Native App Accessibility Testing Methodology

- Step 1: Identify devices
- Step 2: Define application functionality
- Step 3: Test critical issues
- Step 4: Test mobile issues
- Step 5: Test mobile assistive technology and feature support

ICT Accessibility Testing Symposium

Table of Contents

Native Mobile Testing Methodology

The ICT Accessibility Testing Symposium has developed a methodology for evaluating the accessibility of native apps. This document is an amalgamation of accepted native app accessibility testing standards from around the world, including additional developments from the ICT Accessibility Testing Symposium's Native App Sub-Committee (for more information, see Acknowledgements on page 16).

WCAG2 success criteria are applicable to native apps, however, not all aspects of mobile accessibility are specifically covered by WCAG2. It is the opinion of this committee that merely conforming WCAG2 (or WCAG 2.1) does not provide for a fully accessible experience for users with disabilities.

Please note that this methodology does not include those issues already included in WCAG2, however does include issues identified in WCAG2.1. This guide was written with the intent to clarify the unique needs of users with disabilities who use native apps and to raise the bar for the web development community. This is a work-in-progress, and, as such, we do not make a claim that conforming to these requirements will ensure that your native app is fully accessible to all users.

A note on WCAG2.1

The Committee decided that it was important to also include issues added in WCAG2.1, so that testers who were testing against WCAG2 would also benefit from the mobile-related errors published in WCAG2.1. This methodology differs from WCAG2 and WCAG2.1 in some areas, and these are detailed in the Test Cases documents.

Mobile sites versus native apps

There is a great difference between mobile sites and native apps – native apps utilize a completely different codebase. Therefore, the ICT Accessibility Testing Symposium has decided to separate the native app methodology from the mobile site testing methodology.

A note on hybrid native apps

Hybrid mobile apps consist of both HTML and native code. The HTML may be included as a specific page within an application, or within a specific container. When testing a hybrid app, you will need to use this methodology and the Mobile Site Accessibility Testing Methodology.

Relationship with other Native App documents

There are two overview documents:

- Native App Methodology (**this current document**)
- About Native App Testing – Devices, assistive technologies and capturing errors

There are three sets of test cases documents, which detail how to test a particular requirement in the methodology, why it is important and example failures and passes of the requirement.

- Native App Critical Test Cases
- Native App Test Cases
- Native App Assistive Technology Test Cases

Native App Testing Methodology

Overview

Please note that this methodology does not include those errors already included in WCAG2. In order to ensure your native app is fully accessible, you need to meet WCAG2 and this native app testing methodology.

The following steps should be followed to ensure that a native app is accessible to all users:

- Step 1: Identify devices (page 8)
- Step 2: Define application functionality (page 9)
- Step 3: Test critical issues (page 10)
- Step 4: Test mobile issues (page 11)
- Step 5: Test mobile assistive technology and feature support (page 14)

Step 1: Identify devices

What needs to be tested is dependent on the native app and chosen devices.

Identify devices and browsers to test on

Recommended devices and browser combinations:

- iPhone
- iPad
- Android phone

Other devices to consider

- Android tablet (for example, Samsung Tab A or Chromebook)
- Alternative devices such as a Kindle device

Recommendations

Test on the latest version of iOS.

Test on latest two versions of Android.

Step 2: Define application functionality

Through your understanding of the purpose of the native mobile application, define which functionality is critical to its purpose and use. That must be tested for efficacy, operability and workflow from a user experience perspective.

All functionality should be accessible within the native application; however, it is important to define and include the critical functionality for each individual app as priority in your testing.

A good question to ask: how would the experience be impacted if the functionality failed, the content could not be reached and/or the experience caused a barrier to the user?

Common elements to test

- Navigation – Menus, header, footer
- Landing screen(s)
- Emergency sections and content
- Login flows
- Settings
- Account and profile
- Contact Us
- Real-time updates (eBay, Uber)
- Privacy policy, Terms and Conditions
- Interactional functionality (adding items to a shopping cart, payment details, live chat, selections for a product in a catalogue, scanning a barcode, VR, QR code)
- Help section
- Widgets (calendars, date pickers)
- Third-party integrations (geo-locational maps)
- High-traffic areas

Step 3: Test critical issues

Exit traps

Applies to: All users

Ensure there is always an accessible actionable item (e.g. a close button that meets color contrast requirements and has an accessible name) that closes any feature that overlays the current screen (such as a full-page ad).

Swipe / scroll traps

Applies to: Touch users

Ensure you do not override standard mobile touch functions (swiping, scrolling, etc.) on the majority of the screen.

Text-to-speech traps

Applies to: Screen reader users

If the app has an ability to provide content via text-to-speech, the screen reader user must be able to pause or stop the app speaking in a simple manner, e.g. by performing a swipe on the screen.

Headset traps

Applies to: Headset users

Headset users must always be able to pause media (audio or video) content by using the Pause/Play control on the headset.

Layer trap

Applies to: All users (but mostly encountered by screen reader users)

The user must not be trapped on a non-visible layer.

Step 4: Test mobile issues

Alternatives

Functionality that can be operated by device motion or user motion, interaction and/or gesture can also be operated by user interface components, and responding to the motion, interaction and/or gesture can be disabled to prevent accidental actuation, except for certain situations (for more information see SC 2.5.4: Motion Actuation).

Any touch gesture must have an alternate gesture, such as a link (for more information see SC 2.5.1: Pointer Gestures and SC 1.4.13: Content on Hover or Focus).

Alternatives are provided for geolocation functionality that is mandatory (for example, requiring a specific geolocation before functionality appears), unless the geolocation is essential for legal reasons or doing so would invalidate the activity. This applies to geolocation via GPS, user statement, IP address or other methods.

Changes of state of non-standard controls (e.g. hamburger menu, star ratings) are clearly indicated.

Audio cues have an equivalent, accessible, visual cue.

Status messages are available to all users without receiving focus (for more information see SC 4.1.3: Status Messages).

All abbreviations are expanded the first time they are used on the page, or a glossary of abbreviations and their expansions is provided (for more information see SC 3.1.4: Abbreviations).

Where the text requires reading ability more advanced than Flesch Kincaid level 8, a summary or description of content is provided (for more information see SC 3.1.5: Reading Level).

Controls, primary headings, links, field labels and page titles are not ambiguous when read aloud.

Display

Web pages do not contain more than three flashes in a one second period (for more information see SC 2.3.2: Three Flashes).

Changes of context must always be user-initiated unless it is time-sensitive or an emergency (for more information see SC 3.2.5: Change on Request).

Size of touch targets is at least 44 by 44 CSS pixels (approximately 7 to 10 millimeters) (for more information see SC 2.5.5: Target Size).

Touch targets have sufficient inactive space between them (inactive space of at least 22 by 22 CSS pixels, or 4 to 5 mm, should be provided around active elements).

Do not use fixed size containers for blocks of text unless the display is essential.

Justified text has not been used.

Text and actionable items (including text and non-actionable items that will become actionable) or items that convey meaning should have a minimum color contrast ratio of 4.5:1 when compared with the surrounding background.

The native app can be used in portrait mode (for more information see SC 1.3.4: Orientation).

The native app can be used in landscape mode (for more information see SC 1.3.4: Orientation).

The native app does not swap orientation unexpectedly.

Animation triggered by interaction can be disabled (for more information see SC 2.3.3: Animation from Interactions).

Actionable items

When additional content appears on hover, focus or input, it is dismissable, hoverable and persistent (for more information see SC 1.4.13: Content on Hover or Focus).

Native UI controls, objects, alerts and elements have been used.

Where text links are used, the text is visually descriptive (for more information see SC 2.4.9: Link Purpose (Link Only)).

When direct input via the keyboard is not required, provide options for the user to achieve the same result (e.g. use dropdown, radio buttons and checkboxes, etc.).

Infinite scrolling has not been used.

Color alone should not be used to indicate actionable items (if not underlined) within inline text. A secondary method, such as underlines, should be used in addition to color.

Actionable elements are triggered only on removal of touch (for more information see SC 2.5.2: Pointer Cancellation).

Navigational aids

Visual indicators (such as arrows, next and previous buttons) have been used to indicate swipe or scroll areas or additional functionality.

Single character key shortcuts can be turned off, modified by the user or are active only on focus (for more information see SC 2.1.4: Character Key Shortcuts).

Blocks of content have descriptive headings (for more information see SC 2.4.10: Section Headings).

Prior to starting a process, users are warned if there is a timeout and the length of time of inactivity that will trigger the timeout. Please note you will also need to comply with WCAG2 SC 2.2.1 Timing Adjustable (for more information see SC 2.2.6: Timeouts).

Navigation features such as back buttons, breadcrumbs, next and previous buttons are provided (for more information see SC 2.4.8: Location).

Audio and video

All audio and video have an accessible transcript (for more information see SC 1.2.8: Media Alternative (Prerecorded)).

Captions must be Closed Captions.

Live captions and audio descriptions are provided for any live audio or video.

Forms

Visual and audio CAPTCHAs are not used.

All complex forms contain context-sensitive help as instructions at the beginning of the form and/or specific instructions at each field (for more information see SC 3.3.5: Help).

All submitted forms are reversible by the user, checked for errors by the application or confirmed prior to submission (for more information see SC 3.3.6: Error Prevention (All)).

Field labels are positioned adjacent to their input field and appear closest to their respective input field in relation to other field labels and other input fields.

Fields must have an associated visible label which is also programmatically associated with the field (please note that placeholding characters do not meet this requirement) (for more information see SC 2.5.3: Label in Name).

If there is visible text label for a input field or component, the accessible name matches the visual name (for more information see SC 2.5.3: Label in Name).

Forms interact appropriately with the keyboard (e.g. providing submission on Enter, moving between fields and can be dismissed).

Step 5: Test mobile assistive technology and feature support

All actionable items and important content can be accessed and activated by assistive technologies or mobile features.

Native App inheritance of settings

Native applications should permit the following user preferences to be inherited from platform settings:

- Color
- Contrast
- Font type
- Font size
- Focus
- Mono audio
- LED flash for alerts (iOS) / Flash alerts (Android)

Assistive technologies and mobile features

iOS

Please note that iOS 13 features, such as Dark Mode and Voice Control, are not included in this list.

iPhone

- VoiceOver
- Keyboard
- Switch
- Zoom
- Reduce Motion
- Invert colors
- Grayscale
- Larger Text

iPad

- VoiceOver
- Keyboard
- Switch
- Zoom

- Reduce Motion
- Invert colors
- Grayscale
- Larger Text

Android

- TalkBack
- Keyboard
- Switch
- Magnification
- Remove Animations
- Color Inversion
- Grayscale
- Increase font size
- Increase display size

Samsung (optional)

- Voice Assistant

Kindle / Amazon Firestick (optional)

- Voice View

Acknowledgements

Relationship to existing Accessibility testing standards

This document is based on:

- W3C Web Content Accessibility Guidelines, Version 2.0
- W3C Web Content Accessibility Guidelines, Version 2.1
- BBC Mobile Accessibility Guidelines
- AccessibilityOz Mobile Testing Methodology
- TPG Mobile Testing Guide

ICT Accessibility Testing Symposium Native App Sub-Committee

This document was developed by the ICT Accessibility Testing Symposium Native App Sub-Committee. Members include: Gian Wild (Co-Chair), Jennifer Chadwick (Co-Chair), Kathy Eng, Ryan Pugh, Kathryn Weber-Hottleman, Brent Davis, Laura Renfro, Peter McNally, Karen Herr, Steve Sawczyn, Sunish Gupta, Tom Lawton, Sam Bouchat, Rafal Charlampowicz, Damon Wandke, Morgan Lee Kester, Mona Rekhi, Corbb O'Connor and Chris Law.

Contacts

Gian Wild

Web: www.accessibilityoz.com

Phone: 415 621 9366

Email: gian@accessibilityoz.com

Jennifer Chadwick

Web: www. siteimprove.com

Phone: 905 483 9139

Email: jcha@siteimprove.com