genau

Interviewer: Wirkt es bei so den meisten Apps so, als ob sich da wirklich jemand Mühe gegeben hat, damit die gut nutzbar sind mit einem Screenreader, oder nur so viel, dass es gerade so funktioniert?

Teilnehmer: Das ist eine komplexe Frage. Also ich will sagen, mit den meisten Apps funktioniert das schon, also man merkt irgendwie, dass das schon ordentlich gemacht wurde, aber das ist schwer zu sagen, weil Apps ständig upgedatet werden. Und dann kommt eine neue Version raus und dann merkt man plötzlich, dass irgendwie eine Benutzeroberfläche überhaupt nicht richtig zu navigieren ist oder so und dann muss man halt warten, bis das gefixt wird und muss einen Bug-Report machen. Zum Beispiel Apples eigenen Apps sind meistens richtig gut. Googles eigene Apps sind normalerweise problemlos nutzbar, außer bei YouTube vielleicht, Microsofts auch. Von den großen Unternehmen sind die Apps meistens OK.

Interviewer: OK, dann danke für die Antworten.

Teilnehmer: Kein Problem