UsableNet

WCAG Principles Applied To **Mobile**

Table of Contents

Foreword

The content of this document is for the most part taken from the W3C Working Draft, *"Mobile Accessibility: How WCAG 2.0 and Other W3C/WAI Guidelines Apply to Mobile"*. published in 2015.

At UsableNet, we created this document and added practical visual examples to it, to use it as a quick reference tool for everyone who needs a straightforward introduction or guidance on how WCAG principles can apply to the mobile context, whether is mobile web content, mobile web apps, native apps, or other hybrid solutions.

This document version has been recently updated and includes references to the latest WCAG 2.1 guidelines.

Principle 1

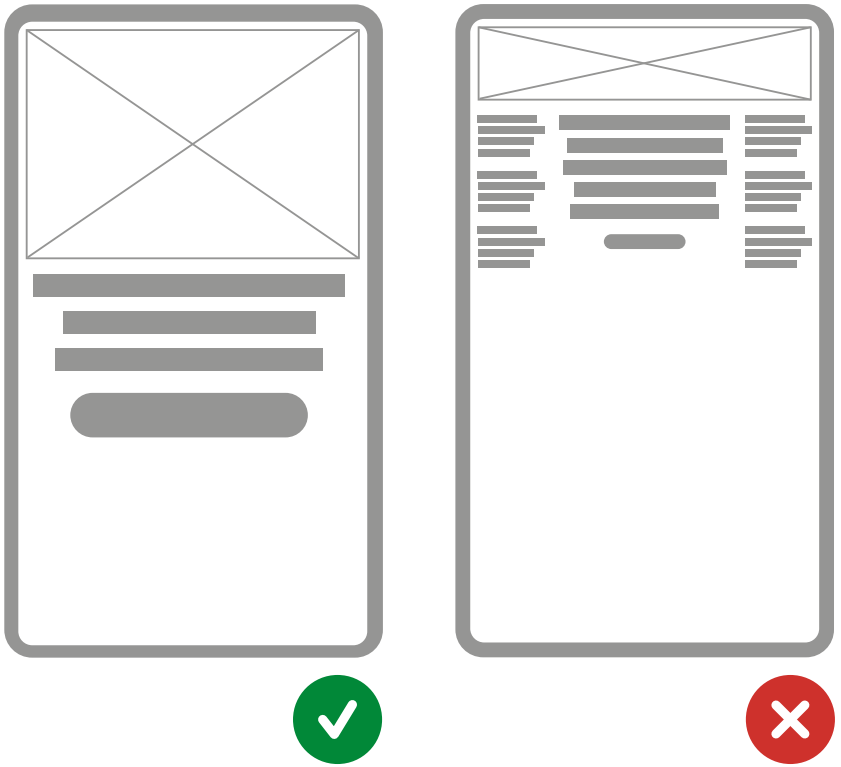
Perceivable

Example 1: Small Screen Size

Small screen size is one of the most common characteristics of mobile devices. While the exceptional resolution of these screens theoretically enables large amounts of information to be rendered, the small size of the screen places practical limits on how much information people can actually view at one time, especially when magnification is used by people with low vision.

Some best practices for helping users to make the most of small screens include

- Minimizing the amount of information that is put on each page compared to desktop/laptop versions by providing a dedicated mobile version or a responsive design:
  - a dedicated mobile version contains content tailored for mobile use. For example, the content may contain fewer content modules, fewer images, or focus on important mobile usage scenarios.
  - a responsive design contains content that stays the same, but CSS stylesheets are used to render it differently depending on the viewport width. For example, on narrow screens the navigation menus may be hidden until the user taps a menu button.
- Providing a reasonable default size for content and touch controls (see "B.2 Touch Target Size and Spacing") to minimize the need to zoom in and out for users with low vision.
- Adapting the length of link text to the viewport width.
- Positioning form fields below, rather than beside, their labels (in portrait layout)



> Approaches such as content prioritization and progressive disclosure favors clarity over density of information, helping the user stay focused throughout the whole app experience.

Example 2: Zoom/Magnification

A variety of methods allow the user to control content size on mobile devices with small screens. At the browser level these methods are generally available to assist a wide audience of users. At the platform level these methods are available as accessibility features to serve people with visual impairments or cognitive disabilities.

The methods include the following:

- OS-level features
  - Set default text size (typically controlled from the Display Settings) *Note:* System text size is often not supported by mobile browsers.
  - Magnify entire screen (typically controlled from the Accessibility Settings). *Note:* Using this setting requires the user to pan vertically and horizontally.
  - Magnifying lens view under user's finger (typically controlled from the Accessibility Settings)
- Browser-level features
  - Set default text size of text rendered in the browser's viewport
    - Reading mode that renders main content at a user-specified text size
  - Magnify browser's viewport (typically "pinch-zoom"). *Note:* Using this setting requires the user to pan vertically and horizontally.
    - *Note:* Some browsers have features that might modify this type of magnification (e.g. re-flowing the content at the new magnification level, overriding author attempts to prevent pinch-zoom).

The WCAG 2.0 success criterion that is most related to zoom/magnification is

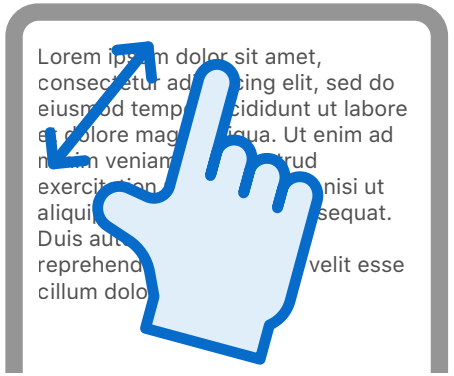
- **1.4.4 Resize text** (Level AA)

SC 1.4.4 requires text to be resizable without assistive technology up to 200 percent. To meet this requirement content must not prevent text magnification by the user.

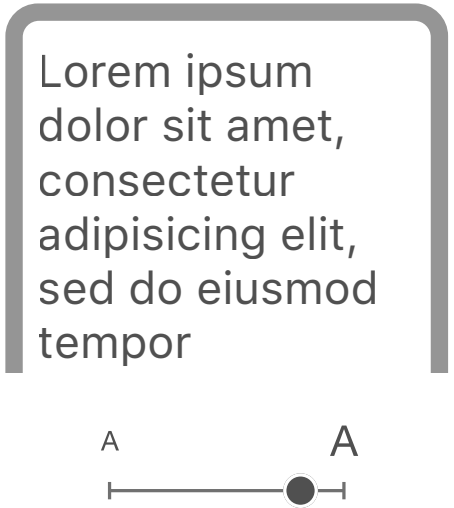
The following methods might be used:

- Ensure that the browser pinch zoom is not blocked by the page's viewport meta element so that it can be used to zoom the page to 200%. Restrictive values for user-scalable and maximum-scale attributes of this meta element should be avoided. *Note:* Relying on full viewport zooming (e.g. not blocking the browser's pinch zoom feature) requires the user to pan horizontally as well as vertically. While this technique meets the success criteria it is less usable than supporting text resizing features that reflow content to the user's chosen viewport size. It is best practice to use techniques that support text resizing without requiring horizontal panning.
- Support for system fonts that follow platform level user preferences for text size.
- Provide on-page controls to change the text size.

Accessibility features geared toward specific populations of people with disabilities fall under the definition of assistive technology adopted by WCAG and thus cannot be relied upon to meet the success criteria. For example, a platform-level zoom feature that magnifies all platform content and has features to specifically support people with low vision is likely considered an assistive technology.



pinch to zoom



OS larger text support

Example 3: Text Contrast

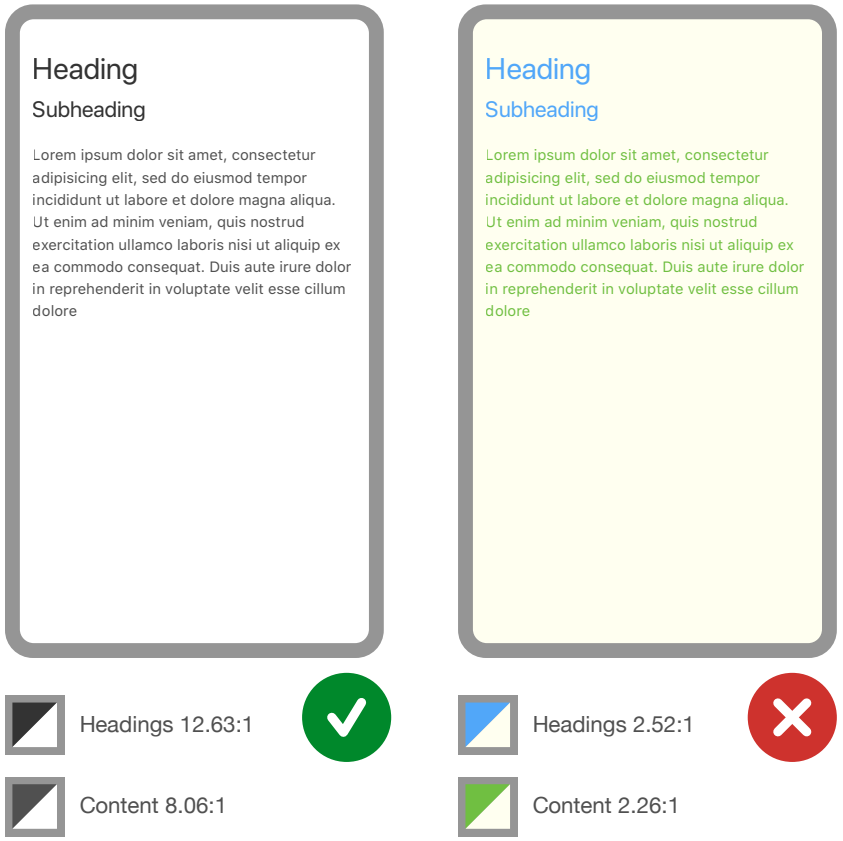
Mobile devices are more likely than desktop/laptop devices to be used in varied environments including outdoors, where glare from the sun or other strong lighting sources is more likely. This scenario heightens the importance of use of good contrast for all users and may compound the challenges that users with low vision have accessing content with poor contrast on mobile devices.

The WCAG 2.0 success criteria related to the issue of contrast are:

- **1.4.3 Contrast (Minimum)** (Level AA) which requires a contrast of at least 4.5:1 (or 3:1 for large-scale text) and
- **1.4.6 Contrast (Enhanced)** (Level AAA) which requires a contrast of at least 7:1 (or 4.5:1 for large-scale text).

SC 1.4.3. allows for different contrast ratios for large text. Allowing different contrast ratios for larger text is useful because larger text with wider character strokes is easier to read at a lower contrast. This allows designers more leeway for contrast of larger text, which is helpful for content such as titles. The ratio of 18-point text or 14-point bold text described in the SC 1.4.3 was judged to be large enough to enable a lower contrast ratio for web pages displayed on a 15-inch monitor at 1024x768 resolution with a 24-inch viewing distance. Mobile device content is viewed on smaller screens and in different conditions so this allowance for lessened contrast on large text must be considered carefully for mobile apps.

For instance, the default point size for mobile platforms might be larger than the default point size used on non-mobile devices. When determining which contrast ratio to follow, developers should strive to make sure to apply the lessened contrast ratio only when text is roughly equivalent to 1.2 times bold or 1.5 times (120% bold or 150%) that of the default platform size. Note, however, that the use of text that is 1.5 times the default on mobile platforms does not imply that the text will be readable by a person with low vision. People with low vision will likely need and use additional platform level accessibility features and assistive technology such as increased text size and zoom features to access mobile content.



Text legibility is preserved by an adequate contrast between the font color and the background. For AA compliance, text should have color ratio of at least 4.5:1 (larger text at least 3:1).

Example 4: Changing Screen Orientation (Portrait/Landscape)

Some mobile applications automatically set the screen to a particular display orientation (landscape or portrait) and expect that users will respond by rotating the mobile device to match. However, some users have their mobile devices mounted in a fixed orientation (e.g. on the arm of a power wheelchair).

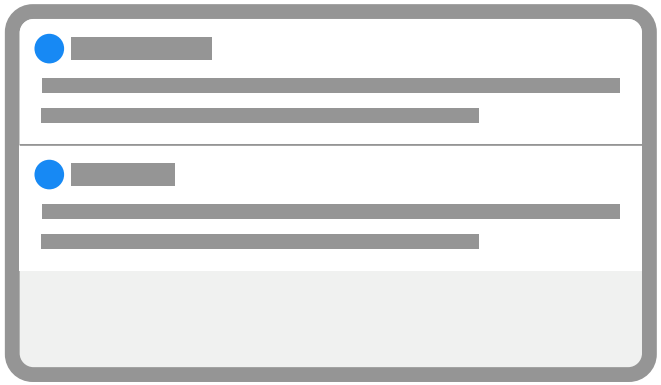
Therefore, mobile application developers should try to support both orientations. If it is not possible to support both orientations, developers should ensure that it is easy for all users to change the orientation to return to a point at which their device orientation is supported.

Changes in orientation must be programmatically exposed to ensure detection by assistive technology such as screen readers. For example, if a screen reader user is unaware that the orientation has changed the user might perform incorrect navigation commands.

The WCAG 2.1 success criterion related to the issue of screen orientation is:

- **1.3.4 Orientation** (Level AA)

SC 1.3.4 requires that content does not restrict its view and operation to a single display orientation, such as portrait or landscape, unless a specific display orientation is essential.



> Mobile apps should never force the user to use a specific screen orientation. Using a device in portrait or landscape mode is a matter of user's preference so app designers and developers should strive at supporting complete functionality regardless of user's behavior.

Example 5: Provide Easy Methods for Data Entry

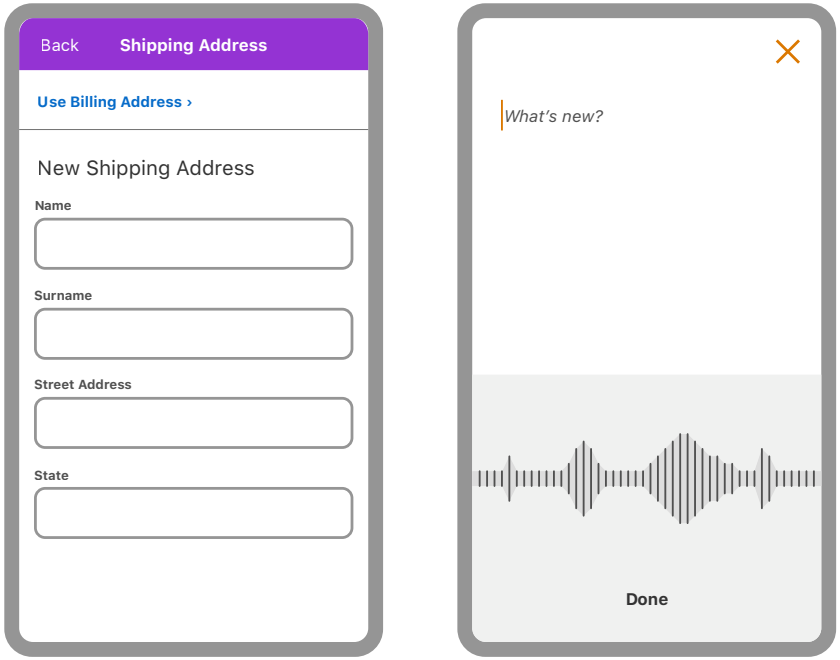
Users can enter information on mobile devices in multiple ways such as on-screen keyboard, Bluetooth keyboard, touch, and speech. Text entry can be time-consuming and difficult in certain circumstances. Reduce the amount of text entry needed by providing select menus, radio buttons, check boxes or by automatically entering known information (e.g. date, time, location).

The WCAG 2.1 success criterion related to this issue is:

- **1.3.5 Identify Input Purpose** (Level AA)

SC 1.3.5 requires that the purpose of each input field collecting information about the user can be programmatically determined when:

- The input field serves a purpose identified in the Input Purposes for User Interface Components section
- The content is implemented using technologies with support for identifying the expected meaning for form input data



> Typing is proven to be a slow method of data entry. Providing alternatives such as autofill, data sharing between apps or dictation improves the overall app experience and prevents errors.

Principle 2

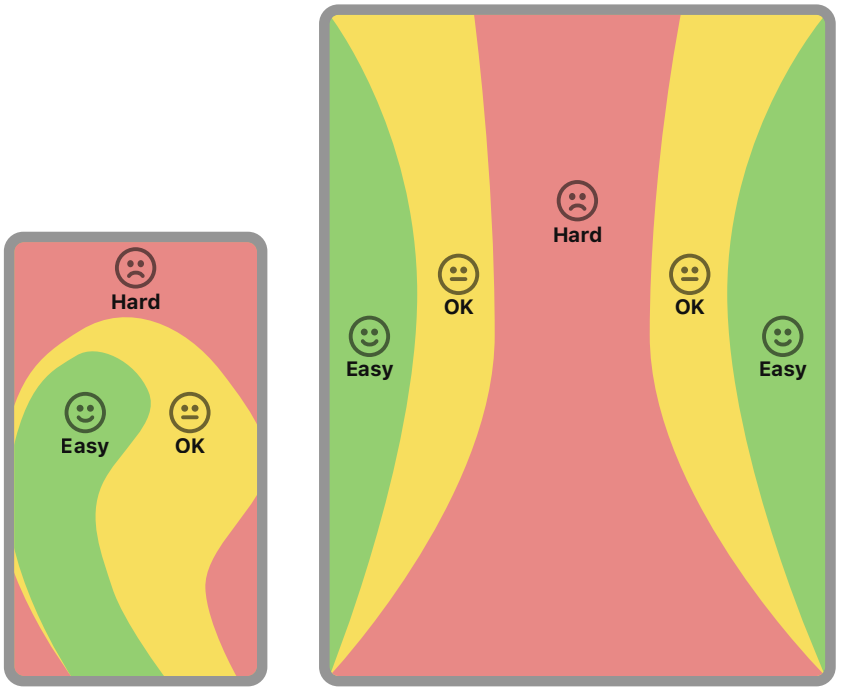
Operable

Example 1: Placing Buttons Where They Are Easy To Access

Mobile sites and applications should position interactive elements where they can be easily reached when the device is held in different positions.

When designing mobile web content and applications many developers attempt to optimize use with one hand. This can benefit people with disabilities who may only have one hand available, however, developers should also consider that an easy-to-use button placement for some users might cause difficulties for others (e.g. left- vs. right-handed use, assumptions about thumb range of motion). Therefore, flexible use should always be the goal.

Some (but not all) mobile operating systems provide work-around features that let the user temporarily shift the display downwards or sideways to facilitate one-handed operation.



Different screen sizes require different handholds. Between phones and tablets the regions of the screen we can comfortably reach with our thumbs change enormously. The green thumb zone identifies the areas where our thumbs can perform accurate tapping. The red zones instead are those most difficult to reach, so touch targets placed within these areas should be definitely larger than usual.

Example 2: Keyboard Control for Touchscreen Devices

Mobile device design has evolved away from built-in physical keyboards (e.g. fixed, slide-out) towards devices that maximize touchscreen area and display an on-screen keyboard only when the user has selected a user interface control that accepts text input (e.g. a textbox).

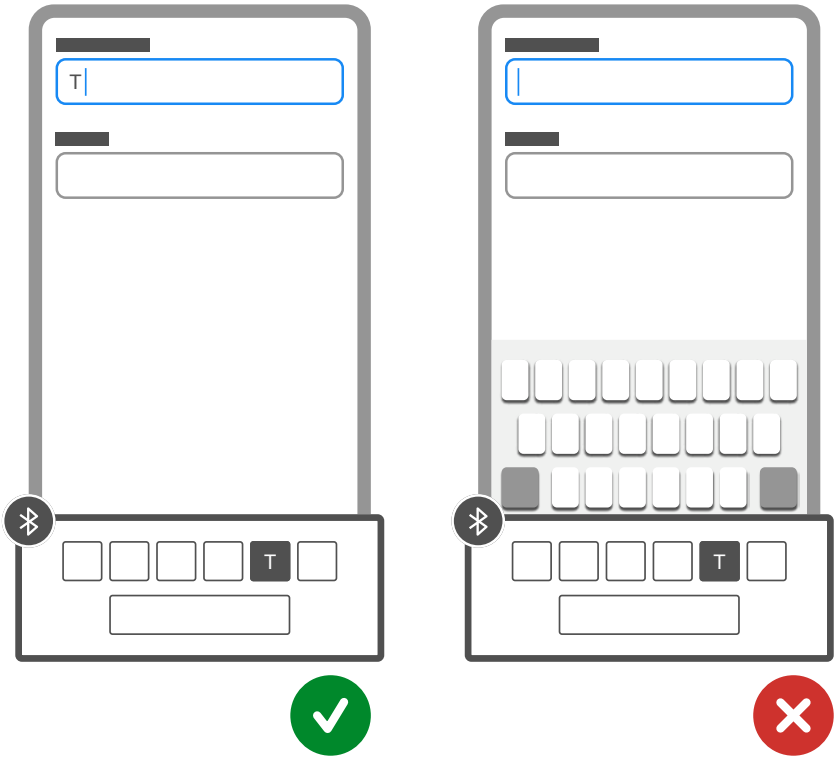
However, keyboard accessibility remains as important as ever and most major mobile operating systems do include keyboard interfaces, allowing mobile devices to be operated by external physical keyboards (e.g. keyboards connected via Bluetooth, USB On-The-Go) or alternative on-screen keyboards (e.g. scanning on-screen keyboards).

Supporting these keyboard interfaces benefits several groups with disabilities:

- People with visual disabilities who can benefit from some characteristics of physical keyboards over touchscreen keyboards (e.g. clearly separated keys, key nibs and more predictable key layouts).
- People with dexterity or mobility disabilities, who can benefit from keyboards optimized to minimize inadvertent presses (e.g. differently shaped, spaced and guarded keys) or from specialized input methods that emulate keyboard input.
- People who can be confused by the dynamic nature of onscreen keyboards and who can benefit from the consistency of a physical keyboard.

Several WCAG 2.0 and 2.1 success criteria are relevant to effective keyboard control:

- **2.1.1 Keyboard** (Level A)
- **2.1.2 No Keyboard Trap** (Level A)
- **2.1.4 Character Key Shortcuts** (Level A)
- **2.4.3 Focus Order** (Level A)
- **2.4.7 Focus Visible** (Level AA)
- **2.5.6 Concurrent Input Mechanisms** (Level AAA)



Some mobile app experiences may sometimes end up being less engaging (if not frustrating) for users with disabilities, who rely on an external physical keyboard for interactive operations.

Example 3: Touch Target Size and Spacing

The high resolution of mobile devices means that many interactive elements can be shown together on a small screen. But these elements must be big enough and have enough distance from each other so that users can safely target them by touch.

Best practices for touch target size include the following:

- Ensuring that touch targets are at least 9 mm high by 9 mm wide.
- Ensuring that touch targets close to the minimum size are surrounded by a small amount of inactive space.

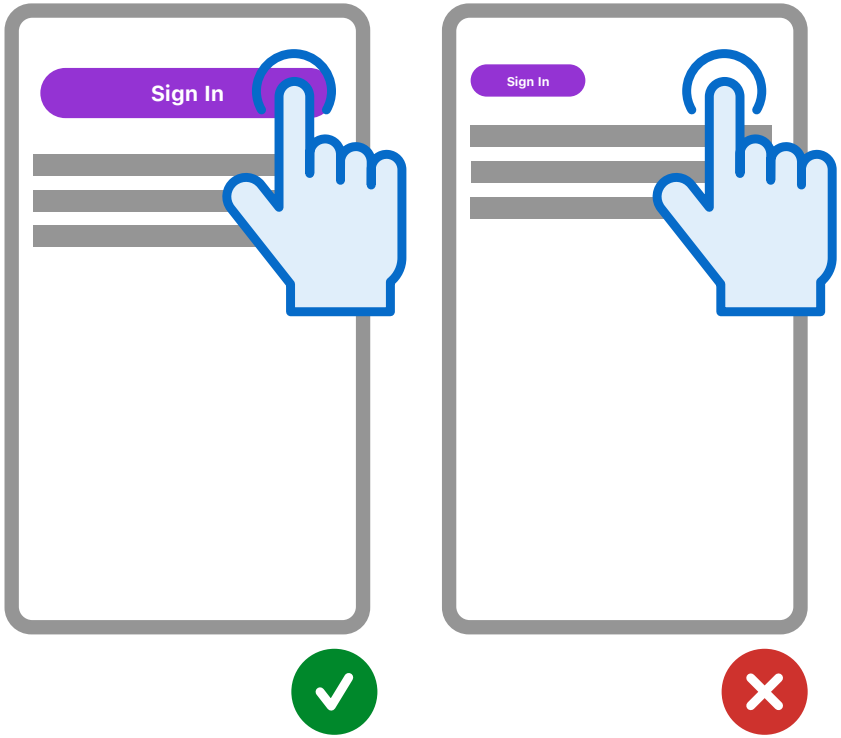
Note: This size is not dependent on the screen size, device or resolution. Screen magnification should not need to be used to obtain this size, because magnifying the screen often introduces the need to pan horizontally as well as vertically, which can decrease usability.

The WCAG 2.1 success criterion related to the issue of touch target size and spacing is:

- **2.5.5 Target Size** (Level AA)

SC 2.5.5 requires that the size of the target for pointer inputs is at least 44 by 44 CSS pixels except when

- **Equivalent**: the target is available through an equivalent link or control on the same page that is at least 44 by 44 CSS pixels
- **Inline**: the target is in a sentence or block of text
- **User Agent Control**: the size of the target is determined by the user agent and is not modified by the author
- **Essential**: a particular presentation of the target is essential to the information being conveyed



> Since human fingers are a very imprecise pointing tool, tap targets within an app should be big enough to help people interact with precision and confidence, even when they have to perform tasks in a hurry.

Example 4: Touchscreen Gestures

Many mobile devices are designed to be primarily operated via gestures made on a touchscreen. These gestures can be simple, such as a tap with one finger, or very complex, involving multiple fingers, multiple taps and drawn shapes.

Some (but not all) mobile operating systems provide work-around features that let the user simulate complex gestures with simpler ones using an onscreen menu.

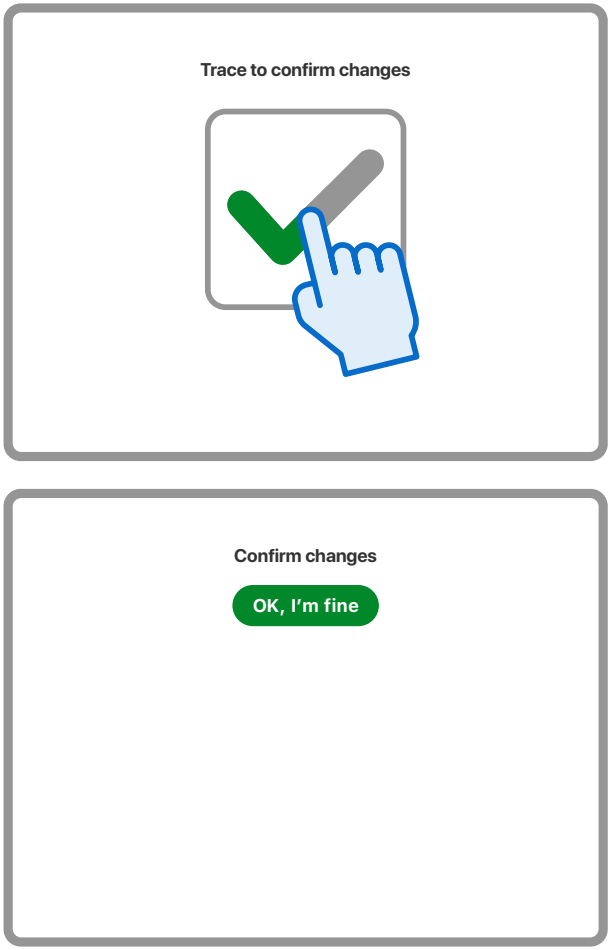
Some best practices when deciding on touchscreen gestures include the following:

- Gestures in apps should be as easy as possible to carry out. This is especially important for screen reader interaction modes that replace direct touch manipulation by a two-step process of focusing and activating elements. It is also a challenge for users with motor or dexterity impairments or people who rely on head pointers or a stylus where multi-touch gestures may be difficult or impossible to perform. Often, interface designers have different options for how to implement an action. Widgets requiring complex gestures can be difficult or impossible to use for screen reader users. Usually, design alternatives exist to allow changes to settings via simple tap or swipe gestures.
- Activating elements via the mouseup or touchend event. Using the mouseup or touchend event to trigger actions helps prevent unintentional actions during touch and mouse interaction. Mouse users clicking on actionable elements (links, buttons, submit inputs) should have the opportunity to move the cursor outside the element to prevent the event from being triggered. This allows users to change their minds without being forced to commit to an action. In the same way, elements accessed via touch interaction should generally trigger an event (e.g. navigation, submits) only when the touchend event is fired (i.e. when all of the following are true: the user has lifted the finger off the screen, the last position of the finger is inside the actionable element, and the last position of the finger equals the position at touchstart).

Another issue with touchscreen gestures is that they might lack onscreen indicators that remind people how and when to use them. For example, a swipe in from the left side of the screen gesture to open a menu is not discoverable without an indicator or advisement of the gesture.

The WCAG 2.1 success criterion related to this issue is:

- **2.5.1 Pointer Gestures** (Level A)



Customized gestures are sometimes used as an effective replacement for annoying confirmation dialogs. However, users with disabilities may still benefit from the latter, so app designers should implement them as an alternative feature.

Example 5: Device Manipulation Gestures

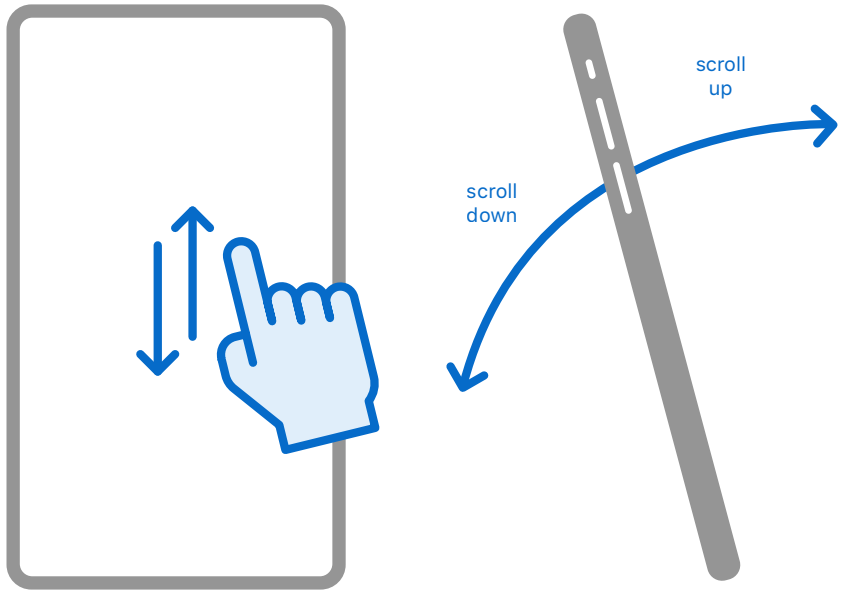
In addition to touchscreen gestures, many mobile operating systems provide developers with control options that are triggered by physically manipulating the device (e.g. shaking or tilting). While device manipulation gestures can help developers create innovative user interfaces, they can also be a challenge for people who have difficulty holding or are unable to hold a mobile device.

Some (but not all) mobile operating systems provide work-around features that let the user simulate device shakes, tilts, etc. from an onscreen menu.

Therefore, even when device manipulation gestures are provided, developers should still provide touch and keyboard operable alternative control options.

- **2.1.1 Keyboard** (Level A)
- **2.5.4 Motion Actuation** (Level A)

Another issue with control via device manipulation gestures is that they might lack onscreen indicators that remind people how and when to use them.



> Accelerometers allow designers and developers to explore innovative and unique experiences for the users. Those kinds of features should always be treated as enhancements of the standard control options, which keep the app totally usable and accessible.

Example 6: Provide Mechanisms To Abort or Undo the Action

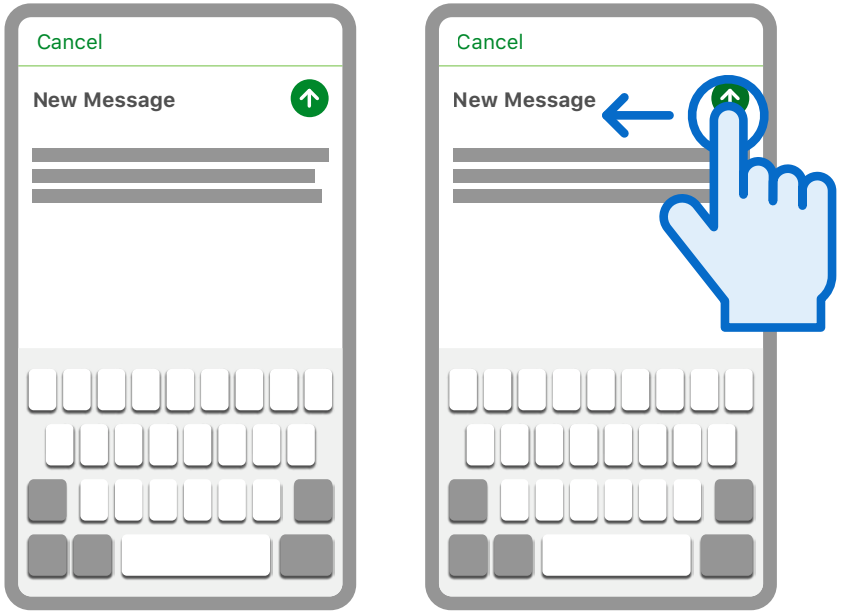
The WCAG 2.1 success criterion related to the issue is:

- **2.5.2 Pointer Cancellation** (Level A)

SC 2.5.2 requires that for functionality that can be operated using a single pointer, at least one of the following is true.

- **No Down-Event**: the down-event of the pointer is not used to execute any part of the function;
- **Abort or Undo**: completion of the function is on the up-event, and a mechanism is available to abort the function before completion or to undo the function after completion;
- **Up Reversal**: the up-event reverses any outcome of the preceding down-event;
- **Essential**: completing the function on the down-event is essential.

This success criterion provides guidelines about how actions should be properly designed and implemented. Actions should usually happen on the up event, using the generic OS built-in event for that prupose. Down events should only be only when the behavior is considered essential (e.g., a piano player simulator).



> Not only users with cognitive disabilities, but also people with tremor, mobility impairments or someone in a hurry may touch or click on the wrong screen location by mistake, experiencing an unexpected bahavior of the application.
>
> Users should be able to abort or undo an unintended action, like the ability of sliding the finger away from a button, before lifting it without invoking any action.

Principle 3

Understandable

Example 1: Consistent Layout

Components that are repeated across multiple pages should be presented in a consistent layout. In responsive web design, where components are arranged based on device size and screen orientation, web pages within a particular view (set size and orientation) should be consistent in placement of repeated components and navigational components. Consistency between the different screen sizes and screen orientations is not a requirement under WCAG 2.0.

For example:

- A Web site has a logo, a title, a search form and a navigation bar at the top of each page; these appear in the same relative order on each page where they are repeated. On one page the search form is missing but the other items are still in the same order. When the Web site is viewed on a small screen in portrait mode, the navigation bar is collapsed into a single icon but entries in the drop-down list that appears when activating the icon are still in the same relative order.
- A Web site, when viewed on the different screen sizes and in different orientations, has some components that are hidden or appear in a different order. The components that show, however, remain consistent for any screen size and orientation.

The WCAG 2.0 success criteria that are most related to the issue of consistency are:

- **3.2.3 Consistent Navigation** (Level AA)
- **3.2.4 Consistent Identification** (Level AA)

Consistency is key to creating seamless and cross-channel user experiences. It helps the user feeling comfortable and in control while starting a task, which may start on mobile and get completed on tablet or desktop.

Example 2: Positioning Important Page Elements Before the Page Scroll

The small screen size on many mobile devices limits the amount of content that can be displayed without scrolling.

Positioning important page information so it is visible without requiring scrolling can assist users with low vision and users with cognitive impairments.

If a user with low vision has the screen magnified only a small portion of the page might be viewable at a given time. Placing important elements before the page scroll allows those who use screen magnifiers to locate important information without having to scroll the view to move the magnified area. Placing important elements before the page scroll also makes it possible to locate content without performing an interaction. This assists users that have cognitive impairments such as short-term memory disabilities. Placing important elements before the page scroll also helps ensure that elements are placed in a consistent location. Consistent and predictable location of elements assists people with cognitive impairments and low vision.

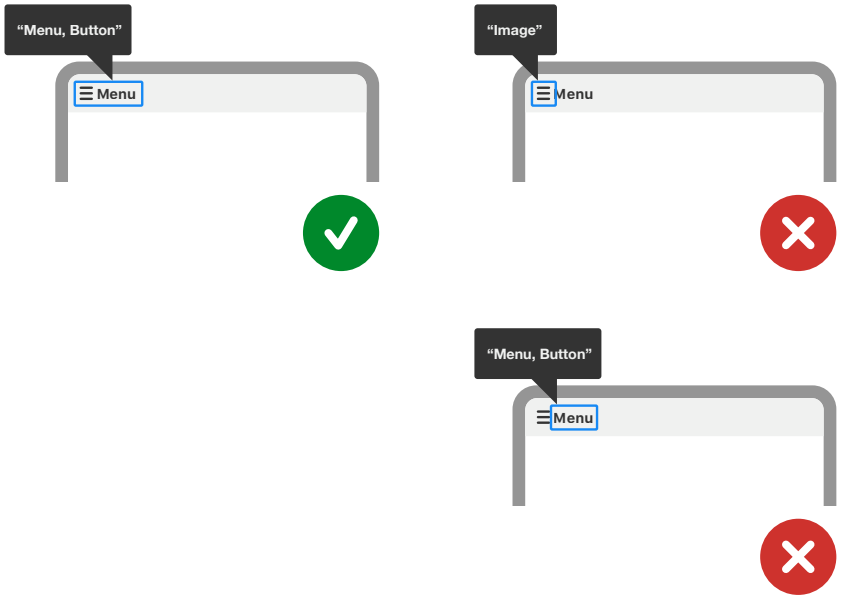
Properly prioritized layout design helps users focus on the most important content displayed on screen and makes the tasks faster to perform.

Example 3: Grouping Operable Elements That Perform the Same Action

When multiple elements perform the same action or go to the same destination (e.g. link icon with link text), these should be contained within the same actionable element. This increases the touch target size for all users and benefits people with dexterity impairments. It also reduces the number of redundant focus targets, which benefits people using screen readers and keyboard/ switch control.

The WCAG 2.0 success criteria that are most related to grouping of actionable elements are:

- **2.4.4 Link Purpose (In Context)** (Level A)
- **2.4.9 Link Purpose (Link Only)** (Level AA)



> Properly grouped elements keep the navigation order logical for keyboard users and allow screen reading technologies to properly describe focused targets.

Example 4: Provide Clear Indication That Elements Are Actionable

Elements that trigger changes should be sufficiently distinct to be clearly distinguishable from non-actionable elements (content, status information, etc). Providing a clear indication that elements are actionable is relevant for web and native mobile applications that have actionable elements like buttons or links, especially in interaction modes where actionable elements are commonly detected visually (touch and mouse use). Interactive elements must also be detectable by users who rely on a programmatically determined accessible name (e.g. screen reader users).

Visual users who interact with content using touch or visual cursors (e.g. mice, touchpads, joysticks) should be able to clearly distinguish actionable elements such as links or buttons. Existing interface design conventions are aimed at indicating that these visual elements are actionable. The principle of redundant coding ensures that elements are indicated as actionable by more than one distinguishing visual feature. Following these conventions benefits all users, but especially users with vision impairments.

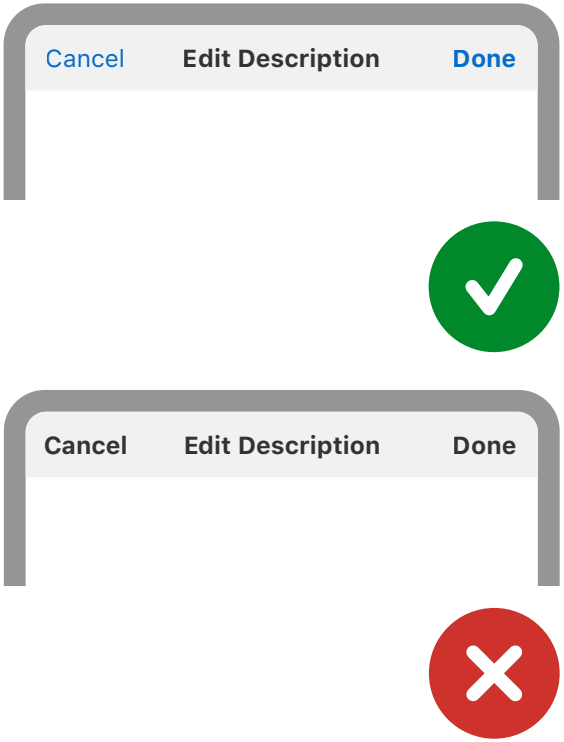
Visual features that can set an actionable element apart include shape, color, style, positioning, text label for an action, and conventional iconography.

Examples of distinguishing features:

1. Conventional shape: Button shape (rounded corners, drop shadows), checkbox, select rectangle with arrow pointing downwards
2. Iconography: conventional visual icons (question mark, home icon, burger icon for menu, floppy disk for save, back arrow, etc)
3. Color offset: shape with different background color to distinguish the element from the page background, different text color
4. Conventional style: Underlined text for links, color for links
5. Conventional positioning: Commonly used position such as a top left position for back button (iOS), position of menu items within left-aligned lists in drop-down menus for navigation

The WCAG 2.0 and 2.1 success criteria do not directly address issue of clear visual indication that elements are actionable but are related to the following success criteria:

- **3.2.3 Consistent Navigation** (Level AA)
- **3.2.4 Consistent Identification** (Level AA)



Actionable elements and controls should be clearly distinguishable, properly styled and labelled in order to make the interaction feel easy and straightforward. Considering specific OS interface design guidelines is key for delivering consistent user experiences.

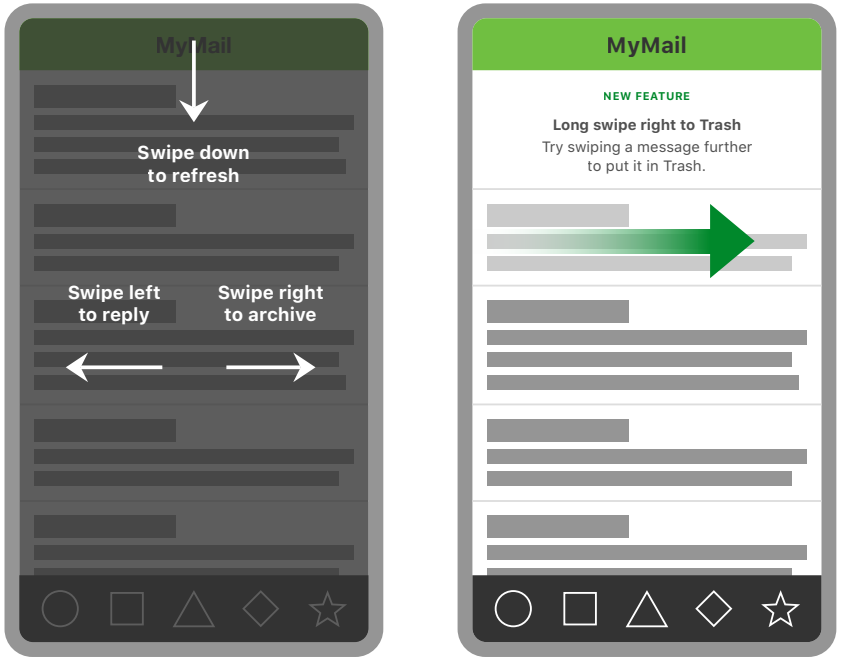
Example 5: Provide Instructions for Custom Touchscreen and Device Manipulation Gestures

The ability to provide control via custom touchscreen and device manipulation gestures can help developers create efficient new interfaces. However, for many people, custom gestures can be a challenge to discover, perform and remember.

Therefore, instructions (e.g. overlays, tooltips, tutorials, etc.) should be provided to explain what gestures can be used to control a given interface and whether there are alternatives. To be effective, the instructions should, themselves, be easily discoverable and accessible. The instructions should also be available anytime the user needs them, not just on first use, though on first use they may be made more apparent through highlighting or some other mechanism.

These WCAG 2.0 success criteria are relevant to providing instructions for gestures:

- **3.3.2 Labels or Instructions** (Level A)
- **3.3.5 Help** (Level AAA)



Static screen instructions are the most common type of tutorials adopted by many mobile apps. The “learn by playing” technique, largely adopted in video game design, is however a much more effective way of coaching: by pairing instructions with action, the user learns new gestures through active participation.

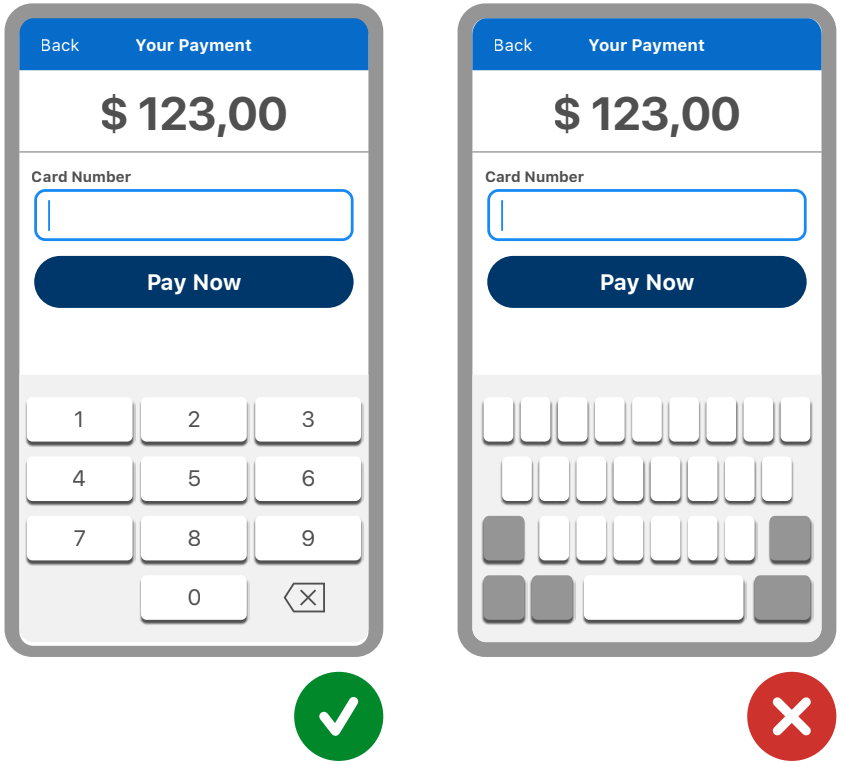
Principle 4

Robust

Example 1: Set the Virtual Keyboard To the Type of Data Entry Required

On some mobile devices, the standard keyboard can be customized in the device settings and additional custom keyboards can be installed. Some mobile devices also provide different virtual keyboards depending on the type of data entry. This can be set by the user or can be set to a specific keyboard.

For example, using the different HTML5 form field controls (see Method Editor API) on a website will show different keyboards automatically when users are entering in information into that field. Setting the type of keyboard helps prevent errors and ensures formats are correct but can be confusing for people who are using a screen reader when there are subtle changes in the keyboard.



> Providing users with the appropriate keyboard for the type of data they have to enter plays an important role in terms of simplifying filling forms and make typing faster.

Example 2: Support the Characteristic Properties of the Platform

Mobile devices provide many features to help users with disabilities interact with content. These include platform characteristics such as zoom, larger fonts, and captions. The features and functions available differ depending on the device and operating system version.
For example, most platforms have the ability to set large fonts, but not all applications honor it for all text. Also, some applications might increase font size but not wrap text, causing horizontal scrolling.

iOS Android

Both iOS and Android provide accessibility features for zooming and magnifying screen content. On iOS the three finger double tap gesture activates the zoom feature, whereas on Android the same feature gets triggered with just one finger double tap.

References

Webpages

- *Mobile Accessibility: How WCAG 2.0 and Other W3C/WAI Guidelines Apply to Mobile,* W3C First Public Working Draft 26 February 2015, Copyright © 2015 World Wide Web Consortium, (MIT, ERCIM, Keio, Beihang), https://www.w3.org/TR/mobile-accessibility-mapping/
- *Web Content Accessibility Guidelines (WCAG) 2.0,* https://www.w3.org/TR/WCAG20/
- *Web Content Accessibility Guidelines (WCAG) 2.1,* https://www.w3.org/TR/WCAG21/
- *How Do Users Really Hold Mobile Devices?,* Steven Hoober, http://www.uxmatters.com/mt/archives/2013/02/how-do-users-really-hold-mobile-devices.php
- *iOS Human Interface Guidelines,* https://developer.apple.com/ios/human-interface-guidelines/overview/design-principles/

Books

- *Designing for Touch*, Josh Clark, (A Book Apart, 2015)
- *Mobile First,* Luke Wroblewski, (A Book Apart, 2011)