Entwicklung barrierefreier Apps auf iOS-Mobilgeräten

Motivation und Praxistipps

Autor: SBV-FSA/Technologie und Innovation/EK
Version: 31. Januar 2018

SBV
Schweizerischer Blinden- und Sehbehindertenverband

Inhaltsverzeichnis

1. Motivation

Warum braucht es barrierefreie Apps?
Mit einer barrierefreien App wird nicht nur die potenzielle Nutzerzahl der eigenen App gesteigert. Wie schon Bundesrat Alain Berset im Vorwort der Accessibility Studie 2016[1] anmerkt, ist die zunehmende Digitalisierung eine Chance für Menschen mit einer Behinderung. Die Digitalisierung kann vielen Menschen den Zugang zu öffentlichen und privaten Dienstleistungen öffnen. Warum? Damit alle Menschen ihre Talente entfalten und arbeiten können. Allerdings ist dies nur dann vollumfänglich möglich, wenn Apps auch barrierefrei sind.
Apple liefert gleich weitere Gründe für eine barrierefreie App:

- Die App kann benutzt werden, ohne dass man einen Blick auf das Display werfen muss, sprich Nutzer können die App mit der Sprachausgabe bedienen
- Accessibility Richtlinien werden eingehalten
- Weil es sich einfach richtig anfühlt

Im Gegensatz zu den Anfängen der Digitalisierung hält die Entwicklung barrierefreier Apps Softwareentwickler in keiner Weise davon ab, innovative und nutzvolle App zu entwickeln. Die Accessibility-Benutzeroberfläche nimmt weder Einfluss auf die Darstellung der App noch hat sie einen Einfluss auf deren Hauptfunktion.[2]

[1] Vgl. Zugang für alle, Accessibility-Studie, 2016, Vorwort

[2] Vgl. Apple, Accessibility-Understanding, 2012

2. Richtlinien zur Barrierefreiheit

Das World Wide Web Consortium (W3C) hat eine Initiative zur Barrierefreiheit gegründet. Diese beziehen sich aber in erster Linie auf mobile Webseiten. Einzelne Punkte[3] können aber auch auf iOS-Apps adaptiert werden:

1. **Nutzer können einfach navigieren, Inhalt finden und feststellen wo sie sich gerade befinden auf dem Screen**
    a. Controls, Tabellen etc. haben verständliche und beschreibende Titel
    b. Der aktuelle Fokus wird von der Sprachausgabe gekennzeichnet und gesprochen
    c. Einzelne Controls können mit der Sprachausgabe navigiert werden und die Fokus-Reihenfolge ist sinnvoll

2. **Text ist verständlich und lesbar für die Sprachausgabe**
   Dieser Punkt ist insbesondere wichtig für die Sprachausgabe, damit der Inhalt laut und verständlich vorgelesen werden kann
    a. Den Text so einfach wie möglich halten
    b. Die voreingestellte Sprache des Mobilgerätes wird durch die App erkannt

3. **Dynamische Anpassung der Textgrösse**
    a. Text kann vergrössert/verkleinert werden ohne, dass dabei Informationen verloren gehen
    b. Die Schriftgrösse der App passt sich der auf dem Mobilgerät eingestellten Schriftgrösse an
    c. Vorder- und Hintergrundfarbe weisen einen hohen Kontrast auf
    d. Farben werden nicht in erster Linie dazu gebraucht um Informationen zu vermitteln

4. **Text-Alternativen für Nicht-Text-Controls sind vorhanden**
    a. Bilder, Icons, Buttons und Grafiken weisen Text-Alternativen auf
    b. Daten, welche als Diagramme oder Illustrationen dargestellt sind, haben eine Beschreibung

[3] Für eine vollständige Übersicht der Punkte siehe W3G, Accessibility Principles, 2017

c. Detaillierte Beschreibung des Inhaltes eines Videos ist verfügbar. Optimal wäre eine Version mit Audio-Deskription.

5. **Nutzer haben genügend Zeit Inhalt zu lesen**
   a. Für das Ausführen bestimmter Aufgaben sind keine Zeitlimits gesetzt
   b. Blinkende Inhalte können pausiert oder gestoppt werden
   c. Bei Ablauf der Session kann sich der Nutzer nochmals Einloggen, ohne dass dabei seine Daten verlorengehen
   d. Damit Störungen mit der Sprachausgabe vermieden werden, sind Hintergrund-Töne leise oder können abgestellt werden
   e. Nutzer können Lautstärke einer Audio-Datei einstellen, pausieren oder stoppen

Wie bereits oben erwähnt sind die Richtlinien des W3C nur für mobile Webseiten gedacht. Daher gehen wir in diesem Artikel nicht mehr darauf ein und fokussieren uns vielmehr auf die Leitfäden, Richtlinien und Vorgaben von Apple zur Barrierefreiheit.
Denn es braucht noch zusätzliche Informationen wie wir zum Beispiel die Standard-Controls von Apple benutzen oder die Reihenfolge der gesprochenen Controls selbst bestimmen können.

Anbei eine Übersicht der iOS Richtlinien:

1. Übersicht „Accessibility on iOS“
   https://developer.apple.com/accessibility/ios/

2. Richtlinien Barrierefreiheit auf dem iPhone
   https://developer.apple.com/library/content/documentation/UserExperience/Conceptual/iPhoneAccessibility/Introduction/Introduction.html

3. Testen der Barrierefreiheit einer App unter iOS
   https://developer.apple.com/library/content/technotes/TestingAccessibilityOfiOSApps/TestingtheAccessibilityofiOSApps/TestingtheAccessibilityofiOSApps.html#//apple_ref/doc/uid/TP40012619

4. Apple-WWDC-Video zur Programmierung einer barrierefreien App
   https://developer.apple.com/videos/play/wwdc2015/201/

Im dritten und vierten Teil werden diese Richtlinien aufgegriffen, erklärt wie die Sprachausgabe auf dem iOS-Mobilgerät funktioniert und wie eine barrierefreie App programmiert werden kann.

3. Barrierefreiheit auf dem iOS

VoiceOver ist Apple's innovative Sprachausgabe. Mit VoiceOver hat eine sehbehinderte Person die Möglichkeit, Kontrolle über den iPhone-Bildschirm zu haben, ohne dass er diesen sehen muss. Dabei fungiert die Sprachausgabe als eine Art Schnittstelle zwischen der graphischen Benutzeroberfläche und den Berührungen des Nutzers. Die Sprachausgabe liest dabei den Bildschirminhalt vor und gibt Informationen über weiterführende Aktionen. So läuft ein sehbehinderter Nutzer nicht Gefahr aus Versehen einen Kontakt zu löschen oder einen Anruf zu tätigen. Er weiss zu jedem Zeitpunkt Bescheid an welcher Stelle im Screen er sich gerade befindet. Allerdings nur, wenn die App auch barrierefrei ist.

Eine App wird als barrierefrei bezeichnet, wenn auch alle Controls, mit denen der Nutzer interagieren kann, barrierefrei sind.
Controls sollten im Idealfall genaue Angaben über deren Position, Namen, Verhalten, Wert und Typ machen. Denn diese Informationen werden von VoiceOver vorgelesen. Die SDK des iOS beinhaltet eine Programmierumgebung und Werkzeuge, um sicherzustellen, dass eine barrierefreie App programmiert werden kann.

Die im nachfolgenden Kapitel vorgestellten Hilfsmittel sollte jeder Entwickler während der Entstehungsphase seiner App einschalten und testen. Nur so kann er sich einen Einblick darüber verschaffen wie Menschen mit einer Sehbehinderung seine App wahrnehmen. Aus den gewonnenen Erkenntnissen kann die App dann verbessert und Barrieren für sehbehinderte Personen abgebaut werden.

3.1 Sprachausgabe und andere Hilfsmittel auf dem iPhone

Die kleinere Bildschirmgrösse auf Mobilgeräten zwingt Entwickler zu einer klareren Strukturierung der Inhalte auf dem Screen. Dies sollte die Navigation erleichtern, insbesondere für Menschen mit einer Sehbehinderung. Menschen mit einer Sehbehinderung können eventuell nicht alle Controls auf der Benutzeroberfläche sehen oder erkennen. Dies ist aber Voraussetzung um die App korrekt bedienen zu können. Damit wir uns in

die Lage von sehbehinderten Nutzer versetzen können, bieten sich auf dem iPhone folgende Möglichkeiten an:[4]

3.1.1 Sprachausgabe (VoiceOver)

Wird VoiceOver mittels *„Einstellungen/Allgemein/Bedienungshilfen/VoiceOver“* eingeschaltet, kann sich der Nutzer Inhalte auf dem Screen vorlesen lassen. Fährt er mit dem Finger auf den Button, mit dem Label „speichern“, so wird ihm der Text „speichern/Taste“ vorgelesen. So weiss er, dass mit diesem Control weiterführende Aktionen durchgeführt werden können.

Abbildung 1: VoiceOver auf dem iPhone aktivieren

3.1.2 Farben umkehren / Graustufen

Werden die Displayfarben mittels *„Einstellungen/Allgemein/Bedienungshilfen/Display-Anpassungen/Farben umkehren“* umgekehrt so werden weisse Farbflächen zu schwarzen Farbflächen umgewandelt. Diese Funktion ist für diejenigen Nutzer gedacht, welche Farbenblind sind oder Probleme mit dem Kontrast haben. Diese Einstellungen helfen, den Standort von Inhalten und Controls einfacher zu erkennen.

[4] Vgl. Vollmer G., [App Engineering, 2017]: Mobile App Engineering, S. 132

3.1.3 Lupe.

Mit der Lupenfunktion „Einstellungen/Allgemein/Bedienungshilfen/Lupe/“ werden sämtliche Inhalte auf dem Bildschirm vergrössert. Die Kamera wird dann zum Vergrösserungsglas. [5]

3.1.4 Grösserer Text

Unter *„Einstellungen/Allgemein/Bedienungshilfen/Grösserer Text/“* kann die Textgrösse angepasst werden. Apps, welche dynamischen Text unterstützen, passen sich dann an die festgelegte Schriftgrösse an.

Abbildung 2: Einstellen eines grösseren Texts auf dem iPhone

3.2 Die iOS-API

Seit der iOS Version 3.0 liefert Apple eine API mit der man sämtliche Informationen bereitstellen kann, um die Benutzeroberfläche für sehbehinderte Menschen zu beschreiben.
Die Programmierschnittstelle ist Teil des UIKIT und standardmässig in den UIKIT-Controls verfügbar. Das heisst, dass beim Einsatz von Standard-Controls die meiste Arbeit bereits von Apple übernommen wird. Die iOS-SDK bietet folgende Hilfsmittel, um eine barrierefreie App zu programmieren und zu überprüfen:

[5] Vgl. Semler J., [App Design, 2016]: App-Design, S. 150ff

1. **Interface-Builder**, in der eingestellt werden kann (Per Default ausgewählt) ob das GUI-Control barrierefrei ist oder nicht. In Kapitel vier gehen wir näher darauf ein.

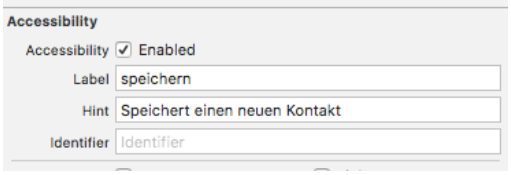

Abbildung 3: Accessibility-Inspector im Einsatz

2. Den **Accessibility Inspector**, welche sämtliche Controls anzeigt die barrierefrei sind. Die Validierung erfolgt direkt in XCode, nach Ausführen der App im Simulator.

3. Einschalten von **VoiceOver** auf dem Mobilgerät, Ausführen der App und Kontrolle wie sich die App verhält.[6]

3.3 Testen der Barrierefreiheit auf einem Gerät mit Sprachausgabe

Das Testen einer App auf Barrierefreiheit bietet einem Entwickler die Möglichkeit die App aus Sicht eines sehbehinderten Menschen zu erfahren. Man muss kein Experte von VoiceOver sein, eine Handvoll Gesten reichen dabei aus, um die App mit eingeschalteter Sprachausgabe zu bedienen.

3.3.1 VoiceOver Gesten

Gehe zu *„Einstellungen/Allgemein/Bedienungshilfen/VoiceOver“* und tipp auf „VoiceOver“ um die Sprachausgabe zu aktivieren. Falls „Hints“ im Storyboard eingestellt wurden, vergewissern ob unter „Ausführlichkeit“ die Option „Hinweise sprechen“ aktiviert wurde (Per Default ausgewählt). Auch kontrollieren ob die Sprechgeschwindigkeit angemessen ist.
Nach dem Aktivieren wird klar, dass bekannte Gesten einen anderen Effekt haben. Zum Beispiel wird beim einmaligen tippen das ausgewählte Control vorgelesen und beim zweimaligen tippen das ausgewählte Control ausgewählt. Anbei eine Auswahl an Gesten und deren Bedeutung:

[6] Vgl. Apple, Accessibility-Understanding, 2012

VoiceOver Geste	Aktion
Mit einem Finger über den Screen fahren	Selektiert und spricht jedes ausgewählte Control (Nur diejenigen, welche auch als barrierefrei gekennzeichnet wurde)
Mit einem Finger tippen	Das selektierte Control wird gesprochen
Mit einem Finger nach rechts oder links streichen	Das nächste oder vorherige Control auswählen. Die Reihenfolge ist abhängig von den Screen-Koordinaten (Links nach rechts/Oben nach Unten) der einzelnen Controls.
Doppeltipp mit einem Finger	Eines der folgenden Aktionen wird ausgeführt: - Aktiviert das ausgewählte Control - Schaltet zwischen den Werten eines Switches hin und her

Alle weiteren Gesten sind auf folgender Webseite im Detail beschrieben: https://www.apple.com/de/voiceover/info/guide/_1131.html - vo27992

3.3.2 VoiceOver im Einsatz

Ab Minute 12:20 erklärt IAN Fisch (Engineer Accessibility Team) im folgenden Video wie die VoiceOver-Gesten funktionieren: https://developer.apple.com/videos/play/wwdc2015/201/

Eine tolle Möglichkeit die Sprachausgabe-Gesten zu trainieren bietet Apple gleich selbst. Auf dem iPhone können unter *„Einstellungen/Allgemein/Bedienungshilfen/VoiceOver/VoiceOver-Übungen"* Übungen gestartet werden. Das iPhone reagiert dabei auf jede Fingerbewegung und liest vor was sie zu bedeuten haben. Wird zum Beispiel mit dem Finger nach rechts gestrichen, wird der Text „Nach rechts streichen – Zum nächsten Objekt bewegen" gesprochen. Lohnt sich, einfach mal ausprobieren.

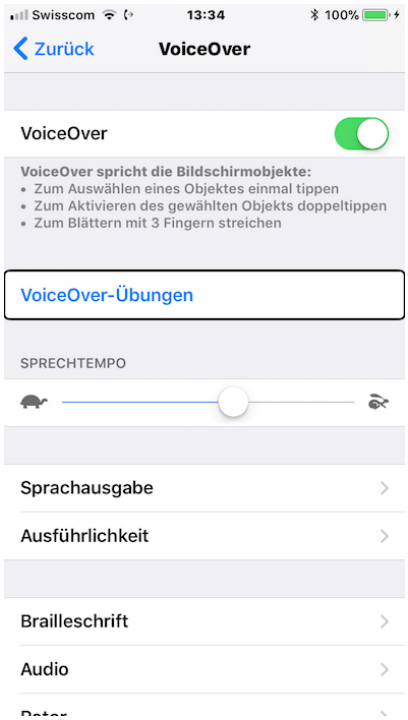

Abbildung 4: Mit „VoiceOver-Übungen“ kann die Sprachausgabe geübt werden

3.3.3 VoiceOver Tipps und Tricks

VoiceOver schnell ein- und ausschalten
Alternativ kann VoiceOver auch durch Dreifachklicken der Home-Taste ein- und abgeschaltet werden. Eingestellt wird dies unter *„Einstellungen/Allgemein/Bedienungshilfen/Kurzbefehl/VoiceOver*“. Dies ist wesentlich schneller als VoiceOver immer über den Menüpunkt „Einstellungen“ einzustellen. Falls man sich nicht sicher ist welche Geste welche Bedeutung haben, kann so VoiceOver einfach wieder ein- und abgeschaltet werden.

Schneller Scrollen mit der Objektauswahl
Mit der Objektauswahl kann schnell auf gesuchte Controls zugegriffen werden. Er wird mittels „Dreifachtippen mit zwei Fingern“ auf den Screen aktiviert. Anschliessend werden in einer Tabelle alle verfügbaren Controls aufgelistet. Die Liste kann durchsucht oder alphabetisch sortiert werden. Die Objektauswahl wird mit der „Mit zwei Fingern eine Zickzackbewegung ausführen“-Geste wieder geschlossen. Anbei ein Screenshot der App eKiosk mit und ohne Objektauswahl:

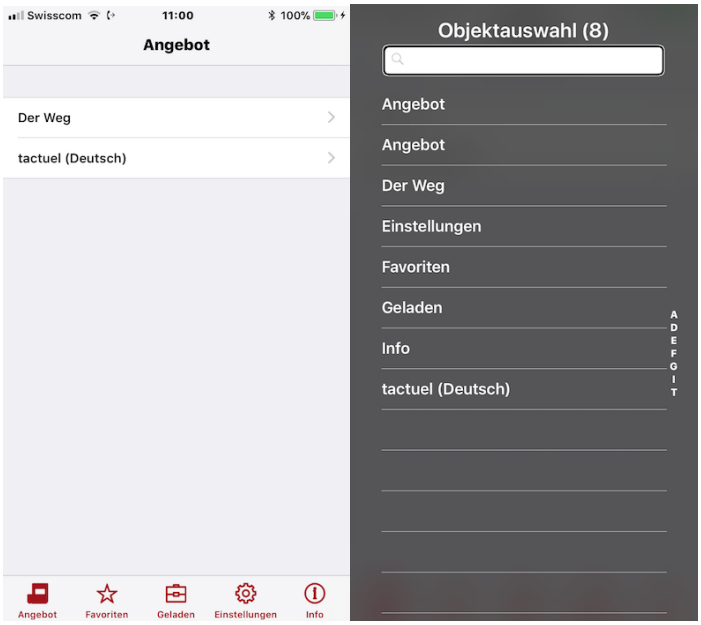

Abbildung 5: Objektauswahl der eKiosk-App für die Angebots-Seite

Einschalten des Bildschirmvorhangs

Um sich in eine sehbehinderte Person hineinversetzen zu können, kann der Bildschirmvorhang aktiviert werden. Im aktiven Zustand schaltet VoiceOver das Display aus, so dass niemand mitlesen kann. Das Ein- und-Ausschalten des Vorhangs wird mit der Geste „Dreifachtippen mit drei Fingern“ gesteuert.[7]

[7] Vgl. Apple, Accessibility-Verification, 2013

4. Programmieren einer barrierefreien App

In diesem Kapitel geht es darum wie iOS-Apps für Menschen mit einer Sehbehinderung zugänglich gemacht werden können.

4.1 Was wir als App-Entwickler tun können

Im Folgenden eine Auflistung von Aufgaben, welche wir uns als App-Entwickler ausführen können, damit iOS-Apps auch für Menschen mit einer Sehbehinderung zugänglich sind:

1. **Barrierefreies Design**
   Wenn man an die Zielgruppe der eigenen App denkt, sollte man die sehbehinderten Nutzer nicht vergessen. Hier kann während der Entwicklungsphase die Sprachausgabe (VoiceOver) eingeschaltet werden, um sich einen Einblick darüber zu verschaffen wie die App von einem Menschen mit einer Sehbehinderung wahrgenommen wird.

2. **Accessibility Features**
   Die beste Methode der Bestimmung der Barrierefreiheit einer App ist die Validierung folgender Punkte:

   1. Ich kann mit der Geste „Finger nach rechts/links streichen“ die einzelnen Controls der App ansteuern

   2. Alle Controls der Benutzeroberfläche, welche angesteuert werden können, haben ein Label

   3. Die Reihenfolge, in der VoiceOver die Controls vorliest, ist sinnvoll

   4. Controls mit sich änderndem Inhalt werden aktualisiert und von VoiceOver vorgelesen

   5. Die Hints der einzelnen Controls sind hilfreich

Treffen die oben genannten fünf Punkte zu, ist die Wahrscheinlichkeit gross, dass die App auch barrierefrei ist. Im Kapitel 4.2 wird näher be-

schrieben wie diese Attribute in der Entwicklungsumgebung XCode definiert werden können.

3. **Audit von bestehenden Apps mit dem Accessibility Inspector**
   Falls die App bereits entwickelt wurde, kann die Barrierefreiheit auch noch nachträglich überprüft werden. Die Kontrolle erfolgt mit einem
   Werkzeug namens Accessibility Inspector. Mehr dazu in Kapitel 5.[8]

Sind während der Entwicklung die Standard-Controls von Apple eingesetzt worden, stehen die Chancen gut, dass die App auch barrierefrei ist. Wir als Software-Entwickler sollten in erster Linie identifizieren, welche Behinderung und Schwächen bei den Nutzern auftreten können, und versuchen Lösungen anzubieten. Denn von einer angemessenen Schriftgrösse und klaren Kontrasten profitieren nicht nur Nutzer mit einer Behinderung, sondern auch alle anderen Anwender unserer Apps.[9]

4.2 Accessibility Attribute

Sehbehinderte sind auf Beschreibungen der eingesetzten GUI-Controls angewiesen. Wenn sie auf ein Control zugreifen, liest VoiceOver die dazugehörigen Attribut-Informationen vor.[10]

Die UI-Accessibility Programmieroberfläche definiert dabei folgende Attribute:

1. **Label.**
   Das Label wird benötigt, weil es den Namen des Accessibility-Controls beinhaltet.

2. **Traits.**
   Ein Trait beschreibt den Zustand, das Verhalten oder die Nutzung eines Controls.

[8] Vgl. Apple, WWDC-Accessibility, 2015
[9] Vgl. Semler J., [App Design, 2016]: App-Design, S. 153ff
[10] Vgl. Apple, Accessibility-Understanding, 2012

3. **Hint**
   Ein Hint beschreibt das Resultat einer Aktion eines Controls.

4. **Identifier**
   Ein Text, welche das Control eindeutig identifiziert. Habe ich bisher noch nie eingesetzt.

5. **Frame**
   Die Bildschirm-Koordinate des Controls, welche durch die Klasse CGRECT gegeben ist. Das Frame-Attribut wird benötigt, weil ein Accessibility-Control seine Position der Benutzeroberfläche mitteilen muss. Nur so weiss VoiceOver dann auch an welcher Stelle im Screen es ein Control markieren muss.

6. **Value**
   Der aktuelle Wert eines Controls. Wird benötigt, wenn sich der Wert des Controls ändert. Ein Beispiel wäre ein Textfeld-Control mit dem dazugehörigen Label „E-Mail“. Hier wird das Accessibility-Attribut Value benötigt, weil sich der Wert des Controls ändern kann und sich vom Label unterscheiden muss. Ein weiteres Beispiel ist ein Slider mit dem Label „Speed“ und dem aktuellen Wert von „50%“.

In den folgenden Abschnitten folgt eine detailliertere Beschreibung der Accessibility-Attribute und in welchen Fällen sie eingesetzt werden.

4.2.1 Label

Das Label-Attribut ist die Beschreibung eines GUI-Controls. VoiceOver wird nach fokussieren des Controls als erstes diesen Text sprechen. Den passenden Text zu einem Label findet man am besten, wenn man sich vorstellt wie die graphische Oberfläche von einer sehenden Person wahrgenommen wird. Ist die graphische Oberfläche einer App gut umgesetzt, kann sich eine sehende Person genau vorstellen welche Aufgabe jedes einzelne Control hat. Der Titel des Controls oder das Design eines Icons tragen also wesentlich zu diesem Verständnis bei. Es sind genau diese Information, welche man einem sehbehinderten Nutzer zwingend bereitstellen sollte. Nachfolgend eine Auflistung der Regeln, welche für die Definition eines Labels gelten.

Ein Label sollte,

1. **Das Control beschreiben**
   Das Label besteht idealerweise aus einem einzigen Wort, wie zum Beispiel „Hinzufügen“, „Speichern“, „Löschen“ oder „Favoriten“. Manchmal ist es hilfreich einen Ausdruck zu verwenden. Beispiele sind: „Musik abspielen“, „Name hinzufügen“ oder „Person löschen“.

2. **Nicht den Typ des Controls im Namen enthalten**
   Um welche Art von Control es sich handelt, ist im Trait-Attribut definiert und sollte daher im Label-Attribut nicht wiederholt werden. Ein Button mit dem Label „Hinzufügen Taste“, würde VoiceOver mit „Hinzufügen Taste Taste“ vorlesen.

3. **Mit einem Grossbuchstaben beginnen**
   Der Beginn mit einem Grossbuchstaben bewirkt, dass VoiceOver die Beschreibung mit dem richtigen Tonfall vorleist.

4. **Endet nicht mit einem Fragezeichen**
   Die Beschreibung eines Labels ist keine Frage und sollte daher nicht mit einem Fragezeichen enden.

5. **Liegt in der gewünschten Sprache vor**
   Bei der Entwicklung barrierefreier Apps soll darauf geachtet werden, dass die Texte in der gewünschten Sprache vorliegen. VoiceOver wird immer die vordefinierte Sprache des Nutzers verwenden.

Abbildung 6: Der XCode-Accessibility-Bereich im Storyboard

4.2.2 Hint

Das Hint-Attribut beschreibt das Resultat einer Aktion. Ein Hint sollte nur verwendet werden, falls das Resultat einer Aktion nicht bereits aus dem Titel des Labels hervorgeht. Falls sich zum Beispiel ein „Play“-Button auf dem Screen befindet, kann mit dem Hint-Attribut beschrieben werden was nach einem Klick genau passiert. Man könnte so dem Nutzer zum Beispiel mitteilen, dass nach dem Klicken, der ausgewählte Titel aus der Songliste abgespielt wird.

Ein Hint sollte,

1. **Das Resultat beschreiben**
   Ein Hint sollte, das Resultat der Aktion beschreiben. Viele Controls benötigen zwar das Hint-Attribut, aber man sollte sich genau überlegen ob man es auch verwenden sollte. Denn der Nutzer muss diesen immer erst abhören, bevor er das Control verwenden kann.

2. **Eine klare und eindeutige Sprache verwenden**

3. **Mit einem Verb anfangen, gefolgt von einem Subjekt**
   Man sollte die dritte Person im Singular verwenden. Der Hint soll ja schliesslich nicht als Befehl vorgelesen werden. Dem Nutzer soll mit „Spiele Song ab“ nicht befohlen werden den Song abzuspielen, sondern lediglich mitgeteilt werden, dass das Abspielen des Songs ausgelöst wird: „Spielt den Song ab“.
   Wie findet man die richtige Hint-Beschreibung? Man kann sich vorstellen das man jemandem versucht die Funktion des Controls zu erklären. Die Rede ist dann von „Die Auswahl dieses Controls spielt den Song ab“ oder „Ein Wischen nach rechts löscht den Song“. Der zweite Ausdruck kann in den meisten Fällen als Hint verwendet werden. In unserem Beispiel wäre das: „Spielt den Song ab“.

4. **Fängt mit einem Grossbuchstaben an und hört mit einem Punkt auf**
   Ein Hint ist zwar kein Satz, aber ein Punkt am Ende bewirkt, dass VoiceOver den Inhalt mit dem richtigen Tonfall spricht.

5. **Beinhaltet nicht den Namen der Aktion oder Geste**
   Ein Hint beschreibt nicht wie der Nutzer die Aktion auslösen muss, sondern beschreibt was nach der Aktion genau passiert. Also bitte keine Hints erstellen wie „Tippen, um den Song abzuspielen“ oder „Nach rechts wischen, um den Song zu löschen“. Hier wäre „Spielt den Song ab“ oder „Löscht den Song“ eine weit bessere Wahl.

6. **Beinhaltet nicht den Namen des Controls**
   Der Nutzer erhält Informationen zum Control bereits vom Label-Attribut, also sollte man den Namen des Controls im Hint-Attribut nicht noch einmal wiederholen. Also keine Hints wie „Speichern speichert Ihre Änderungen“ oder „Zurück geht zurück zum vorherigen Screen“ verwenden.

7. **Beinhaltet nicht den Typ des Controls**
   Bevor das Hint-Attribut vorgelesen wird, weiss der Nutzer bereits um was für ein Control es sich handelt. Diese Information ist im Trait-Attribut des Controls enthalten. Er weiss also ob es sich um ein Suchfeld oder Button handelt. Also bitte keine Hints wie „Button, welche eine Person hinzufügt“ oder „Slider, welche die Skalierung einstellt“ verwenden.

8. **Ist lokalisiert**
   Genau wie das Label-Attribut sollten Hints in der vom Nutzer bevorzugten Sprache vorliegen.

4.2.3 Trait

Das Trait-Attribut beschreibt das Verhalten eines barrierefreien Controls und kann durch mehrere individuell angepasster Traits miteinander kombiniert werden. Ein Standard-Control wie ein Button oder Textfeld bieten bereits Default-Traits an. Nach dem Fokussieren eines Buttons wird z.B. „Taste“ gesprochen. So weiss der Nutzer das noch weitere Aktionen verfügbar sind. Falls man aber das Verhalten eines Standard-Controls anpassen will, muss man das in Kombination mit dem bestehenden Trait tun.
Die UI-Accessibility Programmierschnittstelle definiert aktuell (XCode Version 9.0.1) 16 individuelle Traits:

1. Button
2. Link

3. Image
4. Selected
5. Static Text
6. Search Field
7. Plays Sound
8. Keyboard Key
9. Summary Control
10. User Interaction Enabled
11. Updates Frequently
12. Starts Media Session
13. Adjustable
14. Allows Direct Interaction
15. Causes Page Turn
16. Header

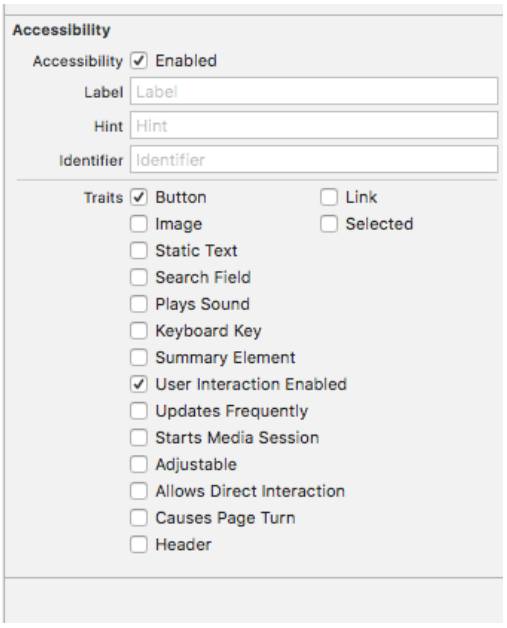

Abbildung 7: Übersicht aller verfügbaren Traits unter XCode

Im Allgemeinen können Traits miteinander kombiniert werden. So zum Beispiel ein „Button“-Trait mit dem „Plays Sound“-Trait. Dieses Control würde sich dann wie ein Button verhalten, aber beim Antippen „Sound“ abspielen.

Falls man Standard-Controls verwendet, braucht es keine zusätzlichen Traits. Also lieber diejenigen Traits verwenden, welche schon für das jeweilige Control definiert sind. Damit diese auch ihre Exklusivität beibehalten.

4.2.4 Reihenfolge wie VoiceOver die Attribute spricht

Wir haben bereits besprochen wie Labels, Hints und Traits definiert werden. Doch wie bestimmt VoiceOver eigentlich, in welcher Reihenfolge die Attribute vorgelesen werden müssen?
Anbei eine Auflistung der Ausgabe, welche VoiceOver beim Fokussieren eines UI-Textfeld spricht:

1. Falls Accessibility-Label definiert: Sprachausgabe des definierten Textes
2. Falls Wert des Controls sich ändert: Sprachausgabe des Wertes im Textfeld
3. Vorlesen um was für eine Art von Element es sich handelt:
   Bei einem Textfeld: „Textfeld“
4. Falls Accessibility-Hint definiert: Sprachausgabe des definierten Textes
5. Falls weitere Aktionen verfügbar: Sprachausgabe der Aktion.
   Bei einem Textfeld: „Zum Bearbeiten Doppeltippen“

4.2.5 Label, Hints und Traits im Code selber definieren

Wer Hints, Labels und Traits nicht über den Interface Builder definieren möchte, kann dies auch direkt im Code tun. Auf folgender Seite gibt es eine ausführliche Anleitung dazu:

https://developer.apple.com/library/content/documentation/UserExperience/Conceptual/iPhoneAccessibility/Making_Application_Accessible/Making_Application_Accessible.html

4.3 Accessibility unter XCode

Nachdem wir besprochen haben welche Accessibility-Attribute VoiceOver benötigt, geht es in diesem Kapitel darum wie wir im XCode diese definieren können. Die Barrierefreiheit einer App kann entweder im Storyboard oder im Programmcode definiert werden. In diesem Kapitel werden beide Möglichkeiten besprochen.

4.3.1 Accessibility im Storyboard

Standardmässig ist ein Apple-Control unter XCode im Accessibility-Inspector als „Accessibility-Control“ gekennzeichnet:

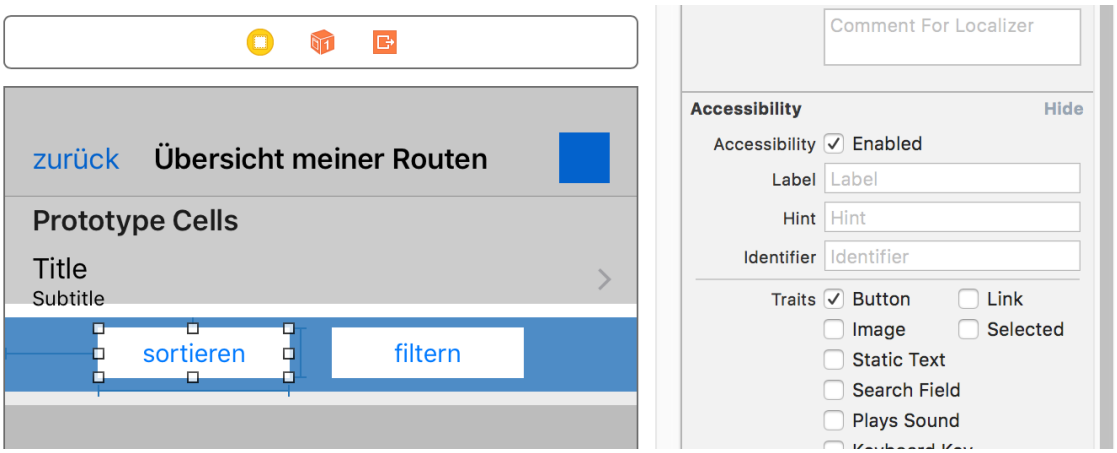

Abbildung 8: Accessibility-Kontrollkästchen aktivieren, damit ein GUI-Control von VoiceOver berücksichtigt wird

Wird dieses Kontrollkästchen so belassen (Kontrollkästchen „Enabled“ markiert), kann VoiceOver das Control auswählen und den unter „Label“ aufgeführten Text vorlesen. Wird unter dem Label-Attribut kein Text erfasst, wird VoiceOver den Wert des Controls vorlesen.
Ist hingegen das Accessibility-Kontrollkästchen nicht aktiviert, wird VoiceOver nicht zu diesem Control navigieren können. Dieses Control existiert dann für eine sehbehinderte Person nicht.

4.3.1.1 Reihenfolge der Sprachausgabe mit dem Interface-Builder festlegen

Die Reihenfolge in der wir die Controls im XCode-Storyboard anordnen, definieren in welcher Reihenfolge VoiceOver die Controls auswählt. Die Reihenfolge ist abhängig von den Screen-Koordinaten (Links nach rechts/Oben nach Unten). In unserem Beispiel-Projekt kann der Benutzer eine Zieladresse eingeben und anschliessend wird ihm, ausgehend von der Startadresse, die Route berechnet. Die Start- und Ziel-Adresse kann nach einem Klick auf den Enter-Button validiert werden. Für die drei vorhandenen Buttons sind Hints definiert. Ansonsten wurden die Defaultwerte so belassen, sprich keine zusätzlichen Labels oder Traits definiert. Anbei zwei Screenshots unseres Beispielprojektes:

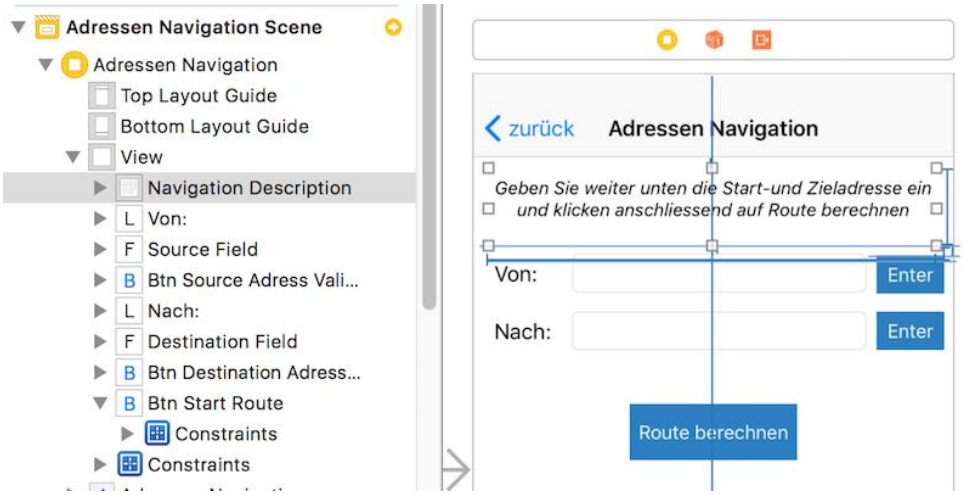

Abbildung 9: Reihenfolge der Controls im Storyboard beeinflusst die Sprachausgabe von VoiceOver

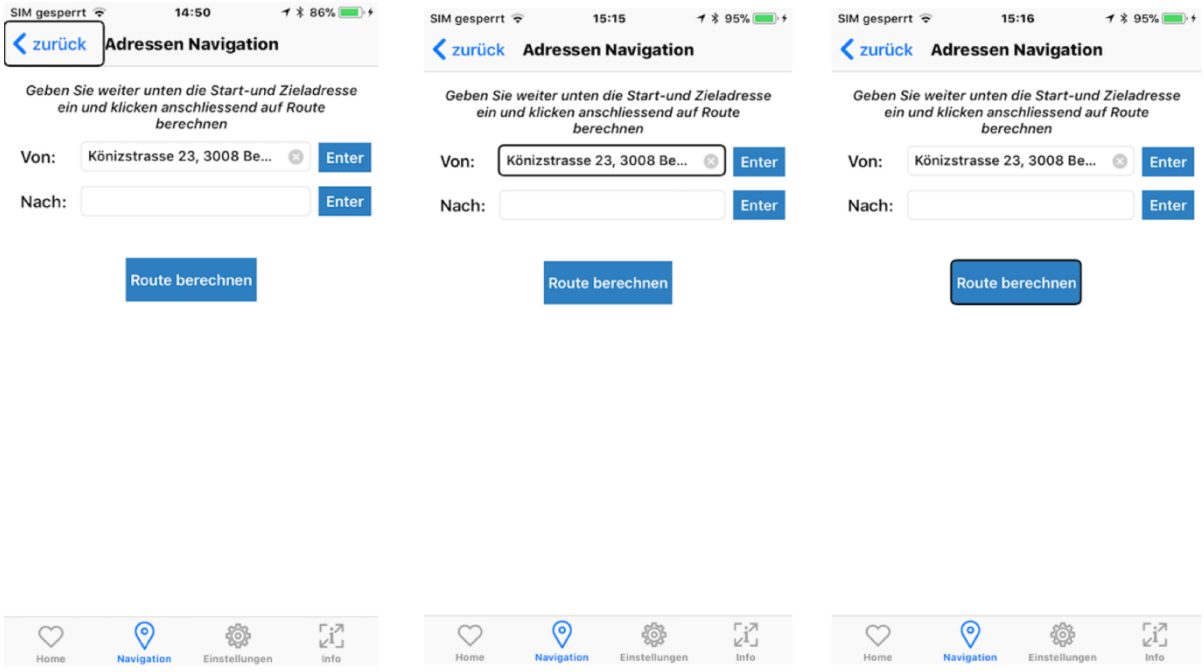

Abbildung 10: Screenshots einer App mit aktiviertem VoiceOver und Cursor-Positionen

VoiceOver wird hier als erstes den „zurück“-Button markieren. Anschliessend wird nach „Adressen Navigation“ die TextView „Geben Sie weiter...“ vorgelesen. VoiceOver spricht dabei folgende Texte, wenn wir mit „Wischen nach rechts“ vom ersten zum letzten Control navigieren:

1. „Zurück-Taste“
2. „Adressen-Navigation: Überschrift“

3. „Geben Sie weiter unten die Start- und Zieladresse ein und klicken anschliessend auf Route berechnen“
4. „Von“
5. „Vorlesen der aktuellen Adresse“ (Hier „Könizstrasse 23“) / „Textfeld“ / „Zum Bearbeiten Doppeltippen“
6. „Clear-Text“ (Kreuz-Icon mit der Möglichkeit den Textinhalt zu löschen) / „Taste“
7. „Enter“ / „Taste“ / „Validiert die Startadresse.“
8. „Nach“
9. „Textfeld“ / „zum Bearbeiten Doppeltippen“
10. „Enter“ / „Taste“ / „Validiert die Zieladresse.“
11. „Route berechnen“ / „Taste“ / „Startet die Routenberechnung.“
12. „Home“ / „Tabulator“ / „1 von 4“
13. „Auswahl“ / „Navigation“ / „Tabulator“ / „2 von 4“
14. etc.

4.3.2 Accessibility im Programmcode

Das Einstellen der Barrierefreiheit im Code hat einige Vorteile. So lässt sich die Sprachausgabe, nach bestimmten Ereignissen, manuell auslösen. Oder man kann die Reihenfolge der Sprachausgabe selber bestimmen.

4.3.2.1 Reihenfolge der Sprachausgabe im Code selber bestimmen

Möchte man die Reihenfolge der Sprachausgabe selber definieren, geht man folgendermassen vor:

1. Als erstes wird die View, auf welchen die einzelnen GUI-Controls angeordnet sind, als nicht barrierefrei gekennzeichnet. Dies ist erforderlich damit wir im zweiten Schritt die Reihenfolge selber bestimmen können:

```
mainView.isAccessibilityControl = false;
```

2. Im zweiten und letzten Schritt werden die Controls, in der von uns gewünschten Reihenfolge, dem Accessibility-Container zugewiesen:

```
mainView.accessibilityControls  = [btnTextModeAuswahl, languagesTableView];
```

VoiceOver wird hier als erstes den Button „btnTextModeAuswahl“ und anschliessend die Table-View „languageTableView“ markieren.

4.3.2.2 Sechs Eigenschaften zur Bestimmung der Barrierefreiheit eines GUI-Controls

Sechs Eigenschaften reichen aus, damit VoiceOver weiss ob ein GUI-Control barrierefrei ist oder nicht. Dazu wird nur wenig Programmcode benötigt. Die gute Nachricht ist, dass beim Einsatz von Apple’s Standard-Controls (Button, Label, Textfeld etc.) die meiste Arbeit bereits von Apple übernommen wird.
Anbei die sechs Fragen, die es im Code zu beantworten gilt: [11]

1. **Soll das Control von VoiceOver markiert werden dürfen?**
   Wir wollen das die sehbehinderte Person mit dem Control interagiert, daher beantworten wir diese Frage mit Ja. Im Programmcode müssen wir so für jedes Control einstellen, ob es von der Sprachausgabe angesprochen werden darf oder nicht. (Hier für den Button „btnSortRoute“):

   ```
   self.btnSortRoute.isAccessibilityControl = true;
   ```

2. **Welchen Titel soll VoiceOver vorlesen?**
   Die Property „accessibillityLabel“ definiert das Label des Accessibility-Controls. Erinnerung: Dieser Text wird als erstes von VoiceOver vorgelesen.

   ```
   self.btnSortRoute.accessibilityLabel = "sortieren";
   ```

3. **Wie kann der Nutzer/VoiceOver mit dem Control interagieren?**
   In diesem Schritt müssen wir definieren, um was für eine Art Control es sich handelt. Damit VoiceOver genau weiss, ob er es mit einem Button oder Textfeld zu tun hat. Bei einem Button würden uns noch weitere Aktionen zur Verfügung stehen. Bei unserem Control

[11] Vgl. Apple, WWDC-Accessibility, 2015

„btnSortRoute“ handelt es sich um einen Button, also definieren wir das folgendermassen:

```
self.btnSortRoute.accessibilityTraits = UIAccessibilityTraitButton
```

4. **Ändert sich der Wert unseres Controls?**
   Falls sich der Wert eines Controls ändert, muss nebst dem Text auch der Accessibility-Wert geändert werden. Nur so kennt Voice-Over den aktuellen Wert des Controls. Bei einem Adressfeld mit Label „Startadresse“ wäre der Accessibility-Wert des Controls die aktuelle Adresse. Diesen müssten wir dann bei einer Aktualisierung ebenfalls ändern:

   ```
   @IBOutlet weak var txtAdressSource: UITextField!
   self.txtAdressSource.accessibilityValue = "Neuer Wert";
   ```

5. **Wie sollen Nutzer mit dem Control interagieren?**
   Mit sogenannten Hints, kann man dem Nutzer Hilfestellung geben, damit er weiss wie er mit dem Control interagieren kann. In unserem Beispiel teilen wir dem Nutzer mit, dass sein Klicken die Liste alphabetisch sortiert:

   ```
   self.btnSortRoute.accessibilityHint = "Sortiert die Liste in alphabetischer Reihenfolge";
   ```

6. **Wo genau auf dem Screen befindet sich das Control?**
   Im letzten Schritt müssen wir VoiceOver noch mitteilen wo sich unser Button im Screen genau befindet:

   ```
   self.btnSortRoute.accessibilityFrame = CGRect(x: 10, y: 10, width: 200, height: 300);
   ```

4.3.2.3 Auslösen der Sprachausgabe

Die Sprachausgabe kann nach bestimmten Ereignissen auch im Code ausgelöst werden. Dies zum Beispiel, wenn sich die Werte von Controls ändern. Dies erfolgt mit sogenannten „PostNotifications“. In unserem Beispiel teilen wir dem Nutzer mit das ein neues Beacon in seiner Umgebung gefunden wurde:

```
func locationManager(_ manager: CLLocationManager, didRangeBeacons beacons:
[CLBeacon], in region: CLBeaconRegion) {
.....
```

```
UIAccessibilityPostNotification(UIAccessibilityAnnouncementNotification, "Neuen Posten gefunden: \(beaconDescr)");

.....

}
```

Weiter besteht die Möglichkeit die Sprachausgabe zu pausieren (UIAccessibilityPauseAssistiveTechnologyNotification) und wiederaufzunehmen (UIAccessibilityResumeAssistiveTechnologyNotification).

4.3.2.4 Dynamische Anpassung der Schriftgrösse

Viele sehbehinderte Nutzer haben eine dynamische Textgrösse auf ihren iPhones eingestellt:

Abbildung 11: Dynamische Textgrösse einstellen in den iPhone-Einstellungen

Damit eine iOS-App diese Schriftgrösse berücksichtigen kann, braucht es folgende Anpassungen im XCode-Projekt:

1. Im „Attributes-Inspector“ ein von Apple vordefinierten Text-Style auswählen
    a. Für Überschriften: Headline
    b. Für Text: Body
2. Auswahl „Automatic Adjusts Font“ im Attributes-Inspector

Diese zwei Schritte reichen bereits aus, damit eine App die eingestellte Schriftgrösse übernimmt. Will man allerdings das die Schriftgrösse

auch angepasst wird, während die App im Hintergrund läuft, so braucht es noch folgende Zeilen Code:

3. In der Methode View-DidLoad definieren, um welche Schrift-Information es sich bei den einzelnen Controls handelt. Jedes Mal wenn die App in den Vordergrund tritt und die View neu geladen wird, werden dann die Schriftinformation neu gelesen:

```
lblRecuniaTitle.font = UIFont.preferredFont(forTextStyle: UIFontTextStyle.headline);

lblVersionValue.font = UIFont.preferredFont(forTextStyle: UIFontTextStyle.body);
```

4.3.3 Wo soll die Barrierefreiheit einer App definiert werden?

Wir haben beschrieben wie die Barrierefreiheit einer App sowhol im Storyboard als auch im Programmcode definiert werden kann. Doch was sollte bevorzugt werden?
Accessibility-Informationen im Code werden gegenüber Accessibility-Einstellungen im Storyboard bevorzugt. Wird so zum Beispiel im Storyboard für ein Control das Kästchen „Accessibility“ nicht aktiviert, aber im Code als „isAccessible“ definiert, so ist es dennoch barrierefrei. Sinn macht das Einstellen der Accessibility im Storyboard also nur wenn die App in einer einzigen Sprache veröffentlicht wird und Controls verwendet werden, deren Werte sich nicht ändern und VoiceOver benachrichtigt werden muss.

5. Der Accessibility-Inspector

Der Accessibility-Inspector zeigt für jedes Control einer Benutzeroberfläche an, ob es barrierefrei ist und welcher Text von VoiceOver gesprochen wird. Mit dem Accessibility-Inspector lässt sich so ganz einfach die Barrierefreiheit einer App überprüfen. Anbei ein Screenshot der Validierung, unserer Beispiel-App, im Accessibility-Inspector:

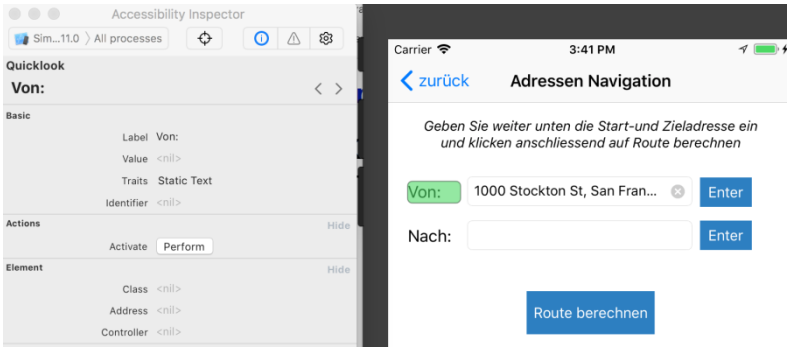

Abbildung 12: Der Accessibility-Inspector im Einsatz

Weitere Möglichkeiten mit dem Accessibility-Inspector sind:

1. Kontrolle, welche Elemente barrierefrei sind
2. Kontrolle der VoiceOver-Reihenfolge
3. Validierung (Audit) der Barrierefreiheit mit «Run Audit». Bei allfälligen Warnungen werden Verbesserungsvorschläge angezeigt
4. Möglichkeit Accessibility-Optionen zu aktivieren, um so zu testen wie die App u.a. auf dynamische Schriftgrössen reagiert

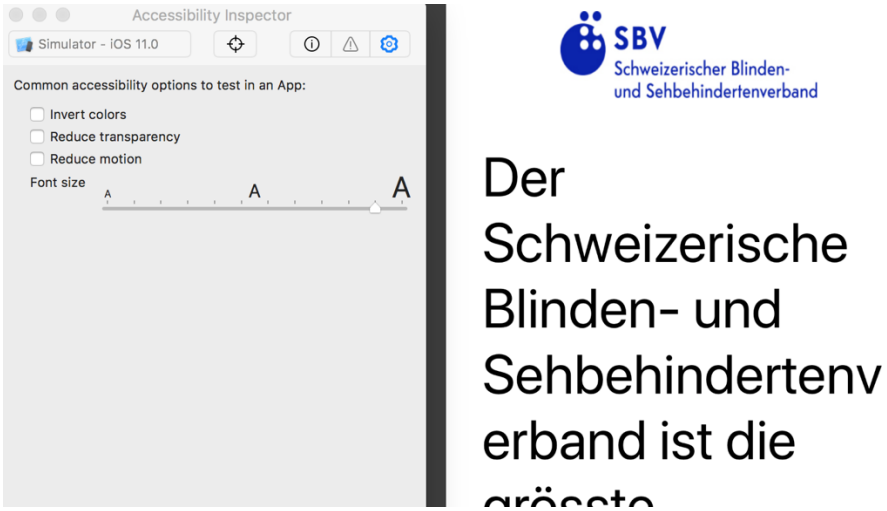

Abbildung 13: Auswahl von Accessibility-Optionen

5.1 Erste Schritte

Der Accessibility-Inspector ist als Developer Tool bereits in XCode integriert und kann folgendermassen gestartet werden:

1. Starten der Entwicklungsumgebung XCode
2. Auswahl eines iPhone-Modells (als Simulationsumgebung) unter XCode
3. Aufruf der zu testenden App im Simulator
4. Aufruf des Accessibility Inspectors: „XCode/Open Developer Tool/Accessibility Inspector“
5. Auswahl von „Simulator – iOS 11.0“ als Target, damit der Inspector auch nur die Accessibility des XCode-Simulators validiert und nicht die eines anderen Prozesses.
   Hier stehen noch weitere Targets zur Verfügung. Da wir unsere App auf einem iPhone-Simulator ausführen, wählen wir hier ein iPhone-Modell (Hier iPhone 6s) aus

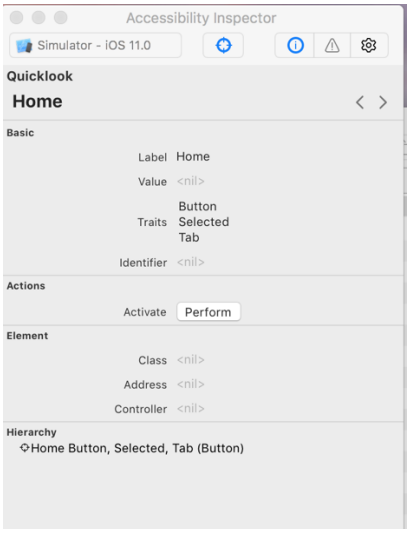

Abbildung 14: Der Accessibility-Inspector mit Auswahl des Simulators als Target

5.2 Benutzeroberfläche

Der Accessibility-Inspector besteht aus den Bereichen Target-Chooser, Inspection-Pointer, Audit und Settings.

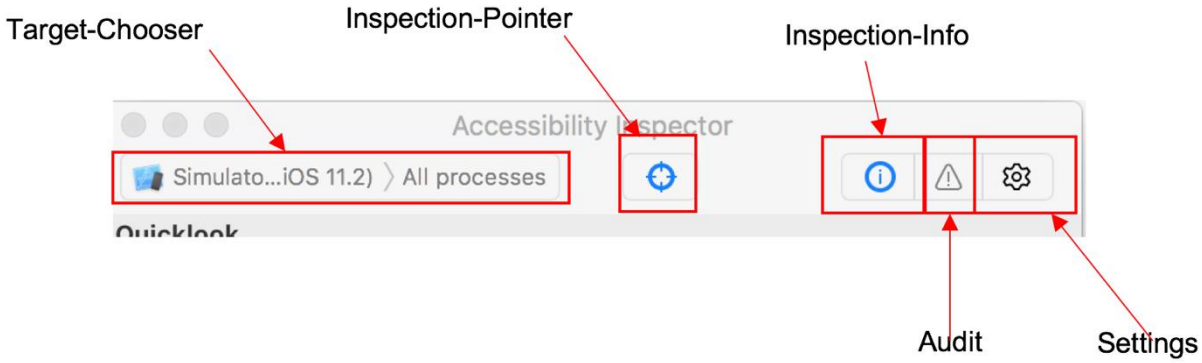

Abbildung 15: Bereiche des Accessibility-Inspectors

Target-Chooser
Auswahl des Targets (Ziel-Gerät) auf welchem der Accessibility-Inspector seine Validierungen vornehmen soll. Wird eine iOS-App validiert, wählt man ein iPhone-Modell im XCode-Simulator aus.

Inspection-Pointer
Ist der Inspection-Pointer aktiviert, kann mit dem Mauszeiger auf ein beliebiges Control navigiert werden.

Ein Einfachklick mit der Maus ist gleichbedeutend wie ein Einfachtipp auf dem iPhone. Scrolling mit der Maus simuliert die Wischgeste mit dem Finger. Zu einem fokussierten Control werden jeweils Accessibility-Informationen wie Label, Value oder Trait angezeigt.
Wurde ein GUI-Control in der App als barrierefrei deklariert, kann es mit dem Pointer auch markiert werden. Ist dies allerdings nicht möglich, so ist es kein Accessibility-Control und eine sehbehinderte Person hat keinen Zugriff darauf.

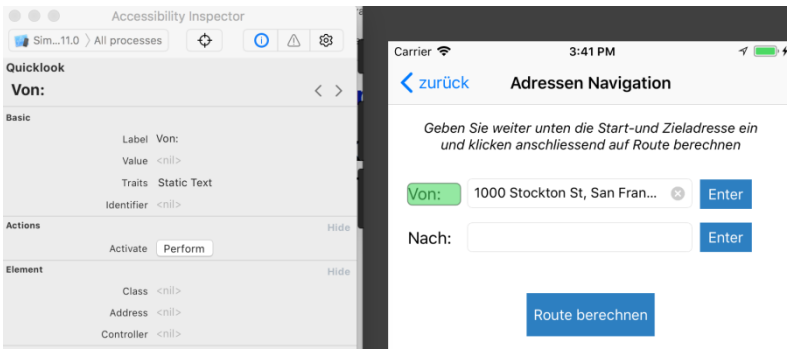

Abbildung 16: Inspection Pointer im Einsatz

Audit
Mit einem Klick auf den «Run Audit»-Button wird die aktuelle Maske auf Barrierefreiheit hin überprüft. Es werden Accessibility-Warnungen und Verbesserungsvorschläge angezeigt. Mit dem Audit kann so herausgefunden werden, wie barrierefrei die eigene App ist.

Settings
Im Settings-Bereich kann das Verhalten der App auf Accessibility-Einstellungen (Farbe invertieren, Grösserer Text etc.) überprüft werden.[12]

5.3 VoiceOver-Reihenfolge validieren

Im Accessibility Inspector wird im Bereich Quicklook das aktuell fokussierte Control angezeigt. Mit einem Klick auf «Pfeil rechts» / «Pfeil links» kann zum nächsten oder vorherigen Control navigiert werden. Dies ist eine gute Möglichkeit die VoiceOver-Reihenfolge zu validieren.

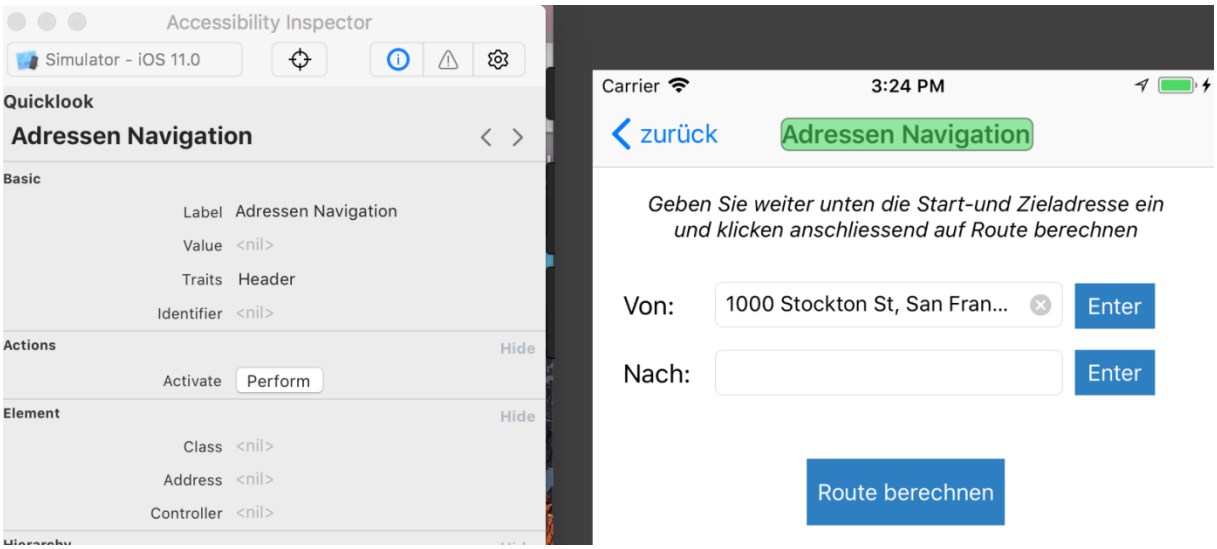

Abbildung 17: Reihenfolge der Accessibility-Elemente überprüfen

5.4 App validieren

Im Register Audit wird nach einem Klick auf „Run audit“ die im Simulator ausgeführte App auf ihre Barrierefreiheit hin überprüft.

[12] Vgl. Apple, WWDC-Accessibility, 2016

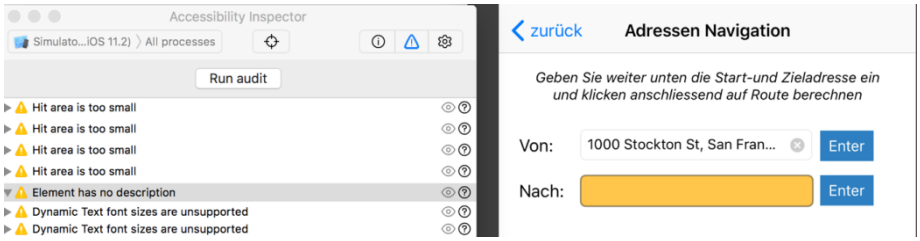

Abbildung 18: Audit einer App

Warnings werden in einer Liste angezeigt. Diese können aufgeklappt werden, um weitere Informationen zu erhalten. Bei einem Klick auf die einzelnen Warnungen wird das entsprechende Control im Simulator ausgewählt. Mit einem Klick auf das Fragezeichen-Symbol wird eine Empfehlung (Fix Suggestion) zur Behebung der Warnung angezeigt:

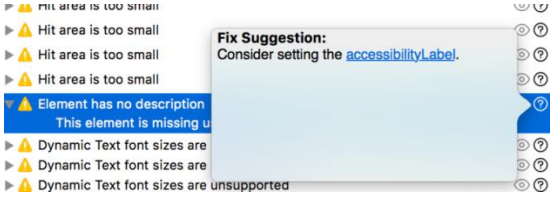

Abbildung 19: Anzeige Fix Suggestion zu einer Warnung

Fehlen Accessibility-Inhalte wird dies mit „This element is missing useful accessibility information“ angezeigt. Auch wird angezeigt falls der Kontrast zu gering sein sollte. Anbei ein Auszug von weiteren Warnungen:

1. **Element has no description**
   Diese Warnung wird angezeigt falls das Accessibility-Attribut „Label“ nicht gesetzt wurde.

2. **Contrast failed**
   Diese Warnung wird angezeigt, wenn die Validierung des Kontrastes fehlgeschlagen ist. Als Lösungshinweis wird angezeigt, dass ein bestimmtes Verhältnis zwischen der Hintergrundfarbe und der Textfarbe gegeben sein sollte. [13]

[13] Vgl. Apple, Accessibility-Verification, 2013

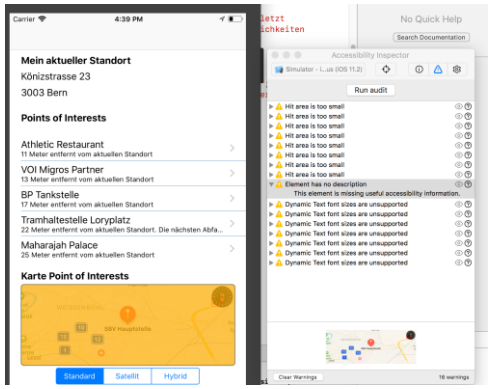

Abbildung 20: Anzeige von Warnungen nach Durchführung eines Audits

5.5 Accessibility-Attribute im Storyboard und Programmcode

Die im Accessibility-Inspector unter Basic aufgeführten Informationen greifen auf Accessibility-Informationen der App zu. Diese können sowohl im Programmcode als auch im Storyboard erfasst werden.

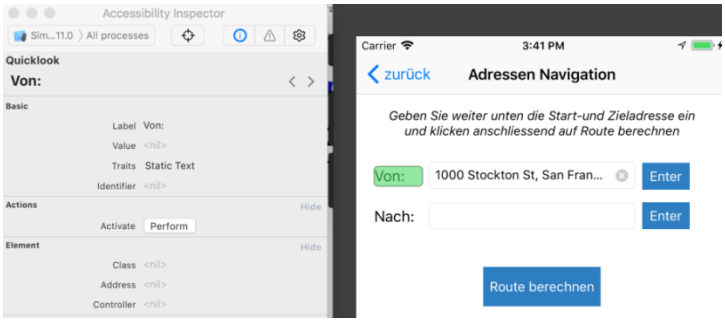

Abbildung 21: Anzeige von Accessibility-Informationen, welche im Storyboard definiert wurden

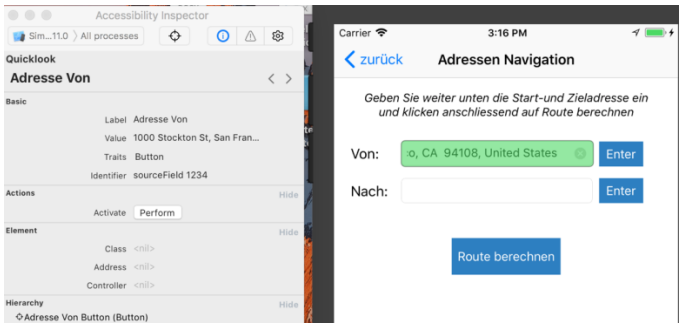

Abbildung 22: Anzeige von Accessibility-Informationen, welche im Programmcode definiert wurden

```
private func setAccessibilityProperties()
{

    sourceField.isAccessibilityElement = true;
    sourceField.accessibilityLabel = "Adresse Von";
    sourceField.accessibilityHint = "Eingabe der Ausgangsadresse";
    sourceField.accessibilityIdentifier = "sourceField 1234";
    sourceField.accessibilityTraits = UIAccessibilityTraitButton;

}
```

Abbildung 23: Definition der Accessibility-Attribute im Programmcode

5.6 Fazit Einsatz Accessibility-Inspector

Der Accessibility-Inspector gibt einen schnellen Überblick über die Barrierefreiheit der eingesetzten GUI-Controls. Während unserer Testphase hat sich gezeigt, dass die Validierung mit dem Accessibility-Inspector nicht immer ganz zuverlässig ist. Beim Ausführen des Audits wurden Elemente mit vorhandenem Accessibility-Label nicht erkannt und dementsprechend Warnungen angezeigt. Weiter liefert der Accessibility-Inspector keine Informationen zu Hint-Attributen.
Hinzu kommt noch, dass auf dem Simulator VoiceOver nicht funktioniert.

Um eine App vollständig auf Barrierefreiheit zu überprüfen, braucht es daher noch unbedingt Tests auf einem physischen iOS-Mobilgerät mit eingeschaltetem VoiceOver. Auch um zu überprüfen ob Wertänderungen eines Controls aktualisiert werden oder aber auch ob Texte in der vom Nutzer bevorzugten Sprache vorgelesen werden.

Mehr Informationen zur Nutzung des Accessibility-Inspectors gibt es auf folgender Webseite:
https://developer.apple.com/videos/play/wwdc2016/407/

6. Weiterführende Links

1. Understanding Accessibility on iOS
   https://developer.apple.com/library/content/documentation/UserExperience/Conceptual/iPhoneAccessibility/Accessibility_on_iPhone/Accessibility_on_iPhone.html

2. Prüfen einer App auf Barrierefreiheit mit dem Accessibility-Inspector
   https://developer.apple.com/library/content/technotes/TestingAccessibilityOfiOSApps/TestAccessibilityiniOSSimulatorwithAccessibilityInspector/TestAccessibilityiniOSSimulatorwithAccessibilityInspector.html#//apple_ref/doc/uid/TP40012619-CH4

3. Testen der Barrierefreiheit einer App mit der Sprachausgabe auf einem Mobilgerät
   https://developer.apple.com/library/content/technotes/TestingAccessibilityOfiOSApps/TestAccessibilityonYourDevicewithVoiceOver/TestAccessibilityonYourDevicewithVoiceOver.html#//apple_ref/doc/uid/TP40012619-CH3-SW1

4. Making your iOS App Accessible
   https://developer.apple.com/library/content/documentation/UserExperience/Conceptual/iPhoneAccessibility/Making_Application_Accessible/Making_Application_Accessible.html#//apple_ref/doc/uid/TP40008785-CH102-SW5

5. WWDC 2015 Video: iOS Accessibility on iOS
   https://developer.apple.com/videos/play/wwdc2015/201/

6. W3C Accessibility Prinzipien für Mobilgeräte
   a. https://www.w3.org/WAI/intro/people-use-web/principles#distinguishable
   b. https://www.w3.org/WAI/mobile

c. http://www.einfach-barrierefrei.net/umsetzen/grundlagen/uebersicht_wcag.html

7. Auditing our Apps for Accessibility
   https://developer.apple.com/videos/play/wwdc2016/407/

8. What's New in Accessibility
   https://developer.apple.com/videos/play/wwdc2016/202/

9. Schweizer Accessibility-Studie 2016
   http://www.access-for-all.ch/images/Accessibilty_Studie/2016/SchweizerAccessibilityStudie2016.pdf

7. Literaturverzeichnis

Apple [Accessibility-Understanding, 2012]: Accessibility Progamming Guide for iOS, https://developer.apple.com/library/content/documentation/UserExperience/Conceptual/iPhoneAccessibility/Accessibility_on_iPhone/Accessibility_on_iPhone.htm, (Zugriff 2018-01-15)

Apple [Accessibility-Verification, 2013]: About Accessibility Verification on iOS, https://developer.apple.com/library/content/technotes/TestingAccessibilityOfi-OSApps/TestingtheAccessibilityofiOSApps/TestingtheAccessibilityofiOSApps.html, (Zugriff 2018-01-15)

Apple [WWDC-Accessibility, 2015]: iOS Accessibility, https://developer.apple.com/videos/play/wwdc2015/201/, (Zugriff 2018-01-15)

Apple [WWDC-Accessibility, 2016]: Auditing Your Apps for Accessibility, https://developer.apple.com/videos/play/wwdc2016/407/, (Zugriff 2018-01-15)

Roden, Golo [Qualitativ-CSharp, 2008]: Kenne Deinen Feind! – Zehn merkmale von qualitativ hochwertigem Code, http://www.dotnetpro.de/core/c/hochwertiger-code-1130747.html, (Zugriff 2018-01-15)

Semler, Jan [App Design, 2016]: App Design – Alles zu Gestaltung, Usability und User Experience, Bonn: Rheinwerk-Verlag GmbH, 2016

Stiftung „Zugang für alle“ [Accessibility-Studie, 2016]: Schweizer Accessibility-Studie 2016 – Bestandesaufnahme der Zugänglichkeit bedeutender Schweizer Internet-Angebote, Zürich: Stiftung „Zugang für alle“, 2016

Vollmer, Guy [App Engineering, 2017]: Mobile App Engineering – Eine systematische Einführung – von den Requirements zum Go Live, Heidelberg: dPunkt-Verlag GmbH, 2017

W3G [Accessibility Principles, 2017]: Accessibility Principles – How People with Disabilities Use the Web, https://www.w3.org/WAI/intro/people-use-web/principles#distinguishable, (Zugriff 2017-01-18)