Quick Guide: Accessible Mobile Applications

Posted on:July 22, 2021 Written by:Chris Porter

Whether planning a new project or revisiting an existing one, there are a few techniques you can adopt to ensure your mobile app is accessible for a variety of users.

How do you ensure your mobile application is accessible?

To date, most accessibility laws around the world are largely based, or directly point to W3C WAI's accessibility guidelines (WCAG). The European Union has issued harmonized standards (EN 301 549) along with directives (pointing to such standards) requiring developers to make mobile apps and services accessible for all by 2021 for the public sector and 2025 for the private sector. The EU standard is based on WCAG 2.1, and requires a minimum compliance level equivalent to WCAG 2.1 AA (while *recommending* AAA compliance for around 28 criteria). Furthermore, EN 301 549 offers accessibility success criteria for web pages (rendered on any device), along with criteria for non-web documents and non-web software (e.g. mobile apps, platform software, smartwatch apps, ATMs, e-readers, home appliances, car dashboards, airplane seatbacks and so forth) – all largely based on the latest W3C WAI guidelines (WCAG 2.1). However, this standard goes a step further and includes aspects such as the use of biometrics and operable parts (e.g. physical toggle switches) in applications.

WCAG 2.1 incorporates recommendations (and success criteria) that directly address mobile accessibility. For instance, does your app require users to "shake" their device to activate certain actions? WCAG 2.1 tackles specific situations where alternatives might be required for people whose device is, say, mounted in a fixed position. The WCAG2ICT working group also provides further guidance on applying the widely adopted WCAG 2.0 guidelines to technologies other than web applications.

In principle, all of the information and user interface components presented through an app must be visible to users in some way or another and that all users are able to achieve their goals. Furthermore, the content and interface itself should not be confusing and must also be authored in a way that is interpreted and rendered reliably across different devices, operating systems and associated assistive technologies (ATs). This may sound overwhelming, however if you follow these few simple steps, you and your team will start developing an accessibility-first mindset and in turn, **accessible and usable** mobile applications. In case things start getting out of hand and you need more assistance, feel free to reach out by clicking on this link.

Let's get started!

Developing native mobile apps should follow an accessibility-first approach. This of course requires developers to be aware of and understand platform-specific approaches, techniques

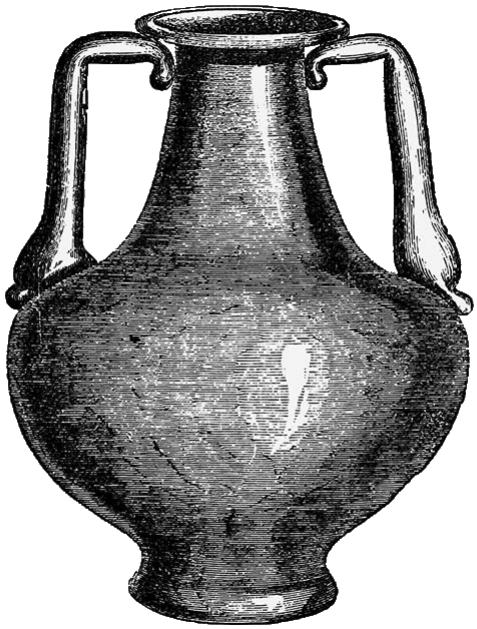
and technologies that will help them build accessible applications. Having said that, whichever platform you're building for, the same underlying objective remains – that of making your native application accessible through adherence to WCAG success criteria. How each success criteria is best translated for iOS or Android is a different matter altogether.

You have a great opportunity to introduce an accessibility-first mindset, which translates into the following 4-step pattern as part of your technology delivery process:

1. Get the docs for your target platform
2. Understand the guidelines and translate for your platform
3. Test features and journeys
4. Involve target users and get audited

So, let's dig slightly deeper.

Step 1: iOS or Android? Get the docs!



Both iOS and Android ship with built-in accessibility features as well as accessibility APIs that allow developers to build accessible apps.

iOS

iOS is well known for its accessibility features and services, and its equally powerful accessibility APIs. Each iOS device ships with *VoiceOver*, a powerful screen reader for blind and low-vision users.

Android

Similarly, Android ships with powerful accessibility features and services, along with developer tools to help you build accessible mobile applications. Android devices offer *TalkBack*, a powerful screen reader specifically designed for blind and low-vision users.

Equipped with these resources, you should now move on to understand the respective accessibility guidelines – and how these resources can help you achieve them.

Step 2: Understand the guidelines and translate for your platform

Making native mobile apps accessible requires you to first understand WCAG 2.1 guidelines and associated success criteria. The challenge is to translate each recommendation into platform-specific code. Unlike the web, there will be a distinct approach to making mobile apps accessible for different platforms – however the same underlying principles will apply.

Having said that, iOS and Android developers will have no issues understanding what needs to be done once they know which guidelines to follow, what criteria to apply and how to test them properly. So, let's dig in.

By definition, and based on guidelines that are applied almost universally (WCAG), an accessible mobile application needs to be perceivable, operable, understandable, and robust. This means that all of the information and user interface components must be visible to users in some way or another and that all users are able to achieve their goals – irrespective of the users' abilities or limitations – which could be both physical and cognitive.

The following sections will outline each of these principles, along with some example guidelines to get you started.

IMPORTANT: *Please bear in mind that the following sections are just meant to outline each principle along with some examples. This is not a comprehensive accessibility checklist. Your app might still have accessibility issues even if the aspects below are tackled. In case you need more information or assistance on this matter, feel free to reach out by clicking on this link.*

Principle 1: Perceivable

If you're building for iOS or Android, then you might be targeting various device types and form-factors (e.g. phones, tablets, TVs, watches, appliances and others). Even if it's just phones you're interested in, you need to remember that, particularly for the Android platform, resolution and screen sizes can vary quite drastically. Also, you can never assume that a user will interact with your app visually, or in optimal lighting conditions. Basically, information and UI components need to be perceived by all of your users, irrespective of their disability and context of use. Let's give examples of some guidelines that are there to ensure adherence to this principle.

Text alternatives – in most cases, any non-text content in your app (e.g. images, videos) might need to be represented using a text-alternative, if possible and where applicable. Sometimes, this means providing proper names to describe the purpose of an element, be it a video, control or an image. In iOS this could be as simple as setting the *accessibilityLabel* property, while for Android this would be the *android:contentDescription* property – both of which are announced by VoiceOver and TalkBack respectively.

Refer to Guideline 1.1 for more details and examples.

Time-based media: If you have pre-recorded audio or video streamed through your app, then you need to ensure that the same information conveyed in the stream is also represented in a textual format. Captions could also be provided for pre-recorded media, which could also be presented synchronously. In more advanced scenarios, live audio should also be accompanied with captions. Stricter compliance to these guidelines could present technical, but interesting, challenges. Success Criterion 1.2.6 recommends the introduction of sign-language interpretation for pre-recorded audio content, however this is only for developers wishing to adhere to the strictest compliance levels (AAA).

Refer to Guideline 1.2 for more details and examples.

Distinguishable: you need to make it easier for your users to distinguish foreground content from background content. For instance, the use of colour only to convey important information (e.g. a warning) is not recommended unless this is accompanied with other cues (e.g. use of text cues or additional visual cues, such as underlining). Furthermore, any text or images of text must generally have a contrast ratio of at least 4.5:1 (except in some cases), making your text readable for a wider range of users and in a wider range of contexts. Furthermore, any in-app text (including captions) should be re-sizable without the use of magnifiers up to 200% without clipping content or reducing functionality. Another important criterion presented by this guideline is that in-app content should not be restricted to only one device orientation (e.g. landscape) – unless this is essential.

Refer to Guideline 1.4 for more details and examples. In case you need more information or assistance on this matter, feel free to reach out by clicking on this link.

Principle 2: Operable

Mobile devices are mainly operated via touch, through virtual keyboards and gesture-driven interaction. Having said that, some users also attach external input modalities to improve interaction efficiency, including physical keyboards and switches.

Keyboard accessible: if a user is using an external keyboard, or a switch interface, then you must ensure that all of the functionality is operable through these alternative input modalities. Having said that, criteria for this guideline include aspects such as making sure users are not trapped in a specific area of the application without being able to go back.

Refer to Guideline 2.1 for more details and examples.

Navigable: Users making use of screen-readers such as VoiceOver and TalkBack will use swipe gestures to move forwards and backwards across the interface, hopping from one element of your application to the next, while listening to prompts describing what each element's purpose is. You must ensure that the focus order (akin to tabbing sequence) is logical and that users can traverse all elements efficiently without getting stuck along the way. Generally speaking, focus order is determined by the physical layout of controls in your views/activities, however at times developers tend to changes these programmatically, and in such cases, extra care is necessary.

Refer to Guideline 2.4 for more details and examples. In case you need more information or assistance on this matter, feel free to reach out by clicking on this link.

Input modalities: This guideline is meant to help users operate applications using various means. For instance, if your app depends on motion to actuate certain functionality (e.g. shake device to undo), you should also provide means by which such functionality could be disabled to prevent accidental use (e.g. due to some involuntary movement), while at the same time providing alternative methods to access the same functionality. Also, components such as buttons or input elements should be large enough to avoid accidental selections, while being placed in positions that are easy to reach even when the device is held in different positions.

Refer to Guideline 2.5 for more details and examples. In case you need more information or assistance on this matter, feel free to reach out by clicking on this link.

Principle 3: Understandable

Both the information and the UI must be understandable for all users. This is particularly challenging when screen size is limited, and one needs to be careful what to present and when.

Readable: Any text should be easy to understand and follow. Furthermore, important information (e.g. warnings) should be immediately visible without requiring the user to scroll down to see it. This is particularly important for users who are using screen magnifiers, which further limit page viewability. Also, in multi-lingual apps, it is important that localized labels are used to benefit from built-in device locales and associated services (e.g. multi-lingual narration).

Refer to Guideline 3.1 for more details and examples.

Predictable: As users start using an app, they will start learning about the various components used and how they are laid out across the screen. It is therefore very important that if similar components are used across different pages, then these are presented consistently throughout. This becomes challenging when an app is rendered on different screen sizes, and therefore testing for placement is important. UI consistency supports users in terms of interaction efficiency and UI predictability. It is also important that if functionality is dependent on screen layout, screen-reader users are also informed on how to perform actions correctly – depending on their current orientation.

Refer to Guideline 3.2 for more details and examples. In case you need more information or assistance on this matter, feel free to reach out by clicking on this link.

Input Assistance: In case users make mistakes while using your application, you should help them recover as efficiently as possible, displaying items that are in error with proper descriptions. Furthermore, input fields should be supplemented with labels, clearly explaining what is required. When it comes to actions that have legal or financial implications, users should be assisted to avoid making errors in the first place (error prevention). A simple confirmation step would generally do the trick, however there are alternative approaches one can opt for, including introducing “undo” actions (reversible actions). A final example relates to the use of custom developed touchscreen gestures to interact with some functionality in the app. In such cases, developers might unknowingly exclude a large portion of the user population, either because some users might find it hard to discover and remember custom gestures or because they are simply unable to perform them. In such cases, users should be

assisted to learn how to perform such gestures or are offered alternative interaction approaches.

Refer to Guideline 3.3 for more details and examples. In case you need more information or assistance on this matter, feel free to reach out by clicking on this link.

Principle 4: Robust

Irrespective of what device is being used, and whether assistive technologies are adopted by users, your content and functionality must be robust enough to be interpreted and used correctly.

People who have been making use of iOS or Android accessibility features for some time would of course expect all apps to behave in a certain way. This is why you should always make use of the platform's APIs to ensure consistent rendering and behaviour across devices and associated accessibility features (e.g. screen-reader, magnifier, text re-sizing, voice control, switch-interfaces etc…).

Furthermore, while developing an app, you should always consider providing appropriate methods for data entry, while minimising interaction steps whenever possible. For instance, if your app is expecting numerical input on a certain field, then you should programmatically instruct the platform to display an appropriate virtual keyboard that supports such input. Hints could be used to instruct platforms to display appropriate keyboards to support users when interacting with specific input fields for decimal values, phone numbers, emails, web searches and so forth.

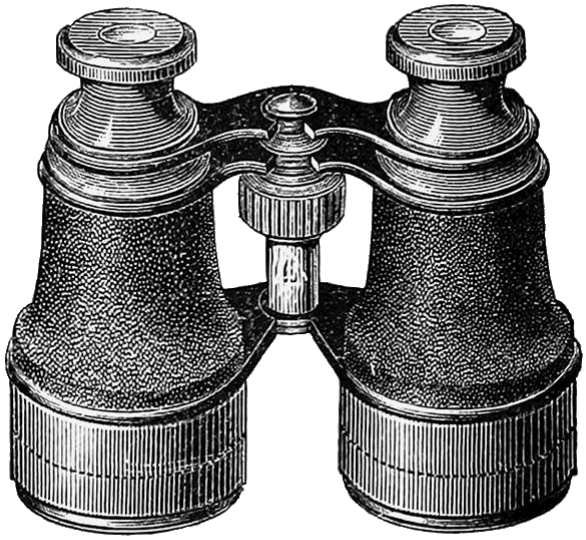
In case you need more information or assistance on this matter, feel free to reach out by clicking on this link.

Is this all?

The above is only a brief treatment of these guidelines – with some examples on the application of WCAG 2.1 guidelines for mobile app development. You should also consider other resources such as:

- Mobile Accessibility: How WCAG 2.0 and Other W3C/WAI Guidelines Apply to Mobile
- WCAG 2.0 Techniques that Apply to Mobile (w3.org)
- WCAG2ICT Overview | Web Accessibility Initiative (WAI) | W3C
- Guidance on Applying WCAG 2.0 to Non-Web Information and Communications Technologies (WCAG2ICT) (w3.org)

Step 3: Test features and journeys



After completing a feature or user journey, test your work. It is recommended that you test incrementally, rather than everything at one go. Consider adopting at least one of the following approaches:

Manual testing

1. Try to use the feature or complete the entire journey using the platform's built-in screen-reader (e.g. VoiceOver or TalkBack). VoiceOver (on iOS) or TalkBack (on Android) are assistive technologies that ship by default with the respective mobile operating systems, and are designed to help blind and visually impaired users access native apps. Try to navigate and complete actions using the **default gestures** (e.g. swipe left to select the next element, double tap to operate element etc…). Can you access all the elements on the screen? Try to navigate the feature or entire journey without looking at the device. Is the screen reader giving you enough information to understand the purpose of the page? Ensure you have meaningful screen *headings*. Are input elements announced in a meaningful way? Make sure controls are labelled properly. Is the screen reader producing too much audio clutter? Make sure that decorative elements are marked as such so that screen readers bypass them. Can you complete the intended task? Also, are lists announced? Make sure that lists are communicated appropriately, giving your users a better contextual understanding of the page content. Have you got long running processes in your app? If so, ensure that status messages are communicated appropriately (e.g. "Searching", "Found 20 items"). Are you communicating important alerts? Ensure these are also communicated appropriately to assistive technology users. Have you got any audio or video resources embedded in your web application? Make sure videos are captioned and that audio transcripts are also available.
2. List down areas where difficulties were noted.

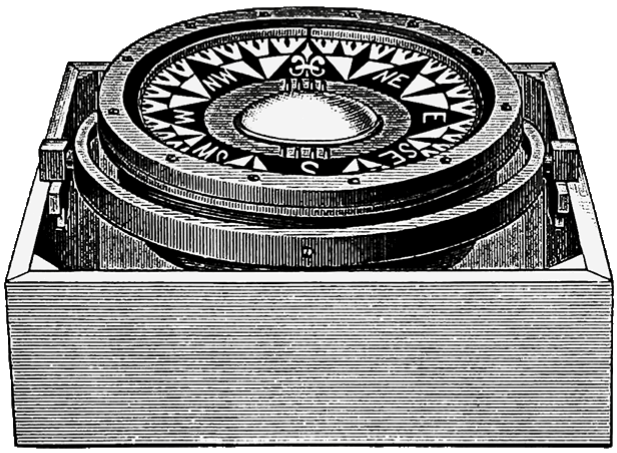
3. Discuss all difficulties encountered during a team meeting – and plan for possible updates/fixes.

Semi-automated testing

1. Along with manual testing, you can also review your native app using purpose built tools.
2. There are a number of tools you can use, including: Mobile Accessibility Analyser (from evinced.com), iOS Accessibility Debugger (from apple.com), and Accessibility Scanner for Android Apps.
3. Follow official accessibility testing recommendations for Android and iOS.
4. List down areas where difficulties were noted.
5. Discuss all difficulties encountered during a team meeting – and plan for possible updates/fixes.

The tests discussed here are definitely not comprehensive and are only meant to put your development efforts on the right path. Complete accessibility audits by specialists will give you a true picture of how accessible your technology and digital content is. If you need more assistance, feel free to reach out by clicking on this link.

Step 4: Involve target users and get audited



Expert accessibility audits will give you a true picture of the state of your technology. These audits, which are generally carried out incrementally, are done by domain experts during which rigorous testing (both manual and automated) is carried out, following any applicable legal framework and guidelines at national and international levels. Auditors will generally produce detailed accessibility reports, based on WCAG success criteria across the four guiding principles and at an agreed level of conformance. Audits also generally include technical recommendations for any technology you are working with.

It is also recommended to test your technology with target users. This is not always an easy task, and it might feel slow until you get the logistics sorted. However, organisations such as FITA are there to support you and your team, and will happily review your work.

Do you need more assistance on this matter? Feel free to reach out by clicking on this link.
Categorized in:Accessibility, Mobile accessibility Tagged as:accessibility, Android, mobile accessibility, Talkback, VoiceOver, wcag

Published by**Chris Porter**

View all posts by Chris Porter