Barrierefreiheit: Stolpersteine bei mobilen Anwendungen überwinden, Teil 1

Entwickler mobiler Anwendungen müssen besonders auf den barrierefreien Zugang zu ihren Apps achten. Dabei hil es, die entscheidenden Herangehensweisen ebenso zu kennen wie potenzielle Hindernisse.

23.07.2019 09:55 Uhr – Sergey Yurchenko, Theresa Jordan, Martin Kinting

(Bild: Shutterstock)

Beim Entwickeln mobiler Anwendungen ist Barrierefreiheit ein wichtiges Thema, weil ein Alltag ohne Smartphone für viele Nutzer nicht denkbar ist und sie auf einige mobile Apps angewiesen sind. Unzureichende Optimierung kann dazu führen, dass einzelne Funktionen schlecht erreichbar bis unzugänglich bleiben. Dadurch verschlechtert sich die Benutzererfahrung erheblich. Dabei ist die Nutzerakzeptanz eines Produkts ein wichtiger Indikator, der in der Wettbewerbssituation am Softwaremarkt schnell entscheidet, ob Anwender zu zufriedenen Kunden werden oder lieber zu besseren Alternativen der Konkurrenz greifen.

Damit alle Nutzer sich gleich behandelt fühlen können, existieren Konzepte mit dem Ziel eines weitgehend barrierefreien Zugangs zu allen Lebensbereichen. Bei mobilen Anwendungen ist Barrierefreiheit zwar kein neues Thema, aber es herrscht eine gewisse Unsicherheit bei manchen Entwicklern. Im Folgenden finden sich Antworten auf typische Fragen wie "Für wen ist Barrierefreiheit primär gedacht?", "Welche Besonderheiten in Bezug auf Barrierefreiheit hat der mobile Anwendungsfall?" oder "Wo sollen Entwickler anfangen, um ihre Anwendung weitgehend barrierefrei zu gestalten?".

Gleichberechtigte Teilhabe am gesellschaftlichen Leben

Barrierefreiheit: Stolpersteine bei mobilen Anwendungen überwinden

- **Teil 1: Herangehensweise und Hindernisse**
- Teil 2: Praktische Umsetzung für Android

Die moderne Gesellschaft setzt eine aktive Teilnahme in unterschiedlichen Bereichen voraus, sei es im Gesundheits- und Versicherungswesen, beim Banking, bei der Kommunikation und Mobilität im Alltag, in Bildung und Politik. Die Gesellschaft ist ebenso heterogen wie die genannten Bereiche: Sie umfasst Menschen mit unterschiedlichen Kulturen und Hintergründen, Bildungsgraden und Berufsausrichtungen, Lebenssituationen und Präferenzen. Die Vielfalt wird von weiteren Merkmalen körperlicher Natur ergänzt. Dabei sind physische und psychische Eigenschaften von Menschen zu berücksichtigen, die bei einzelnen Personen unterschiedlich ausgeprägt bis eingeschränkt sein können. Im letzten Fall spricht man von einer Einschränkung oder Behinderung.

Die Menge von Einschränkungs- oder Behinderungsarten lässt sich für die Nutzung von Anwendungen auf vier Kategorien aufteilen:

- sehbeeinträchtigte Nutzer
- blinde Nutzer
- motorisch beeinträchtigte Nutzer
- gehörgeschädigte Nutzer

Einschränkungs- oder Behinderungsarten sind in temporär und permanent zu unterteilen. Während eine vorübergehende Verletzung eine temporäre, motorische Einschränkung mit sich bringen kann, verursacht eine altersbedingt auftretende Sehschwäche eine dauerhafte Beeinträchtigung. Bestimmte äußerliche Bedingungen wie schlechte Lichtverhältnisse oder eine die Bewegung einschränkende Arbeitskleidung können weitere Einschränkungen verursachen. Die Beispiele zeigen, dass das sensible Thema alle Menschen im Laufe ihres Lebens mehr oder weniger betrifft oder betreffen wird. In der Praxis bedeuten solche Einschränkungen und Hindernisse eine Barriere für Menschen.

Der mobile Anwendungsfall

Da Nutzer mit Einschränkungen und Behinderungen ihre mobilen Geräte überdurchschnittlich intensiv benutzen, ist die Bereitstellung eines barrierefreien Nutzererlebnis von hoher Relevanz. Der mobile Anwendungsfall unterscheidet sich vom Desktop durch einen anderen Nutzungskontext und variierende äußerliche Bedingungen. Mobile Anwendungen führen Funktionen meist schnell und präzise aus, um konkrete Aufgaben mit minimalem Aufwand seitens des Nutzers zu erledigen. Unterschiedliche Licht- und Geräuschkulissen, Abstand und Betrachtungswinkel zwischen mobilem Gerät und den Augen des Nutzers, Bedienung mit einer oder mit beiden Händen, Ablenkungs- und Störfaktoren wie Straßenverkehr oder fehlender Netzempfang – viele Faktoren, auf die Entwickler keinen Einfluss haben, können die Nutzung mobiler Anwendungen erschweren und Barrieren begünstigen.

Barrieren lassen sich allerdings vermeiden, indem man während der Konzeption und Entwicklung auf potenzielle Ursachen achtet und deren Entstehung verhindert. Zu den häufigen Ursachen für Barrieren, die das Benutzererlebnis negativ beeinträchtigen können und einigen Nutzergruppen den Zugriff zu mobilen Anwendungen erschweren, gehören:

1. Wegen wachsender Bildschirmgrößen der mobilen Geräte sind die dadurch ebenfalls häufig vergrößerten Benutzeroberflächen nicht überall gleich leicht zu erreichen (die sogenannte "The Thumb Zone") – als Folge haben unter anderem ältere Nutzer Schwierigkeiten mit dem Erreichen einiger Bereiche der Bildschirmoberfläche, ohne ständig umgreifen zu müssen.
2. Interaktive Elemente auf der Benutzeroberfläche sind zu nah beieinander platziert oder klickbare Flächen sind zu klein, wodurch sich Nutzer häufiger vertippen oder bestimmte Schaltflächen nur schwer auswählen können.
3. Ungenügende Kontraste wirken auf die Augen ermüdend und können ein Hindernis für sehschwache Nutzergruppen darstellen. Zudem leidet dabei die Lesbarkeit in der App stark.
4. Bei der Textvergrößerung über die Systemeinstellung können Elemente verrutschen und für Orientierungslosigkeit sorgen. Noch ungünstiger ist, wenn die Systemeinstellung keine Auswirkung

5. Beim Navigieren mit Gesten (Schaltersteuerung, Voice Over oder TalkBack) erfolgt die Auswahl in einer ungewünschten beziehungsweise ungünstigen Reihenfolge. Bei modalen Views kann es vorkommen, dass die darauf liegenden Elemente nicht auswählbar sind und die Navigation weiter auf der darunterliegenden Ebene stattfindet. Solche Modal Views sind umständlich bis unerreichbar.
6. Die Sprachausgabe versagt häufig wegen passender Bezeichnungen der Elemente. Wenn nicht alle Felder auf Sprachausgabe eingestellt sind, lassen sich Unterseiten und Funktionen nicht aufrufen. Fehlende oder nicht eindeutig benannte Beschriftungen von interaktiven Elementen erschweren die Navigation für blinde oder sehbeeinträchtigte Menschen, wenn beispielsweise ein Button ein Piktogramm enthält, das mit der Beschriftung "pictures/buttons/123456.jpeg" hinterlegt ist. Auch fehlende Formularbeschriftungen blockieren die Nutzung. Auf dem Desktop per Mouseover erreichbare Inhalte werden vom Screenreader häufig nicht erkannt und bleiben blinden Nutzern verborgen. Wenn sich im Menü durch Berühren eines Menüpunkts selbstständig eine neue Seite öffnet, beginnt der Screenreader die neue Seite, anstatt die Menüpunkte vorzulesen. Pop-ups und Werbung stören die Sprachausgabe oder stoppen sie sogar: Ein Werbebanner inmitten eines Texts wird nicht nur als nervig empfunden, sondern beendet häufig das Vorlesen komplett. Captchas sind unüberwindbar und verhindern sogar mit Audio häufig ein Weiterkommen.
7. Die Umsetzung von Sprachbefehlen ist unter anderem die Kombination aus deutschen Befehlen und englischen Titeln oder bei Eigennamen fehlerhaft.
8. Eine zu hohe Komplexität von Informationen und Funktionen blockiert Verständnis und Bedienbarkeit.
9. Es mangelt an Fehlertoleranz: wie der zusätzlichen Abfrage einer Bestätigung vor wichtigen Aktionen. Oft ist nicht möglich, Aktionen rückgängig zu machen.
10. Fehlen der kontextbezogenen Hilfe wie der Erläuterung von Fehlern können zum Fehlverständnis von Aufforderungen führen.
11. Ein schwieriger Satzbau, zweideutige Begriffe oder Fachsprache sorgen für Missverständnisse und Unsicherheit während der Nutzung.
12. Hilfestellung über alternative Kanäle.

Kriterien und Nutzergruppen

Jede Kategorie hat ihre spezifischen Anforderungen an mobile Anwendungen und Software im Allgemeinen. Konkret sind folgende Kriterien je nach Nutzergruppe zu erfüllen:

	Kriterien	**Kategorien**			
		Sehbeeinträchtigung	**Blindheit**	**Motorische Beeinträchtigung**	**Gehörschaden**
1	The Thumb Zone			+	
2	Abstände und Elementgröße	+		+	
3	Kontrast	+			
4	Schriftvergrößerung	+			
5	Navigation (z.B. TalkBack, Voice Over, Schaltersteuerung): - Wechsel zwischen Inhaltsblöcken, Überschriften, Elementen, Wörtern, Zeichen - Hidden Navigation - Modal Views und vielschichtige Oberflächen korrekt auswählen		+	+	
6	Sprachausgabe (Benennung von		+	+	

		Überschriften, Buttons, Eingabefeldern durch Labels, Hinweistexten, Grafiken, Fehlermeldungen)			
7	Spracheingabe		+	+	
8	Komplexität		+	+	
9	Fehlertoleranz		+	+	+
10	Kontextabhängige Hilfen				+
11	Leichte und einfache Sprache				+
12	Alternative Kontaktmöglichkeiten		+		+

Die Beispiele zeigen, dass Ursachen für Barrieren auf den unterschiedlichen Anwendungsebenen Architektur, Inhalt, Konzept, Design, Implementierung liegen können. Barrierefreiheit ist ein Thema, das Teamarbeit erfordert: Neben Konzeptions- und Designprozessen, bei denen Nutzer und ihre Bedürfnisse im Mittelpunkt stehen sollen, ist die qualitative Implementierung durch das Entwicklungsteam von entscheidender Bedeutung. Letztlich stellt sich die Frage, ob der Aufwand sich lohnt und ob sich die Anpassungen für Unternehmen auszahlen.

Neben dem Erreichen von Unternehmenszielen und Zufriedenheit der Nutzer sind die damit verbundenen Kosten ein weiterer entscheidender Punkt während der Softwareentwicklung. Allerdings ist der erreichbare Nutzen weit höher und der Aufwand meistens geringer als gedacht. Das liegt daran, dass die mobilen Betriebssysteme iOS und Android standardmäßig viele passende Funktionen an Bord haben.

Zwischenfazit

Mobile Geräte sind für ihre Nutzer zu alltäglichen Begleitern geworden und dienen ihnen in jeder erdenklichen Situation als Unterstützung und Hilfe. Dank barrierefreien Anwendungen lassen sich mehr Menschen einbeziehen, damit jeder am Gesellschaftsleben und der Digitalisierung teilnehmen kann. Davon profitieren letztlich Menschen mit und ohne Einschränkungen.

Entwickler sollten besonders bei mobilen Apps darauf achten, dass alle Menschen sie uneingeschränkt benutzen können. Dabei müssen sie unterschiedliche Einschränkungen im Blick haben, die einzelne Benutzergruppen betreffen. Sie helfen damit nicht nur den Betroffenen, sondern auch dem Erfolg der App.

Damit endet der erste Teil des zweiteiligen Artikels. Die Fortsetzung wird konkrete Maßnahmen vorstellen, wie Entwickler ihre Anwendungen anpassen und optimieren können, um sie barrierefrei zu gestalten. (rme)

Sergey Yurchenko
arbeitet bei adesso mobile solutions als UX-Designer. Neben seiner Tätigkeit in der Konzeption und Designentwicklung berät er in Softwareprojekten (Mobile und Web) zu den Themen Usability und Barrierefreiheit.

Theresa Jordan
arbeitet bei adesso mobile solutions als Senior UI-Designerin. Neben ihrer Tätigkeit in der Entwicklung von Gestaltungskonzepten und der Erstellung von User Interface Designs berät sie in Softwareprojekten (Mobile und Web) zu den Themen Usability und Barrierefreiheit. Für ihre herausragende Design- und Kreativleistung erhielt sie zahlreiche nationale und internationale Auszeichnungen, zuletzt den Red Dot Award.

Martin Kinting
ist Niederlassungsleiter der adesso mobile solutions am Standort Berlin und Senior Sales Manager unter anderem in den Bereichen Banking und Life Science.

Barrierefreiheit: Stolpersteine bei mobilen Anwendungen überwinden, Teil 2

Der barrierefreie Zugang zu Apps ist wichtig. Allgemeine Designrichtlinien und konkrete Maßnahmen helfen bei der Umsetzung.

27.08.2019 10:12 Uhr – Roman Zimmer, Alexander Huber, Martin Kinting

Im Frühjahr 2009 erschien die erste nach einer Süßspeise benannte Version des damals noch jungen mobilen Betriebssystems Android. Cupcake (Android 1.5) besaß noch keine Funktion zur barrierefreien Bedienung. Das änderte sich jedoch noch im gleichen Jahr mit der Veröffentlichung von Donut (Android 1.6) sowie Googles mobile Screenreader TalkBack. Damit hielt eine grundlegende Zugänglichkeit Einzug in Android, die kontinuierlich mit jeder neueren Version verbessert und dokumentiert wurde. Damit feiert der komfortable Weg, zugängliche Apps unter Android zu entwickeln, im Jahr 2019 zehnjähriges Jubiläum – eine gute Gelegenheit für einen Blick auf den Status quo.

Stolperfallen verhindern

Eine App unter Android zugänglicher zu gestalten, erfordert in der Regel keine größeren Umstrukturierungen des Quellcodes. Vielmehr gilt es, ähnlich wie bei der Unterstützung unterschiedlicher Displaygrößen einige grundsätzliche Empfehlungen und Ergänzungen zu beherzigen – idealerweise ab dem Start der Entwicklung. Bereits kleine Anpassungen während des Erstellens der Layoutdateien haben einen großen positiven Effekt auf die Zugänglichkeit der App.

Barrierefreiheit: Stolpersteine bei mobilen Anwendungen überwinden

- Teil 1: Herangehensweise und Hindernisse
- **Teil 2: Praktische Umsetzung für Android**

Allgemeine Designempfehlungen zum Gestalten zugänglicher Apps hat Google im Rahmen der Dokumentation zum Material Design veröffentlicht. Im Folgenden stehen daher konkrete Maßnahmen für Entwickler nativer Apps auf Basis der offiziellen Richtlinien von Google im Fokus. Die Codebeispiele konzentrieren sich vorwiegend auf die jeweils ausschlaggebenden XML-Attribute in den Layout-Ressourcen. Diese statischen Angaben sind zu wählen, wenn sich die Werte nicht zur Laufzeit der App ändern. Sollte sich das Aussehen der Bedienelemente beispielsweise durch Nutzerinteraktion dynamisch anpassen, gilt es, die jeweiligen Werte programmatisch zu setzen, um eine barrierefreie App-Bedienung zu gewährleisten. Die Beschreibung der XML-Attribute der View-Klasse hilft dabei, die passende dynamische Methode schnell zu finden.

Beschriftung von Elementen

Um die Funktion eines Bedienelements über einen Screenreader zu vermitteln, sollten Entwickler dem Bedienelement eine aussagekräftige Beschriftung (Label) gegeben. Dabei müssen sie das Vorleseverhalten des Screenreaders berücksichtigen und eventuelle textuelle Redundanzen vermeiden. Die meisten Screenreader einschließlich TalkBack nennen automatisch den Elementtyp, sodass zum beispielsweise "Speichern" eine bessere Beschriftung als "Speicher-Button" ist. Darüber hinaus greift das gewohnte Wechseln der Ressourcen abhängig von der (Sprach-)Konfiguration des Geräts. Die üblichen Regeln zum Lokalisieren von Android-Apps gelten somit auch für Beschriftungen im Interesse einer verbesserten Zugänglichkeit.

Eingabefelder

Bei einem Eingabefeld wie `EditText` lässt sich das Label folgendermaßen über den Hinweistext setzen (ab API-Level 1):

Statisch (XML):

```
<EditText
    ...
    android:hint="@string/label" />
```

Dynamisch (Kotlin):

```
editText.hint = getString(R.string.label)
```

Text- und grafische Elemente

Bei einem Textelement wie `TextView` ist das Setzen eines Labels in der Regel nicht erforderlich, weil der Screenreader den angezeigten Text automatisch erfasst und wiedergibt. Umso wichtiger ist dafür eine klare und verständliche Ausdrucksweise.

Bei einem grafischen Element wie `ImageView` oder `ImageButton` lässt sich das Label – analog zum Alternativtext in HTML – über eine zusätzliche Beschreibung setzen (ab API-Level 4):

Statisch (XML):

```
<ImageButton
    ...
    android:contentDescription="@string/label" />
```

Dynamisch (Kotlin):

```
imageButton.contentDescription = getString(R.string.label)
```

Ausschluss grafischer Elemente

Grafische Elemente, die ausschließlich einen dekorativen Zweck erfüllen, sollten Entwickler als irrelevant für den Screenreader markieren. Das Vorgehen hängt vom minimal unterstützten API-Level der App ab und ist wie folgt möglich:

Statisch (XML):

```
<!-- ab API-Level 4... -->
<ImageView
    ...
    android:contentDescription="@null" />

<!-- ...oder ab minimalem API-Level 16 -->
<ImageView
    ...
    android:importantForAccessibility="no" />
```

Dynamisch (Kotlin):

```
// ab API-Level 4...
imageView.contentDescription = null

// ...oder ab minimalem API-Level 16
imageView.importantForAccessibility =
  View.IMPORTANT_FOR_ACCESSIBILITY_NO
```

Label-Elemente

Ab API-Level 17 lässt sich zudem definieren, dass ein Element als Label für ein anderes Element dienen soll:

Statisch (XML):

```
<TextView
    ...
    android:text="@string/label"
    android:labelFor="@id/input_field" />

<EditText
    android:id="@+id/input_field"
    ... />
```

Dynamisch (Kotlin):

```
textView.labelFor = R.id.input_field
```

Gruppierung von Elementen

Logisch beziehungsweise inhaltlich zusammengehörige Elemente sollten gruppiert sein, sodass der Screenreader ihren Inhalt als zusammenhängende Einheit betrachtet und vorliest.

Die Gruppierung lässt sich folgendermaßen über einen fokussierbaren Container in Form einer beliebige Unterklasse von `ViewGroup` erreichen:

```
<LinearLayout
   ...>

  <RelativeLayout
    android:id="@+id/group1"
     ...
    android:focusable="true">

    <TextView ... />

    <TextView ... />

  </RelativeLayout>

  <RelativeLayout
    android:id="@+id/group2"
    ...
    android:focusable="true">

    <TextView ... />

    <TextView ... />

  </RelativeLayout>

</LinearLayout>
```

Eine `ViewGroup` mit `android:focusable="true"` bildet demnach eine Einheit, die der Screenreader als Ganzes anspringt und die enthaltenen Elemente zusammenhängend vorliest. Für obigen Code fokussiert das Vorleseprogramm die erste Gruppe `group1`, liest deren Inhalt vor und setzt nach einer Nutzerinteraktion bei der zweiten Gruppe `group2` seinen Dienst fort. Die einzelnen Kindelemente (hier die `TextViews`) sind dabei nicht mehr direkt fokussier- beziehungsweise ansteuerbar. Das bedeutet für die Nutzer weniger Navigationsaufwand und erspart ihnen zudem unnötige Pausen beim Erfassen von wichtigen, zusammengehörigen Informationen.

Ein Hinweis sei noch angebracht: `android:focusable` wirkt sich auf die Navigationsreihenfolge bei Nutzung von Eingabehardware wie einer Tastatur aus. Daher lässt sich ab Android 9 Pie (API-Level 28) das nur für Screenreader gültige `android:screenReaderFocusable` anstelle von `android:focusable` in Situationen benutzen, in denen Letzteres unerwünschte Seiteneffekte bei der Navigation hätte.

Gerichtete Navigation

Neben Touch-Steuerung bietet Android weitere Eingabemethoden an. Nutzer können unter anderem Hardware wie eine Tastatur oder ein Steuerkreuz anschließen. Insbesondere erlaubt Android Eingabemethoden, die eine barrierefreie Bedienung ermöglichen. Dazu zählen Sprach- und Schaltersteuerung, die bei motorischen Einschränkungen eingesetzt werden können.

Um eine App für alternative Bedienungsweisen zu rüsten, ist es wie in HTML wichtig, eine gerichtete Navigation sicherzustellen. Standardmäßig verhält sich Android so, dass es fokussierbare Elemente in der Reihenfolge anspringt, in der sie im Layout angeordnet sind. Häufig genügt das für eine angemessene Navigation, aber es gibt Situationen, in denen Entwickler die Navigation explizit durch Festlegen des Fokuswechsels korrigieren müssen.

Bei der Vorwärtsnavigation geht der Fokus beispielsweise durch Drücken der TAB-Taste einer Tastatur an das jeweils nachfolgende Element weiter. Um den Nachfolger festzulegen, lässt sich das Attribut `android:nextFocusForward` oder die Methode `setNextFocusForwardId(int)` verwenden:

```
<RelativeLayout
  ...>

  <Button
    android:id="@+id/button1"
    android:layout_alignParentTop="true"
    android:layout_alignParentRight="true"
    android:nextFocusForward="@+id/editText"
    ... />

  <Button
    android:id="@+id/button2"
    android:layout_below="@id/button1"
    android:nextFocusForward="@+id/button1"
    ... />

  <EditText
    android:id="@id/editText"
    android:layout_alignBottom="@+id/button2"
    android:layout_toLeftOf="@id/button2"
    android:nextFocusForward="@+id/button2"
    ... />

</RelativeLayout>
```

Die Fokusreihenfolge geht damit von `button1` zu `editText`, zu `button2`, was gleichzeitig der visuellen Anordnung entspricht. Ohne das Setzen von `android:nextFocusForward` würde der Screenreader die Reihenfolge der Elemente im Layout verwenden: `button1` zu `button2` zu `editText`.

Bei der unter anderem durch die Pfeiltasten einer Tastatur ausgelösten Richtungsnavigation geht der Fokus an das jeweils in einer bestimmten Richtung liegende Element weiter. Um das Element festzulegen, das den Fokus für die jeweilige Richtung erhalten soll, lassen sich die Attribute `android:nextFocusUp`, `android:nextFocusDown`, `android:nextFocusLeft` und `android:nextFocusRight` beziehungsweise die Methoden `setNextFocusUpId(int)`, `setNextFocusDownId(int)`, `setNextFocusLeftId(int)` und `setNextFocusRightId(int)` verwenden.

```
<Button
  android:id="@+id/button1"
  android:nextFocusRight="@+id/button2"
  android:nextFocusDown="@+id/editText"
  ... />

<Button
  android:id="@id/button2"
  android:nextFocusLeft="@id/button1"
  android:nextFocusDown="@id/editText"
  ... />

<EditText
  android:id="@id/editText"
  android:nextFocusUp="@id/button1"
  ...  />
```

Ausgehend von `button1` geht eine Rechtsnavigation zu `button2`, umgekehrt führt eine Linksnavigation von `button2` zu `button1`. Eine Navigation von einem der Buttons nach unten fokussiert `editText`, und von dort geht es zu `button1`, sobald der Nutzer hochnavigiert.

Zugängliche benutzerdefinierte Views

Manchmal reichen die gegebenen UI-Komponenten für die Umsetzung der Projektanforderungen nicht aus, sodass eine benutzerdefinierte `View`-Klasse, eine sogenannte Custom View, erforderlich ist, die Entwickler ebenfalls zugänglich gestalten sollten. Die hauptsächliche Arbeit besteht darin, spezielle API-Methoden im Interesse der Barrierefreiheit zu implementieren – allen voran `sendAccessibilityEvent(int)` und `onPopulateAccessibilityEvent(AccessibilityEvent)`. Zusätzliche Informationen finden sich in der offiziellen Dokumentation und der Beispielimplementierung.

Probieren geht über Studieren

Am anschaulichsten können Entwickler eine App auf Barrierefreiheit testen, indem sie sie mit unterschiedlichen Bedienungshilfen benutzen. Es empfiehlt sich daher für Entwickler, den mittlerweile zur Android Accessibility Suite gehörenden Screenreader TalkBack (oder Voice Assisant bei neueren Samsung-Geräten) über die Systemeinstellungen Bedienungshilfen zu aktivieren. Die Übungslektionen, die beim ersten Aktivieren erscheinen und auch danach noch zur Verfügung stehen, helfen dabei, ein Gefühl für die Benutzung eines mobilen Screenreader zu bekommen. Nach einem kurzen Moment der Gewöhnung an die Bedienung mittels Screenreader steht dem Test der eigenen App auf Herz und Nieren nichts mehr im Wege. Weiterhin sollten Entwickler und Tester verifizieren, ob ihre App auch mit veränderter Schrift- und Anzeigegröße noch hinreichend gut bedienbar ist.

Selbstverständlich liefern nicht zuletzt Benutzertests wertvolle Einblicke und damit nicht zu unterschätzendes Feedback zu Hürden in der zugänglichen Benutzerführung. Neben dem manuellen Testen gibt es automatisierte Tools, die eine App auf Barrierefreiheit überprüfen.

Analyse mit Lint

Mit Lint bietet der Android-Werkzeugkasten eine Anwendung, um frühzeitig Programmierfehler und Unzulänglichkeiten aller Art im Projekt mit statischer Code-Analyse zu identifizieren. Dazu zählen einige Mängel bei der Barrierefreiheit: Layoutelemente zu finden, bei denen das `contentDescription`-Attribut fehlt, ist beispielsweise für Lint ein Leichtes.

Die Analyse lässt sich in Android Studio über den Menüpunkt Analyze | Inspect Code ausführen. Die Ergebnisse zeigt die IDE anschließend im Bereich Inspection Results unter Android | Lint | Accessibility an.

Alternativ lässt sich die Analyse über die Kommandozeile mit dem Gradle-Wrapper im Projekt-Wurzelverzeichnis starten:

```
./gradlew lint
```

Android Studio generiert anschließend einen Lint-Bericht und gibt die Pfade zu der XML- und der HTML-Version aus.

Accessibility Scanner

Die Google-App Accessibility Scanner gibt durch den Scan von Benutzeroberflächen Empfehlungen zur Verbesserung der Barrierefreiheit. Sie ist als Bedienungshilfe implementiert und wie der Screenreader über eine passende Systemeinstellung zu aktivieren. Konkret analysiert die App folgende Punkte:

- Inhaltsbeschriftungen (Labels)
- Größe von Touch-Zielen

- anklickbare Elemente
- Text- und Bild-Kontraste

Weitere Details zur Benutzung und zur Auswertung der Ergebnisse sind auf der Hilfsseite des Tools zu finden.

Node Tree Debugging

Werkzeuge für Barrierefreiheit wie Screenreader haben eine eigene Sicht auf die Benutzeroberfläche einer App. Das in TalkBack enthaltene Node Tree Debugging bietet eine Ansicht auf die Hierarchie und die Attribute von Bedienelementen aus der Perspektive der Werkzeuge. Es ermöglicht, die Baumstruktur eines App-Screens als Log auszugeben. Wie das genau funktioniert und wie die Ergebnisse zu interpretieren sind, ist in der Anleitung detailliert beschrieben.

Espresso

Das Framework Espresso ermöglicht das Schreiben automatisierter Benutzeroberflächentests. Bestehende Tests lassen sich ohne viel Aufwand um eine Überprüfung auf Mängel in der Barrierefreiheit erweitern. Folgender Code – mit aktualisierter `import`-Anweisung – in der Initialisierungsmethode eines Espresso-Tests konfiguriert ihn auf das Überprüfen der Barrierefreiheit einer View bei jeder Interaktion über `ViewAction`.

```
import
  androidx.test.espresso.accessibility.AccessibilityChecks

@RunWith(AndroidJUnit4::class)
@LargeTest
class AccessibilityChecksIntegrationTest {
  companion object {
    @BeforeClass @JvmStatic
    fun enableAccessibilityChecks() {
      // aktiviert die zusätzlichen Checks zur Barriefreiheit
      AccessibilityChecks.enable()
    }
  }
}
```

Fortschritte und Empfehlungen

In den letzten zehn Jahren gab es deutliche Fortschritte für die Barrierefreiheit unter Android, und jede neue Version des mobilen Betriebssystems bringt weitere Verbesserungen in dem Bereich. Neben einem auf Zugänglichkeit ausgelegtem App-Design ist das Beherzigen simpler Empfehlungen und Anpassungen mit vergleichsweise geringem Aufwand während der Entwicklung der Layout-Dateien bereits die halbe Miete zur barrierenfreien App. Wichtig sind die sinnvolle Beschriftung von Elementen, die Gruppierung zusammengehöriger Komponenten sowie das Anpassen der Navigation für Screenreader beziehungsweise Eingabehardware.

Damit App-Designer und -Entwickler eine Vorstellung haben, mit welchen Herausforderungen Menschen mit Einschränkungen bei der Bedienung mobiler Apps zu kämpfen haben, sei ihnen ans Herz gelegt, ihre Anwendungen mit den unter Android zur Verfügung stehenden Bedienungshilfen ausgiebig zu testen. Der Perspektivwechsel hilft, Schwachstellen in der Zugänglichkeit schnell zu identifizieren und zu beheben.

Für Entwickler lassen sich allgemeine Empfehlungen folgendermaßen zusammenfassen:

1. Keep it simple: einfache Struktur der App, Kernfunktionen klar priorisieren und eindeutig benennen,
2. auf Lesbarkeit wie Schriftgröße und Kontrast achten – nicht nur für Sehbeeinträchtigte,
3. flexibles Auto-Layout benutzen, sodass Elemente bei der Schriftvergrößerung nicht verrutschen oder das Layout verzerren,

4. alle Navigationselemente, Schaltflächen und Bilder für die Sprachausgabe mit eindeutigen Texten hinterlegen und
5. per Spracheingabe alle Funktionen steuerbar und alle Inhalte durchsuchbar machen. (rme)

Roman Zimmer
arbeitet bei adesso mobile solutions als Senior Software Engineer. Neben der Beratung und Realisierung von Full-Stack-Projekten aller Art ist er seit 2011 leidenschaftlicher Android-Entwickler.

Alexander Huber
arbeitet als Android-Entwickler bei adesso mobile solutions am Standort Berlin.

Martin Kinting
ist Niederlassungsleiter der adesso mobile solutions am Standort Berlin und Senior Sales Manager unter anderem in den Bereichen Banking und Life Science.