Enhancing accessibility of an Android app

Mentorship System

by Bartek Pacia Google Code-in 2019

task link

This document contains hyperlinks – it is intended to be read on an electronic device with Internet access. Do not print.

Overview

Accessibility is an important part of every app, though often overlooked. I can think of several possible reasons for this, like a lack of awareness among the developer community, a false conviction that it's unnecessary and nobody will ever use it, not enough time or even laziness. All of this creates an image of accessibility as a complex, difficult topic that most Android developers would prefer not to mess around with. The multitude of human disabilities does not make this task easier either.
Okay, contemplations aside, let's say that we consider ourselves a professional programmer and have decided to take a step in the right direction and make our app *accessible*. What does it mean? Is there a tool that can give us, developers, a helping hand?
To answer the first question, I have come up with this definition:

Accessibility – feature of a software that enables everyone to use it comfortably, no matter whether the user is visually-impaired, deaf, has color blindness or is able-bodied.

To answer the second one, let me introduce you to

Android Accessibility Services

Many programmers are not aware that the Android Framework provides an extensive set of features for users with disabilities. Utilities like **TalkBack**[1], **Select to Speak**[2] or **Magnification**[3] are able to greatly improve the experience of impaired users. However, they are not almighty and, not counting the simplest use cases, it is

[1] TalkBack (a.k.a Android Accessibility Suite) – screen reader included on all Android devices. It gives users spoken feedback so that they can use the device without looking at the screen.
[2] Select to Speak – built-in accessibility feature of Android that enables users to select any text (such as emails and messages) and have it read aloud using the default screen reader.
[3] Magnification – allows visually impaired users to zoom in and pan the whole screen to see it better.

the app developers' responsibility to make the most out of those tools in order to improve the UX of impaired users.
Fortunately, it is not as hard as it may initially seem. Oftentimes, making a typical[4] app accessible comes down to following a few simple best practices. Let's take a look at them.

Best practices for accessibility

- Use contrasting colors
  This is probably the simplest accessibility rule, yet the one which makes the biggest difference, at least for the blind and visually-impaired users. The rule of thumb is that the color contrast ratio for texts (in TextViews, Buttons and everywhere else) should be at least 3:1.

figure 1.1. Example of a TextView with too low contrast.

figure 1.2 Example of a TextView with sufficient contrast.

- Make views large enough
  Thanks to the rapid development of smartphones, developers often try to make use of those beautiful Quad HD displays and stuff as much UI elements as possible on a single screen. It *may* bring stunning results, but what it always brings is the deterioration of readability. In such cases, even able-bodied people have problems tapping a small icon in the edge of the screen – now think how big challenge it becomes for person with, for example, significal farsightedness.
  Android Docs recommends every clickable button to be at least 48dp x 48d.

- Provide descriptive labels when needed
  When the user asks TalkBack to describe what a particular UI element means, it attempts to pull as much information as possible from. If that element is a simple TextView, it simply reads the contents of it. But what should TalkBack do when it cannot infer any useful information about the element – for example, when the user asks about the contents of an

[4] By "typical" is meant an app which consists mainly of the default UI components provided by the Android Framework and following Material Design guidelines.

ImageButton? After all, it's contents are a few quadrillions of 0s and 1s – not very self-descriptive.
For such situations, a special XML attribute exists – it is called **android:contentDescription**. We pass it a string (or, preferably, a string resource) and it serves as a source of information for screen-reading services like TalkBack.
It is important to keep those **contentDescription**s as short and as informative as possible.

- Do not depend *only* on colors to convey information
  To anyone who has been using computers for quite a while (so, basically everyone nowadays), it is obvious to everyone that green means acceptance to something and red means rejection. That is why depending on colors to convey meaning seems like a natural choice. But what if the user has color blindness and both green and red look the same? This ambiguity is going to make his experience horrible.
  This can be fixed by providing more information besides color, like text or icons.

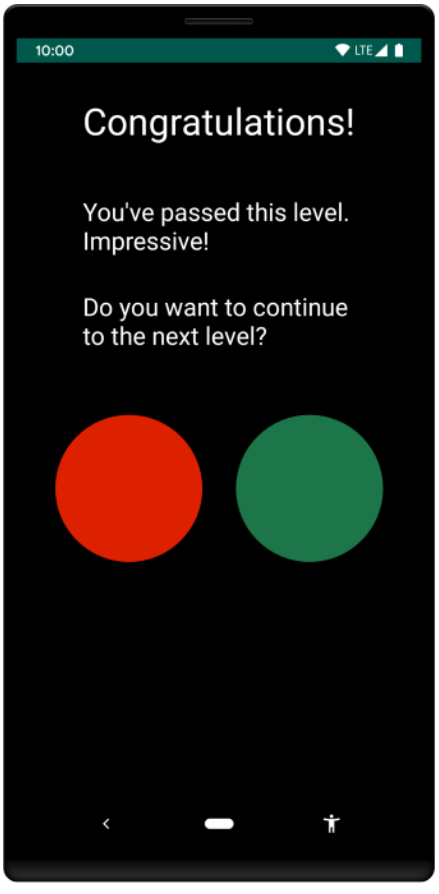

figure 1.4. Bad – depending only on color to convey meaning.

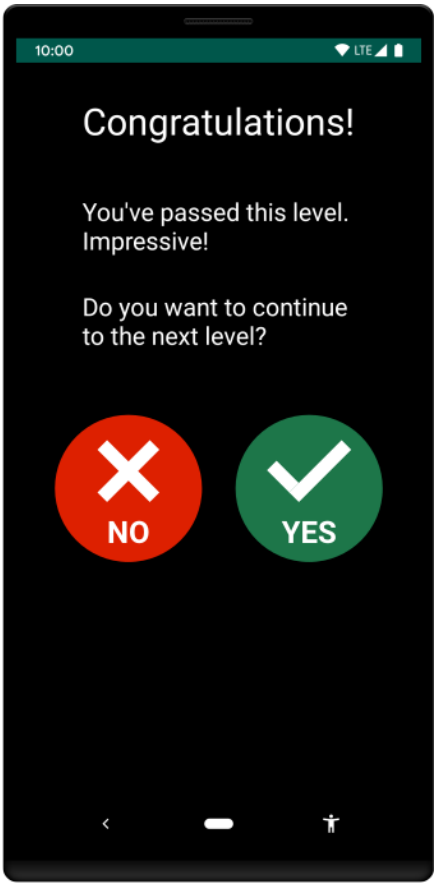

figure 1.5. Good – information is conveyed by color, text and icon.

Accessibility in Mentorship Android

Now I would like to present the status of following accessibility best practices in Mentorship Android app.

While researching about ways to test accessibility, I quickly stumbled upon Accessibility Scanner application. It was made by Google specifically to help with verifying places where app's accessibility could be improved.

I followed the setup process (which is a bit more complex than usual because it requires special permissions) and used the app to verify all Activities and Fragments found in Mentorship Android. I have to say I am positively surprised – usually there was only 1 *color contrast* or *element too small* warning per screen. The one single Activity where a lot of things could be improved is SignUpActivity. You can see screenshots of its scan below.

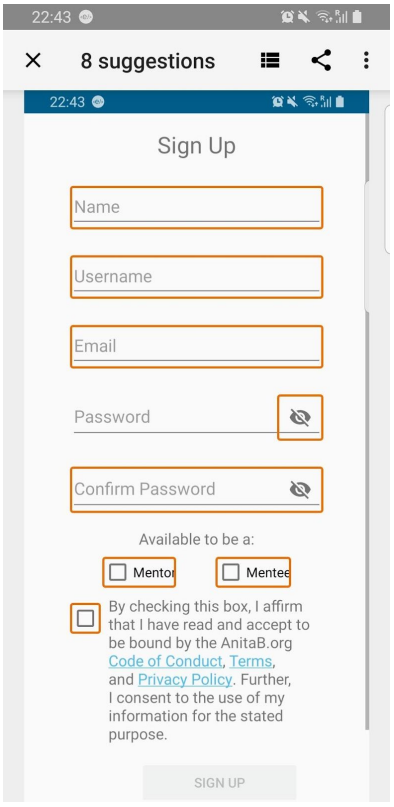

figure 1.5. SignUpActivity as linted by Accessibility Scanner

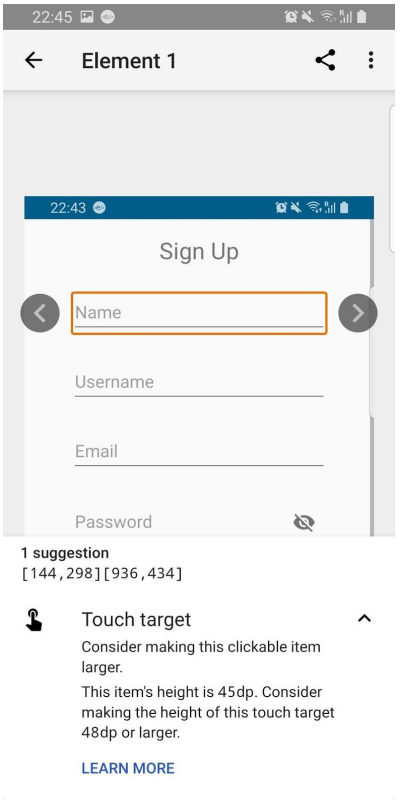

figure 1.6 Warning about element being too small.

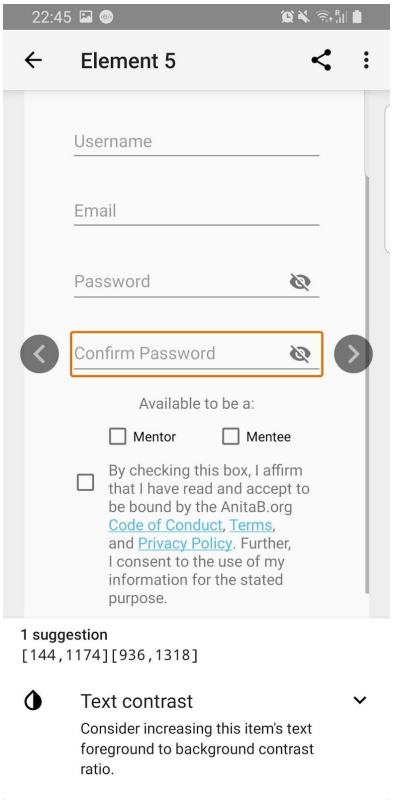

figure 1.7 Warning about element's color contrast being too low.

As you can see, suggestions provided by Accessibility Scanner are incredibly useful and of high quality.

Of course, even the best tools can't replace the opinion of a real, disabled user. If you can't do it, there's an interesting substitute – enable all accessibility settings on your physical Android device, put something (what about a scarf?) on your eyes and...try to use the app.

Being unable to normally use your very own app is a very interesting experience – a bit shocking, disturbing, and unsettling, but certainly – extremely eye-opening.
I'm not sure whether this fits here, but I thought that...if I were Google, I would go one step further and required every developer who wants to publish an app on the Play Store to pass a "Bird Box test". To pass it, the developer would have to complete a certain action in their own app below the time limit - and with eyes closed, using TalkBack as the only way to communicate with the device. I'm sure that it would increase adoption of accessibility best practices by tens of percent.

Wrapping up

Accessibility turned out not to be as hard and complex as I was initially thinking it is. I had no idea how much it can help the disabled and how little effort it required to achieve very satisfying results.
I have to admit that I feel terribly bad about ignoring all those missing attribute android:contentDescription Lint warnings in my personal projects. I will fix them as soon as I have some free time.
Anyway, I hope this document will help people who will work on the Mentorship System to make it even more inclusive – by incorporating best practices for accessibility.

Resources used

Adding Accessibility Features to Apps for Blind and Visually-Impaired Users
Making Android Accessibility Easy (Android Dev Summit 2018)
Build more accessible apps