An Analysis of Five Different Native and Web-Hybrid Platforms for Building Android Apps and their Accessibility for Screen Readers

Lucas Pedroso Carvalho
Depto. de Ciência da Computação
Universidade Federal de Lavras
Lavras, MG, Brazil
lucaspedrosocarvalho@gmail.com

Felipe Silva Dias
Depto. de Ciência da Computação
Universidade Federal de Lavras
Lavras, MG, Brazil
fsdias.v@gmail.com

André Pimenta Freire
Depto. de Ciência da Computação
Universidade Federal de Lavras
Lavras, MG, Brazil
apfreire@dcc.ufla.br

***Abstract*—The choice of an interface platform to develop mobile applications may have important implications to how accessible the resulting product can be for visually-disabled users. This paper aimed to analyze four platforms to develop native and web-hybrid mobile Android applications, and to verify the adequacy of their interface components to implement mobile applications, in order to identify the main accessibility problems that could be encountered by developers when using them, and the main strategies to overcome those issues. We built 5 prototypes of mobile applications with the aim of adhering as much as possible to accessibility recommendations. The applications were built using techniques of native applications developed with Android Studio with and without Web components and hybrid development using the frameworks Apache Cordova, Ionic and Appcelerator Titanium. We then performed an accessibility inspection of a sample of 30 Android interface components present in 5 prototypes of mobile applications, to verify their adequacy for working with screen readers. The results showed that the prototypes developed using web components were more compatible with accessibility criteria in the Web Content Accessibility Guidelines (WCAG 2.0) and with the screen reader TalkBack. The most frequent accessibility problems in native components occurred in tables, headings and multimedia elements. We conclude by showing initial evidence that web-based components in hybrid applications developed using web-hybrid and native with embedded web components currently have better support for accessibility than applications with only native components.**

***Index Terms*—Accessibility, Mobile Applications, Screen Readers**

I. Introduction

Enabling disabled people to effectively participating in society by means of digital resources is fundamental to their lives. This includes providing access to services, products and information in order for them to be more independent and confident to have their rights preserved and to be active citizens.

According to Part 171 of the ISO 9241 standard [16], the accessibility of interactive systems can be defined as the "usability of a product, service, environment, or facility by people with the widest range of abilities".

We thank CNPq for the financial support to this research project

Different guidelines provide developers with resources to develop more accessible applications. The Web Accessibility Initiative (WAI) from the World Wide Web Consortium (W3C) proposed the Web Content Accessibility Guidelines (WCAG) [5], currently in its second version. Despite not yet having specific recommendations for mobile applications, the model was conceived with the aim to try to be independent from specific technologies (such as was the case with its first version, strongly tied to web technologies). WCAG has been used as the main reference in terms of accessibility by several countries, and has also been the basis for other national standards, such as the Brazilian e-MAG (the Accessibility Model for e-Government) [3].

With the growth and the increasing need to use mobile devices to access the Internet, the inclusion of disabled people becomes an important goal to be achieved. According the the market research company Statista [25], the number of smartphone users worldwide was expected to grow from 1.5 billion users in 2014 to 2.5 billion users in 2019.

The development of mobile applications should take into account the way in which different disabilities can impact on different ways of interaction. Designers and developers should be aware of those needs when considering developing approaches and platforms to choose for their systems.

Different approaches have been used in the construction of mobile applications. In this paper, in particular, we have investigated different platforms for development based on three main approaches for building mobile applications: hybrid development (using web technologies as basis for building native mobile apps), pure native apps (without the use of any web-based interface components) and mostly-native development, with some occasional use of embedded web-based interface components.

Given the different techniques to build mobile applications, there is still limited information for developers and designers about the resources provided by them to implement accessible interactions for disabled users. In that sense, members from the international community have endeavoured many efforts to make changes to WCAG 2.0 and its success criteria to

make it possible to apply them in the mobile web context. Further to this, there are technical recommendations specific to developers of mobile platforms, such as iOS and Android. However, little is known about the extent to which the resources developed as hybrid, native or native with occasional embedded components can help developers deliver accessible resources that can be used by assistive technologies.

In a previous study performed by the authors of the present paper [8], we investigated the adequacy of three different types of interface components implemented using purely native Android components, hybrid applications implemented with Apache Cordova, and applications with occasional embedded web components into a mostly native application. The results obtained in that study showed that mobile web-hybrid applications and native apps with embedded web components provided screen-reader users with better support in providing accessibility features to users. On the other hand, many native Android interface components had shortcomings that could seriously hamper blind users while performing their tasks, such as the lack of structural elements to guide navigation (in tables and headers, for instance).

However, despite the initial implications of the findings, the previous work was still limited in the sense that only one platform for building web-hybrid applications was analysed.

Thus, the goal of the work presented in the present paper was to carry out a new set of analyses expanding on the previous analysis of platforms for building mobile applications regarding their accessibility. The current paper aims at expanding the understanding of the interplay between the choices of interface components and platforms and their consequences in terms of the ability to enable developers to build accessible applications for users with visual disabilities. Thus, this paper set out to analyse the adequacy of five prototypes of mobile applications, built with the aim of adhering as much as possible to accessibility recommendations, implemented using: Android Studio 1) with and 2) without Web components and hybrid development using the frameworks 3) Apache Cordova, 4) Ionic and 5) Appcelerator Titanium.

The remainder of this paper is organised as follows. Section II presents the main concepts related to the development of mobile applications. Section III presents a literature review regarding studies related to the accessibility of mobile applications. Section IV presents the method adopted in this study. Section V lists the results and, finally, Section VI discusses the conclusions and proposed future work.

II. Background

The development of apps for mobile devices has grown considerably, as well as the popularization of such apps. According to the most recent report from the Android consultancy AppBrain [1], 686,888 new apps were made available on Google's PlayStore in 2016, totaling an accumulated number of 2,623,790 apps available in that year. Following the growth in availability of apps for the Android platform, the use of this operating system in Brazil has also grown. Android had a share of 82.15% of the Brazilian market in 2016 [26].

Different approaches and platforms can be used to build applications for mobile devices, conceiving different types of applications. According to Budiu [4], one way of classifying different types of apps for smartphones and tablets is as: native apps, web apps or hybrid apps. According to that author, there is not a single best solution for mobile development: each approach has its weaknesses and strengths, in terms of performance, interaction and development capabilities. The choice of one or another option depends on the needs of each organisation. However, each implementation approach brings important implications for design decisions, particularly for accessibility.

Native applications are developed using specific languages according to each platform. For example, Android apps are developed in Java, iOS apps in Objective C and Windows phone apps in C++. In particular, native development for Android also enables the use of embedded web components using HTML (HyperText Markup Language), making it possible to mix web elements to native components. In the development of native apps, the code developed for one platform cannot be ported to another operating system directly, making it necessary to rewrite the entire code to have portability, which may make the process slower and more expensive.

When developing native apps, it is possible to explore the smartphone's resources to the fullest, such as sensors, accelerometer, compass, calling and interface, graphical processing and others. This enables building apps with many resources, able to process images and videos at high resolutions. These apps can use notifications and run off-line [2].

Web applications render web pages accessed via the Internet by means of a web browser installed on mobile devices. Those websites are developed using common technologies, such as HTML, CSS (Cascading Style Sheets) and JS (JavaScript), making the development process faster for many developers, not requiring specific coding for each different platform. On the other hand, web applications have limited access to hardware resources and data on devices, being also dependent on Internet connection for accessing content.

Hybrid development aims to combine the advantages of native development and web applications for creating mobile applications [2]. For this, different frameworks, tools and platforms have been deployed by developers, such as: Appcelerator Titanium[1], Adobe PhoneGap [2], Apache Cordova [3], Ionic[4], among others.

Those tools help the development by using web-based technologies, such as HTML, CSS and JS, so that, at the end of the process, an installable application be generated, working in the same fashion as a native app. This favours the possibility for portability and interoperability among different smartphones, such as Android, iOS and Windows Phone.

In the same way as web apps, hybrid apps also need a web navigator to render HTML. Hybrid apps use embedded web

[1]Appcelerator Titanium - Available on-line at http://www.appcelerator.com
[2]Adobe PhoneGap - Available on-line at http://build.phonegap.com
[3]Apache Cordova -Available on-on-line at http://cordova.apache.org
[4]Ionic - Available on-line at ionicframework.com

navigators made available by those platforms incorporated in the apps. In the case of Android, the navigator WebView is used in such cases.

Hybrid applications help the development of multi-platform apps, and hence, can reduce significantly the development cost. The same HTML components can be reused to generate installable apps in different operating systems for mobile devices.

In the present study, we investigated some of the technologies to develop mobile and web-hybrid applications for the Android platform, the most widely-used in Brazil [26]. We implemented native apps (with and without web components) using Google's Android Studio, and the frameworks Apache Cordova, Ionic and Appcelerator Titanium to build web-hybrid apps.

Frameworks such as Apache Cordova, Appcelerator Titanium and Ionic enable the use of standard web technologies (HTML, CSS and JS) to design user interfaces and to develop the functionalities of the application. Apps executed in each different platform generated by Apache Cordova rely on compatible connections with default interface components in each device, such as sensors, data, and others.

III. Related Studies on the Accessibility of Mobile Applications

A growing number of studies in the literature have dedicated efforts to research methods to evaluate the accessibility of mobile applications. However, fewer studies have focused on the way in which such applications are built and on the implications of the choices of types of interface components on the accessibility of those applications when used in mobile apps.

Many research studies have been conducted on the accessibility of applications and websites on mobile devices for screen reader users. In a study conducted by Chiti and Leporini [10], 4 visually-disabled users (two experienced and two beginner users of smartphones) evaluated a prototype of an Android application. Users provided useful information, such as issues with touchscreen gestures, which could help developers who work to conceive and develop assistive technologies for mobile devices.

Leporini *et al.* [18] later evaluated usability and accessibility problems encountered on Apple's mobile devices with the VoiceOver screen reader. Their research involved the usability inspection of the device's interface components and the input from 55 blind users about their experiences. The results confirmed that VoiceOver was fundamental for their users to interact with their mobile applications. However, there were still usability problems that needed fixing, such as the lack of clarity of details in certain elements, expansible navigation elements and handling form filling.

According to Siebra *et al.* [28], there are several initiatives for the development of recommendations for accessible mobile applications. However, existing recommendations consist mostly of suggestions and not a concrete list of functional requirements. According to those authors, it is still necessary to perform an assessment of requirements for each type of disability to better understand what impact they have on the usability of mobile applications. In their work, they elicited 36 requirements for mobile apps from the literature, being 13 of them focused on visual disabilities, followed by an analysis with visually-disabled users. However, they did not examine the influence of the means of implementation of applications on the accessibility of the resulting apps.

Another study, conducted by Shitkova *et al.* [27], sought to answer which usability guidelines should be considered when developing an accessible mobile website or app. They also investigated to which point those guidelines were applicable in the development process. Those authors proposed a catalog of usability guidelines and applied it in the development of a mobile app and website. However, it was not possible to perform a more in-depth evaluation of the relevance, utility, sufficiency and coverage of the guidelines, due to the nature of the research method applied to create the catalog.

Park *et al.* [20] evaluated how four visually-disabled participants performed certain tasks on their mobile devices. The results pointed out severe accessibility problems in typing tasks and in other tasks on VoiceOver screen reader. As a consequence, the authors of that paper proposed a set of 10 heuristics for the development of more accessible mobile applications.

Barriers from the use of touchscreen devices motivated the studies from Kane *et al.* [17] and Piccolo *et al.* [21]. The authors of those studies also defined a set of guidelines focused on accessible interactions for mobile devices.

Clegg-Vinell *et al.* [11] analysed the WCAG 2.0 guidelines in another paper. They invited people with different disabilities to evaluate several websites and apps in different mobile platforms, comparing the problems encountered by users with the guidelines. The results from that study pointed to the need of a new approach to the guidelines, combining elements of accessibility, usability and user experience. This would help optimise their efficacy and reduce the time spent by developers when applying them.

Casadei *et al.* [9] conducted a study to investigate the experiences of developers in terms of accessibility issues related to user interface design patterns for Android applications. By means of an ethnographic study in 18 virtual communities of developers, they identified recommendations related to input, list and pagination and navigation, as well as specific issues related to the use of the "hamburger menu" and tab navigation.

Silva *et al.* [24] investigated the accessibility of the WhatsApp application on mobile devices, involving five users with visual disabilities. Their study revealed problems with the lack of labels on buttons, difficulties to understand the options available on the app, the lack of sound feedback to actions taken by users.

Damaceno *et al.* [14] performed a review to identify accessibility problems encountered by people with visual disabilities on mobile devices, which were synthesised in the groups: borders not touch sensitive, buttons, voice commands, data entry, gesture interactions, screen reader issues, and user feedback.

They then performed a user study in order to identify which gestures were best for interactions with users, suggesting that gestures resembling the letters V, upside-down V and Z had better performance.

A recent study by Carvalho [7] involved 6 blind and 4 normal-vision users in the evaluation of 4 mobile applications and websites, resulting in 409 problems encountered by blind users and 105 problems by normal-vision users. The results showed that blind users were more severely impacted than other users with no disabilities that participated in the study, with problems related to navigation, lack of alternative text, unclear features when using with screen readers, among others.

In other recent studies from the research group of the authors of the present paper [6], [23], accessibility inspections of governmental mobile applications in Brazil were performed. The methodology of those studies helped form the basis of the method applied in this paper, with the adaptations carried out on WCAG 2.0 for the mobile context, as well as the accessibility problems identified.

IV. Methodology

This section presents the methodological aspects used in this paper, regarding the implementation of mobile applications using different platforms and approaches. It also describes the accessibility inspection methods used to evaluate the implemented prototypes, using an adaptation of the WCAG 2.0, tailoring them to the context of mobile applications, following the methods used and adapted in previous studies [6], [23].

A. Sample of Interface Components

We selected a sample of interface components, based on Web elements, from a categorisation performed by Freire [13] and Power *et al.* [22] in previous studies. Those authors categorised problems encountered by disabled users when using websites into six levels: Content, Delivery media, Web page structure, Website Navigation, Information Architecture and Underlying system characteristics. Each level presented subcategories that described the nature of the problem encountered by users. In particular, in the present work, we used two of those levels: Delivery media and Web page structure. Those levels relate to the structural layer in a web page, with 8 categories of web interface components and their equivalents in native applications, such as text, images, audio, video, multimedia and others.

In order to perform the accessibility inspections, 30 interface components from the web context were selected for the samples, drawn from the 8 categories in the aforementioned categorisation scheme. The 30 interface components were selected from an investigation in the official documentation of HTML and Android. All 30 interface components as listed as follows:

Media

- **Text:** text in paragraph, text in ordered list, text in unordered list, text in columns and text in citation.
- **Images:** informative image, decorative image, functional image, image of text, complex image, group of images and image maps.
- **Audio, video and multimedia:** video and audio player.
- **Other media types:** Mathematical and Chemistry formulae.

Web page structure

- **Headings:** section headers.
- **Links:** internal link, external link e link with image.
- **Tables:** table with one header, table with two headers, table with irregular headers, table with multi-layer headers e table with caption and summary.
- **Controls, forms and functionality:** selection box, radio button, button, list and text box.

B. Implementation of the interface components

In order to inspect the applications in the different platforms, we created prototypes of mobile apps containing the aforementioned interface components in their standard forms. Those apps represent examples of applications with such components and made it possible to verify the accessibility of the features with screen readers on mobile devices. The development of the prototypes was split between two of the authors, following the same principles. One author developed three prototypes and another developed two prototypes. The developers made all possible efforts to adhere their implementations to the guidelines in WCAG 2.0.

Each of the 30 components was implemented in five prototypes of mobile apps. The first prototype used standard interface components in Android studio for native development. In the second prototype, HTML interface components were included in a native app developed in Android studio, generating a native app with embedded web components. The third prototype used the framework Apache Cordova 6.3.1 to implement the components in a hybrid app. The fourth prototype was developed using the open source framework Ionic 3.19.1, also using HTML components to generate a hybrid application. Finally, the fifth prototype used the platform Appcelerator Titanium 5.0 to develop an also hybrid app from HTML interface components.

All prototypes were developed for the Android 6.0 operating system.

1) Purely native Android app with Android studio: The implementation of the prototype using native components was performed using native standard components, without the use of any Web elements. For this, some components had to be adapted, when a direct equivalent was not available.

The interface components “text in ordered list” and “text in unordered list” were replaced by an equivalent standard component in Android Studio - the ListView. The different types of markup for the component “section header” were not available in Android Studio and, hence, were replaced by visual representations of the component TextView. The components “internal link” and “external link” were also

adapted, considering that Android studio does not have the same specific markup for links as HTML does.

Table components were implemented in Android Studio using the GridView component, differently from HTML, which uses specific markup for table elements and its parts. Some more specific types of images also were adapted to the existing components available at Android Studio.

2) Native Android app with Android studio containing embedded web interface components: In the development of the prototype of the native app with embedded web components, no adaptations were needed to fit specific elements. Web-based components were implemented in the same way as in HTML and rendered by means of the WebView service.

On Android, WebView uses the native navigator from the platform. It applies the rendering motor Webkit to show web pages and to include methods to navigate in the history, apply zoom, perform searches, among others.

3) Web-hybrid application using Apache Cordova: Apache Cordova was used to develop the first of the hybrid application, generated from the code implemented as web pages. By default, we created a local file named "index.html", with references to CSS, JavaScript, images, media files and other resources that are necessary for the execution of the app. The application is rendered using the WebView component, available natively at the Android platform.

4) Web-hybrid application using Ionic: The hybrid application developed with Ionic was built using vestion 3.19.1 of the framework. The same HTML components used for the development of the native app with embedded web components were employed in this app, using the same methodology. At the end of the development of the application, we generated an *apk* file, which was installed in an Android device.

Ionic uses Cordova as its basis to access native resources from the device it is running on. The process to generate an apk file with Ionic uses Cordova's commands to generate and built the installable app. For instance, the command *ionic cordova build –release android* was used to compile the code on Ionic for the Android platform.

5) Web-hybrid application using Appcelerator Titanium: Appcelerator Titanium allows developers to create cross-platform applications using JavaScript code, which allows access to all Android APIs, for example. We used the App Builder Studio made available by Appcelerator, which allows development in a single, scalable environment to create, test, compact and publish mobile applications on various devices and operating systems.

The web-hybrid application developed from Appcelerator Titanium used the "Titanium.UI.createWebView" method to create the WebView and load the interface components into HTML5. It is important to mention that Appcelerator Titanium also provides methods that allow the development of native applications that use native components of the mobile operating system. In this work, we included the development and evaluation of the web-hybrid application created by Appcelerator Titanium, allowing the Web components used to be contrasted with the components present in other web-hybrids and with the native application developed from Android Studio.

C. Accessibility Inspections Procedure

The accessibility inspections were performed using review of guidelines on interface components of the developed prototypes. The inspections involved verifying the interface and source codes of the applications by means of verifying whether all interface components adhered or not to the guidelines under consideration. Considering this, each success criterion would be deemed as having passed or failed.

Evaluators: The inspections were performed by two of the authors, who have extensive experience with the design and evaluation of mobile applications and with accessibility guidelines. The two evaluators had a Computer Science degree, and had been working for at least two years on projects related to mobile accessibility, and were well versed with WCAG 2.0. Each prototype was inspected by another author who had not been involved in its development. The evaluations were overseen by the third author, who has more than ten years of experience with accessibility. The third author checked for possible inconsistencies between the evaluations, and suggested corrections to ensure reliability, with follow-up discussions between the three authors when disagreements occurred.

As previously mentioned, currently there are not consolidated accessibility guidelines for the evaluation of mobile applications. Despite the existence of guides for Apple's iOS and Google's Android, those are more restricted to technical aspects related to the coding of specific interface components. Those guides do not include broader accessibility issues present in wider accessibility guidelines, such as the Web Content Accessibility Guidelines. Thus, we analysed different possibilities of guidelines sets for the inspections, such as ISO 9241-171 [16] and WCAG 2.0 [5]. However, those recommendations were not specific to the mobile context.

The set of guidelines for the inspections: According to the Web Accessibility Initiative (WAI) [29] from the World Wide Web Consortium, who published the WCAG 2.0, the main technologies endorsed by the W3C provide support to accessibility, including those most essential to the Web on mobile devices. The W3C has made efforts in version 2.0 of WCAG to make it less technology-specific as its previous version. Further to this, most standard interface components on desktop systems have equivalents in smartphones, including texts, hyperlinks, tables, buttons, menus and others. Considering those points, we opted for WCAG 2.0 to support the accessibility inspections on the developed prototypes, after performing adaptations to specific issues related to the interaction with mobile devices.

For the adaptation, we analysed all 61 success criteria in WCAG 2.0 and verified which criteria were not directly applicable in the case of mobile devices, in order to perform the due

adaptations. The success criteria that needed adaptations were those corresponding to enabling keyboard navigation, specific web markup, layout consistency and text resizing.

One of the adaptations performed was related to success criteria 2.1.1 and 2.1.3, which state that "Keyboard: All functionality of the content is operable through a keyboard interface without requiring specific timings for individual keystrokes". Success criteria 2.1.1 allows for exceptions for essential functions, and 2.1.3 allows no exceptions. When using assistive technologies in mobile devices, such as screen readers, users interact with icons, buttons and other elements using gestures, such as swipe right to go to the next element on the screen, or swipe left, to go the the previous one, or double tap to activate a given feature. Those gestures emulate the "tab" key used on keyboard navigation in desktop interaction for screen-reader users.

Other success criteria that were strongly tied to web technologies, such as HTML, had to be adapted. Among those criteria, we can mention success criterion 4.1.1, which states: "Parsing: In content implemented using markup languages, elements have complete start and end tags, elements are nested according to their specifications, elements do not contain duplicate attributes, and any IDs are unique, except where the specifications allow these features". To inspect the developed prototypes, evaluators should also consider other technologies, such as Java, used in Android Studio for the development of native Android applications. In such cases, evaluators should verify faults in the coding of interface components that could cause problems for users of assistive technologies.

Auditing procedures with screen readers: After adapting the success criteria for the context of mobile devices, two specialists inspected the applications using all 61 success criteria in WCAG 2.0. They then audited the 30 interface components in each of the five prototypes. For this, they used the basic gestures used in TalkBack by visually-disabled users, to navigate between elements and activate features.

Evaluators then recorded each problem encountered as having passed or failed each success criterion, using a checklist containing all success criteria in WCAG 2.0 and the adaptations previously described. In this evaluation in particular, we did not observe the numbers of instances of violations of each success criterion. The reason for this was the fact that the prototypes under evaluation had included only one interface component of each type.

The manual inspections were performed using the TalkBack screen reader version 5.1.0.12 in a Moto G 2nd generation smartphone with the operating system Android 6.0.

Other screen readers for Android are available, such as Mobile Accessiblity[5] and Samsung's Voice Assistant[6]. However, according to the latest survey with screen reader users conducted by WebAIM [30], TalkBack was the most widely used screen reader by Android users.

[5]Available at http://codefactoryglobal.com/app-store/mobile-accessibility/

[6]Available at https://www.samsung.com/uk/accessibility/mobile-voice-assistant/

V. Results and Discussion

This section presents the main results obtained from the accessibility inspections of the five implemented prototypes, followed by a discussion of implications.

The implications and suggested are based on the inspections performed and on the behaviour of the screen reader with each element.

A. Detailed analysis of the accessibility inspection of interface components

In this section, we present the detailed results of the accessibility inspection of the sample of interface components used with TalkBack, considering the 61 success criteria in WCAG 2.0. The inspection outcome for each success criterion could lead to "P" (Pass), in cases in which the component met all the requirements in the success criterion and worked appropriately with the screen reader, or "F" (Fail), when it did not meet the recommendations and did not work appropriately with the screen reader.

Considering that several success criteria were only applicable to certain interface components, the tables in the following sections present a selection of the main success criteria related to each type of component.

1) Text: Textual components are important structures in the development of an application. They enable the use of texts in paragraphs, lists, column arrangements and citations, and convey a significant part of the information available in an app to users.

During the accessibility inspections of text components in web-hybrid apps and in the native app with embedded web components, only success criterion 2.4.7 - focus visible was violated for the component "text in columns", as shown in Table I. In this case, TalkBack did not visually highlight the component in focus, making the interaction more difficult for certain users, such as people with low vision who use their residual vision along with the screen reader.

In the native application (without embedded web components), textual components in lists (ordered and unordered) presented problems with TalkBack. As previously mentioned, the standard Android ListView component was used as an alternative to text in lists. Different from list elements in HTML, TalkBack could not relate the items in the list in the native components as list elements of numbering them, thus violating success criteria 1.3.1 - Info and relationships.

According to WCAG 2.0, it is important to enable users to separate foreground and background information, making the presentation in its standard form adequate for people with visual disabilities. In this context, enabling the resizing of text and changing the visual presentation of blocks of text make it possible to customise content according to users' needs. Android provides tools such as zooming gestures, set larger text sizes, high contrast text, inverted colours and colour corrections. Those features enable changes in the presentation,

TABLE I
ACCESSIBILITY INSPECTION TEXT COMPONENTS

Component Interface	Native (without Web)	Native (with Web)	Hybrid (Cordova)	Hybrid (Appcelerator)	Hybrid (Ionic)
Text in paragraph	1.4.4 - P	1.4.4 - P	1.4.4 - P	1.4.4 - P	1.4.4 - P
	1.4.8 - P	1.4.8 - P	1.4.8 - P	1.4.8 - P	1.4.8 - P
	2.4.7 - P	2.4.7 - P	2.4.7 - P	2.4.7 - P	2.4.7 - P
	3.1.4 - F	3.1.4 - P	3.1.4 - P	3.1.4 - P	3.1.4 - P
Text in ordered list	1.3.1 - F	1.3.1 - P	1.3.1 - P	1.3.1 - P	1.3.1 - P
Text in unordered list	1.3.1 - F	1.3.1 - P	1.3.1 - P	1.3.1 - P	1.3.1 - P
Text in columns	2.4.7 - P	2.4.7 - F	2.4.7 - F	2.4.7 - F	2.4.7 - F
Text in citation	1.4.4 - P	1.4.4 - P	1.4.4 - P	1.4.4 - P	1.4.4 - P
	1.4.8 - P	1.4.8 - P	1.4.8 - P	1.4.8 - P	1.4.8 - P

1.3.1 Info and Relationships, 1.4.4 Resize Text, 1.4.8 Visual Presentation, 2.4.7 Focus Visible, 3.1.4 Abbreviations

but require implementations that do not hinder their functioning. The component "text in paragraph", specifically, was in accordance with success criteria 1.4.4 - resize text and 1.4.8 - visual presentation.

In summary, despite occasional violations in some success criteria, textual components were in general more accessible in hybrid apps and native apps with embedded web components. In such cases, TalkBack was able to have access to more information about the semantics and behaviour of the text components. Figure 1 presents an ilustration of the textual elements in the five apps.

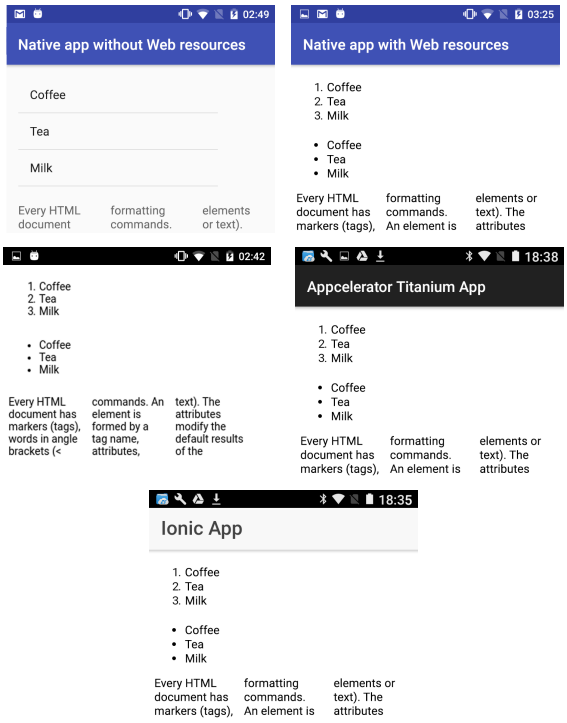

Fig. 1. Components "text in sorted list", "text in unordered list" and "text in columns" implemented in the native application (without Web resources), native application (with Web resources), Cordova application, Appcelerator application and Ionic application, respectively.

2) Images: In order to provide accessibility, images have provide textual alternatives that describe their content or function. This makes it possible for people with several disabilities to have access to their content. Despite having an overall good performance in the evaluation, some of the image components present in the prototypes still had violations of some specific success criteria. According to Table II, the component "group of images" did not meet success criterion 1.3.1 - Info and relationships in any of the five developed prototypes. TalkBack failed to identify the relationship between the images in the group.

For success criterion 1.1.1 - Non-textual content, all five platforms provided resources to include textual alternatives in the images. The component "image of text", for example, which shows text coded as an image (as shown in Figure 2), provides the possibility to include alternative text to describe the information in it. It is important that developers pay attention to the use of appropriate markup of non-textual content to enable its use by screen-reader users. In AndroidStudio, for example, developers should use the tag *android:contentDescription="image description"*, and in HTML, the attribute *alt="description"* to provide a descriptive text representing the content of the image.

Fig. 2. Components "group of images" and "text images" implemented in the native application (without Web resources), native application (with Web resources), Cordova application, Appcelerator application and Ionic application, respectively.

3) Audio, Video and Multimedia: HTML 5 introduced two new multimedia tags to display audio and video on web pages.

TABLE II
ACCESSIBILITY INSPECTION OF IMAGE COMPONENTS

Component Interface	Native (without Web)	Native (with Web)	Hybrid (Cordova)	Hybrid (Appcelerator)	Hybrid (Ionic)
Informative image	1.1.1 - P	1.1.1 - P	1.1.1 - P	1.1.1 - P	1.1.1 - P
Decorative image	1.1.1 - P	1.1.1 - P	1.1.1 - P	1.1.1 - P	1.1.1 - P
Functional image	1.1.1 - P	1.1.1 - P	1.1.1 - P	1.1.1 - P	1.1.1 - P
Image of text	1.1.1 - P	1.1.1 - P	1.1.1 - P	1.1.1 - P	1.1.1 - P
	1.4.5 - P	1.4.5 - P	1.4.5 - P	1.4.5 - P	1.4.5 - P
	1.4.9 - P	1.4.9 - P	1.4.9 - P	1.4.9 - P	1.4.9 - P
Complex image	1.1.1 - P	1.1.1 - P	1.1.1 - P	1.1.1 - P	1.1.1 - P
	1.3.1 - F	1.3.1 - P	1.3.1 - P	1.3.1 - P	1.3.1 - P
Group of images	1.1.1 - P	1.1.1 - P	1.1.1 - P	1.1.1 - P	1.1.1 - P
	1.3.1 - F	1.3.1 - P	1.3.1 - P	1.3.1 - P	1.3.1 - P
Image maps	1.1.1 - P	1.1.1 - P	1.1.1 - P	1.1.1 - P	1.1.1 - P
	1.3.1 - F	1.3.1 - P	1.3.1 - P	1.3.1 - P	1.3.1 - P
	2.4.7 - P	2.4.7 - P	2.4.7 - P	2.4.7 - P	2.4.7 - F

1.1.1 Non-text Content, 1.3.1 Info and Relationships, 1.4.5 Images of Text, 1.4.9 Images of Text (No Exception)

This enabled different file types to be used in different navigators. Multimedia components may include images, songs, sounds, videos, record, films and animations, which can be incorporated in several mobile applications.

The video player (Figure 3) and audio player components, present in the sample, present similar violations of WCAG 2.0 success criteria in the five prototypes developed. Among those violations, we can mention lack of description of standard button to play video, problems showing subtitles on videos, and the lack of consolidated means to insert audio description, subtitles and sign language in the audio and video components in mobile devices.

According to the results presented in Table III, the interface components related to audio, video and multimedia still lack ideal accessibility support for the TalkBack screen reader. Developers who wish to use audio, video and multimedia components in an accessible manner should be aware that some users might not have access to that content, and think of alternative solutions to provide adequate accessibility while more definite solutions be implemented.

4) Other media types: In this study, we also investigated the accessibility to other media types in mobile applications, such as mathematical and chemical notation. Current development resources for mobile devices, as well as screen readers such as TalkBack, have not yet provided adequate support for more complex formulae and mathematical equations. Current web-based and Android native components to not provide resources to visually render mathematical equations and formulae in standard notation, such as MathML (Mathematical Markup Language), as shown in Figure 4.

Table IV shows that those types of special markup elements in the interface of the 5 prototypes violated success criterion 4.1.2 - name, role and value. Those components were completly incompatible with the TalkBack screen reader, preventing important information related to the role, state and value from being synthesised in voice to blind users.

In this case, providing alternative texts to mathematical and chemical formulae in graphics can be an alternative solution for the time being. Despite being in accordance to current

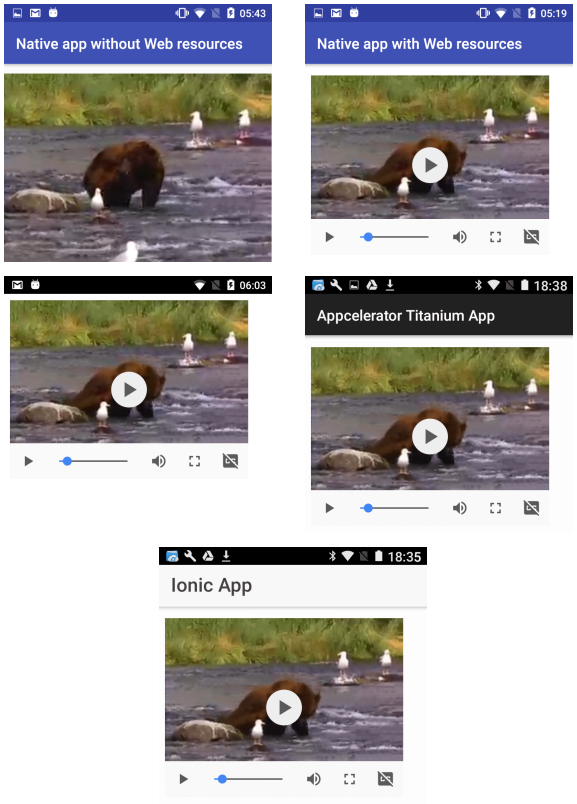

Fig. 3. Video player component implemented in the native application (without Web resources), native application (with Web resources), hybrid application, Appcelerator application and Ionic application, respectively.

accessibility guidelines, this solution is not as rich as making mathematical and chemical content available in a way that enables interaction and navigation to interpret its meaning. Different applications could be used to work around this issue to make this type of content more accessible to screen readers. The use of MathML (Mathematical Markup Language) would enable screen readers to describe mathematical notation and inform users about its structure. However, in the particular case

TABLE III
ACCESSIBILITY INSPECTION OF AUDIO, VIDEO AND MULTIMEDIA COMPONENTS

Component Interface	Native (without Web)	Native (with Web)	Hybrid (Cordova)	Hybrid (Appcelerator)	Hybrid (Ionic)
Video player	1.2.1 - F	1.2.1 - F	1.2.1 - F	1.2.1 - F	1.2.1 - F
	1.2.2 - F	1.2.2 - F	1.2.2 - F	1.2.2 - F	1.2.2 - F
	1.2.3 - F	1.2.3 - P	1.2.3 - P	1.2.3 - P	1.2.3 - P
	4.1.2 - F	4.1.2 - P	4.1.2 - P	4.1.2 - P	4.1.2 - P
Audio player	1.2.1 - F	1.2.1 - F	1.2.1 - F	1.2.1 - F	1.2.1 - F
	1.2.2 - F	1.2.2 - F	1.2.2 - F	1.2.2 - F	1.2.2 - F
	1.2.6 - P	1.2.6 - P	1.2.6 - P	1.2.6 - P	1.2.6 - P

1.2.1 Audio-only and Video-only (Prerecorded), 1.2.2 Captions (Prerecorded), Audio Description or Media Alternative (Prerecorded), 1.2.6 Sign Language (Prerecorded), 4.1.2 Name, Role, Value

TABLE IV
ACCESSIBILITY INSPECTION OF COMPONENTS OF OTHER MEDIA TYPES

Component Interface	Native (without Web)	Native (with Web)	Hybrid (Cordova)	Hybrid (Appcelerator)	Hybrid (Ionic)
Mathematical notation	4.1.2 - F	4.1.2 - F	4.1.2 - F	4.1.2 - F	4.1.2 - F
Chemical formula	4.1.2 - F	4.1.2 - F	4.1.2 - F	4.1.2 - F	4.1.2 - F

4.1.2 Name, Role, Value

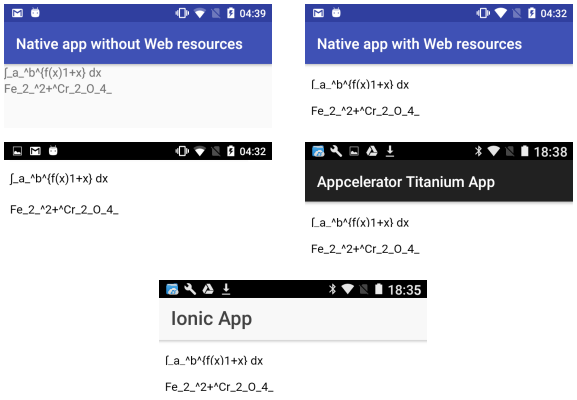

Fig. 4. Components ”mathematical notation” and ”chemical formula” implemented in the native application (without Web resources), native application (with Web resources), Cordova application, Appcelerator application and Ionic application, respectively.

of Brazil, even desktop-based screen readers do not currently have support to reading mathematical formulae in Portuguese, let alone for mobile devices.

Developers who wish to use the aforementioned components in their applications should seek for alternative solutions to integrate in their applications, in order to make their content presentable visually and compatible with the main screen readers available on the market.

5) Headings: HTML enables the use of up to 6 levels of headings to define titles and subtitles in documents. A heading is defined as delimited by its correspondent tag, usually applying the styles with bold and a specific font size, according to the level. Android Studio does not offer specific markup options for different levels of headings, as shown in Figure 5. The lack of such tags in native applications not using embedded web components impacts directly on the ability of screen readers to convey information to help blind users understand the structure of the content. Further to this, TalkBack has a feature that enables users to navigate between the headings of a page, making it possible for blind users to have a better overall view of the screen, making navigation more efficient. However, this feature is currently available only to web pages, due to the lack of structural markup in native components.

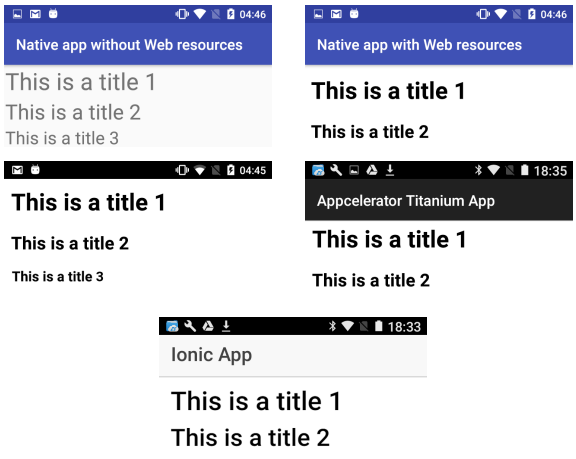

Fig. 5. Component ”title of the sections” implemented in the native application (without Web resources), native application (with Web resources), Cordova application, Appcelerator application and Ionic application, respectively.

According to the main results of the accessibility inspection of the section title components, shown in Table V, hybrid applications and applications with embedded web components had a far superior performance regarding headings than the purely native app. In this context, developers of applications

with different sections, where navigation speed is important, should consider using hybrid approaches of native apps with embedded web components.

6) Links: A link is a connector between one web resource to another. In mobile applications, a link can be used to have access, for example, to an image, a video clip, an audio excerpt, a new page in the application, notifications, an HTML document or others.

WCAG 2.0 provides two specific success criteria dealing with link purpose: one regarding the purpose of a link when read in context (2.4.4) and another one considering the understanding of the purpose of a link with its description alone (2.4.9). According to these success criteria, there should be a mechanism to allow the understanding of a link when read out of context (in a level-AAA criterion), or when the link is read alongside surrounding contextual content, such as paragraphs, lists, headings, and others (level-A criterion). In the inspections, TalkBack did not encounter issues to identify internal links, external links and links in images, making it possible to determine the purpose of a link and explore its content in all platforms investigated. An example of external links implemented in the five prototypes is shown in Figure 6.

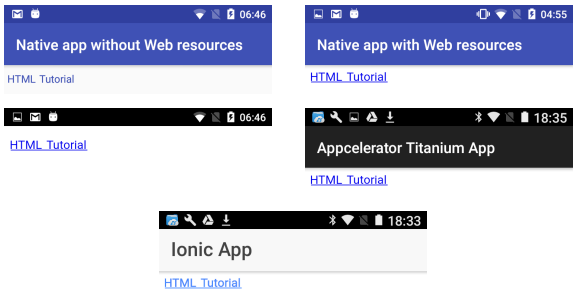

Fig. 6. Component ”external link” implemented in the native application (without Web resources), native application (with Web resources), Cordova application, Appcelerator application and Ionic application, respectively.

Success criterion 4.1.2 - name, role and value also provides specific recommendations for links in HTML, which can be applicable to both native and hybrid applications, as shown in Table VI. This success criterion recommends that links should be used with adequate attributes and values, which could enable assistive technologies to operate those components accordingly. There are different means to implement links in native applications without using web components on Android Studio. However, there are alternatives which would be not accessible to screen readers, resulting in links without name, description and values.

7) Tables: Tables are used to organize data by means of a grid layout with logical relation between them. In order to make tables accessible, it is very important to use appropriate tags to indicate headers and their relation to data cells. Screen readers rely on that information to explicitly convey its meaning. The following code snippet illustrates how to properly code tables relating cells and headers, using the tag $< th >$. The resulting table from that code is shown in Figure 7.

```
<table>
  <caption>Opening hours</caption>
  <tr>
    <td></td>
    <th>Sunday</th>
    <th>Monday</th>
  </tr>
  <tr>
    <th>08:00 - 12:00</th>
    <td>Closed</td>
    <td>Open</td>
  </tr>
  <tr>
    <th>12:00 - 18:00</th>
    <td>Closed</td>
    <td>Open</td>
  </tr>
</table>
```

Listing 1. Example of a marked-up table in HTML

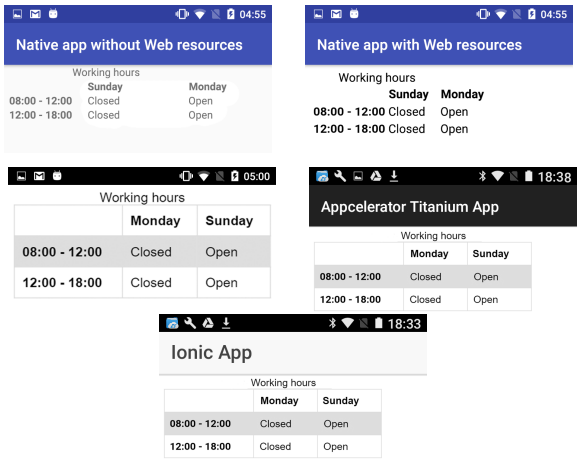

Fig. 7. Component ”table with two headers” implemented in the native application (without Web resources), native application (with Web resources), Cordova application, Appcelerator application and Ionic application, respectively.

Android Studio does not provide components with tags to properly mark table headers and their relation with data cells, which causes serious problems for disabled users with assistive technologies, such as screen readers. In the inspections performed in this study, TalkBack could not relate to which column a given piece of information in a data cell was related to in the heading columns. It is important that developers pay attention to find appropriate alternatives to enable assistive technologies to establish those relationships to make those tables accessible to all users. In the case of technologies that do not support automatic definition of those relationships, this could be done explicitly on text. Table VII presents the two success criteria that failed for the native application that did not use web components.

TalkBack also encountered barriers to identify the relationships in more complex tables in applications that use HTML-based components, such as those with irregular headings and with several levels of headings.

TABLE V
ACCESSIBILITY INSPECTION OF HEADING COMPONENTS

Component Interface	Native (without Web)	Native (with Web)	Hybrid (Cordova)	Hybrid (Appcelerator)	Hybrid (Ionic)
Section title	1.3.1 - F	1.3.1 - P	1.3.1 - P	1.3.1 - P	1.3.1 - P
	2.4.2 - P	2.4.2 - F	2.4.2 - F	2.4.2 - F	2.4.2 - F
	2.4.6 - P	2.4.6 - P	2.4.6 - P	2.4.6 - P	2.4.6 - P
	2.4.10 - P	2.4.10 - P	2.4.10 - P	2.4.10 - P	2.4.10 - P
	4.1.2 - F	4.1.2 - P	4.1.2 - P	4.1.2 - P	4.1.2 - P

1.3.1 Info and Relationships, 2.4.2 Page Titled, 2.4.6 Headings and Labels, 2.4.10 Section Headings, 4.1.2 Name, Role, Value

TABLE VI
ACCESSIBILITY INSPECTION OF LINKS COMPONENTS

Component Interface	Native (without Web)	Native (with Web)	Hybrid (with Web)	Hybrid (Appcelerator)	Hybrid (Ionic)
Internal link	2.4.4 - P	2.4.4 - P	2.4.4 - P	2.4.4 - P	2.4.4 - P
	2.4.9 - P	2.4.9 - P	2.4.9 - P	2.4.9 - P	2.4.9 - P
	4.1.2 - F	4.1.2 - P	4.1.2 - P	4.1.2 - P	4.1.2 - P
External link	2.4.4 - P	2.4.4 - P	2.4.4 - P	2.4.4 - P	2.4.4 - P
	2.4.9 - P	2.4.9 - P	2.4.9 - P	2.4.9 - P	2.4.9 - P
	4.1.2 - F	4.1.2 - P	4.1.2 - P	4.1.2 - P	4.1.2 - P
Link in image	2.4.4 - P	2.4.4 - P	2.4.4 - P	2.4.4 - P	2.4.4 - P
	2.4.9 - P	2.4.9 - P	2.4.9 - P	2.4.9 - P	2.4.9 - P
	4.1.2 - F	4.1.2 - P	4.1.2 - P	4.1.2 - P	4.1.2 - P

2.4.4 Link Purpose (In Context)), 2.4.9 Link Purpose (Link Only), 4.1.2 Name, Value, Role

TABLE VII
ACCESSIBILITY INSPECTION OF TABLE COMPONENTS

Component Interface	Native (without Web)	Native (with Web)	Hybrid (Cordova)	Hybrid (Appcelerator)	Hybrid (Ionic)
Table with a header	1.3.1 - F	1.3.1 - P	1.3.1 - P	1.3.1 - P	1.3.1 - P
	1.3.2 - F	1.3.2 - P	1.3.2 - P	1.3.2 - P	1.3.2 - P
	1.4.8 - P	1.4.8 - P	1.4.8 - P	1.4.8 - P	1.4.8 - F
Table with two headers	1.3.1 - F	1.3.1 - P	1.3.1 - P	1.3.1 - P	1.3.1 - P
	1.3.2 - F	1.3.2 - P	1.3.2 - P	1.3.2 - P	1.3.2 - P
	1.4.8 - P	1.4.8 - P	1.4.8 - P	1.4.8 - P	1.4.8 - F
Table with irregular headers	1.3.1 - F	1.3.1 - F	1.3.1 - P	1.3.1 - P	1.3.1 - P
	1.3.2 - F	1.3.2 - F	1.3.2 - P	1.3.2 - P	1.3.2 - P
	1.4.8 - P	1.4.8 - P	1.4.8 - P	1.4.8 - P	1.4.8 - F
Table with multi-level headers	1.3.1 - F	1.3.1 - F	1.3.1 - F	1.3.1 - F	1.3.1 - F
	1.3.2 - F	1.3.2 - F	1.3.2 - F	1.3.2 - F	1.3.2 - F
	1.4.8 - P	1.4.8 - P	1.4.8 - P	1.4.8 - P	1.4.8 - F
Table with caption and abstract	1.3.1 - F	1.3.1 - P	1.3.1 - P	1.3.1 - P	1.3.1 - P
	1.3.2 - F	1.3.2 - P	1.3.2 - P	1.3.2 - P	1.3.2 - P
	1.4.8 - P	1.4.8 - P	1.4.8 - P	1.4.8 - P	1.4.8 - F

1.3.1 Info and Relationships, 1.3.2 Meaningful Sequence, 1.4.8 Visual Presentation

According to the results presented, tables developed with web-based components provide more resources to allow screen readers as TalkBack to provide users with more detailed information and their structure. Unfortunately, default interface components from Android, such as GridView did not provide the same features. The use of tables in native applications that do not embed web components may create serious barriers to blind users with screen readers. Developers of mobile applications should use alternative components or alternative means of presenting information, making sure that these alternatives are also accessible to disabled people.

8) Controls, Forms and Functionality: Controls, forms and dynamic features are used to provide important means of interaction for users on websites and mobile applications. The five interface components implemented to be tested in this category had positive results in the accessibility inspections. We could not find any violation to any WCAG 2.0 success criteria, and they showed good possibilities for the implementation of accessibility features for screen readers. They allowed screen readers to properly identify and facilitate the understanding of functionality and behaviour of form controls, by associating labels and other structural elements.

In general, it is important for developers to pay attention to the labelling of all input elements. In HTML, for example, the elements *label* and the attribute *title* should be use to help blind users have confidence to interact with an application, be

it searching for information or entering personal data, such as credit card or phone numbers. Table VIII shows the success criteria related to those elements that were inspected during the evaluations.

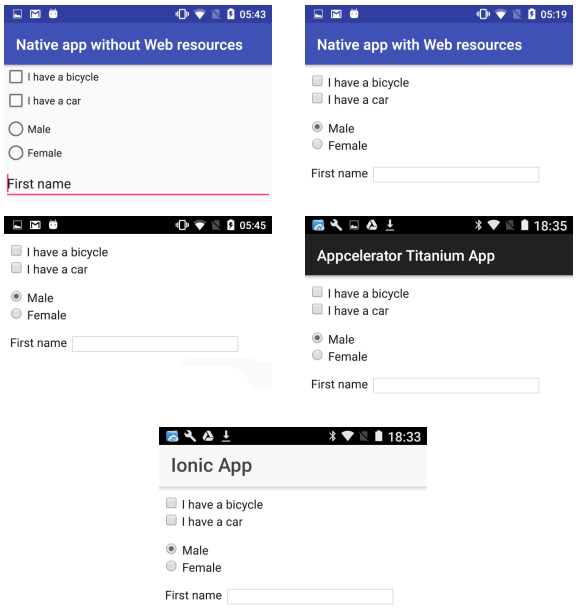

Fig. 8. Components ”radio button”, ”checkbox” and ”text field” implemented in the native application (without Web resources), native application (with Web resources), Cordova, Appcelerator application and Ionic application, respectively.

Figure 8 presents an example of the main interface components related to controls, forms and features created with the five development platforms developed in this study.

B. Discussion

The results of the analyses reported in this paper presented positive aspects related to the implementation of mobile applications using web-based components, both in the case of hybrid applications using Cordova, Appcelerator Titanium and Ionic, and in the case of native applications with embedded web components.

The results of the present paper confirm the findings of a previous study of the authors [8], in which the comparison had been performed using native applications implemented only using Cordova.

In general, web-based components provided more accessibility resources to users who use the TalkBack screen reader. However, in the case of more complex interface components, such as text in columns, tables with several layers of headings, audio, video, and other types of non-conventional media did not provide elements to meet some WCAG 2.0 success criteria even with web-based components. This was the case on Cordova and on the new development platforms included in the new prototypes developed in this paper using Appcelerator Titanium and Ionic.

In the purely native application, which did not use web components, we found a relatively larger number of violations than in the other four prototypes. The native Android components had the highest number of violations of WCAG 2.0 success criteria 1.3.1 - Info and relationships and 4.1.2 - name, role and value, related to the principles of making applications perceivable and robust. The accessibility problems encountered in native components implemented in Android Studio may imply in the lack of presentation of information or difficulties to interpret content by many users who use assistive technologies.

It is also worth noting that the results with hybrid apps and native apps with embedded web components had the same performance. This is largely due to the fact that they all use the component WebView to render web components. When viewing the application, there is little difference for users between hybrid applications and native apps with embedded web elements. However, native apps with embedded web components are more limited in terms of the web features they use, and those features have more limitations in terms of the interaction with the integration with native services available in Android Studio. Hybrid applications, on the other hand, can enable the use of the full potential of HTML, CSS and JavaScript, and other services in the cloud, with the possibility to integrate with other services from the mobile device.

This shows that hybrid applications, besides being a good strategy to promote portability of applications [19], can also help provide better accessibility with its interface components for the TalkBack screen reader.

With the development and inspection of a wider range of hybrid mobile applications using Cordova, Appcelerator Titanium and Ionic, in the present paper we showed that different platforms for developing web-hybrid Android applications did not have an impact on the accessibility of the resulting apps. With this, we can confirm the relationship between the behaviour of such applications in terms of the development approach (web-hybrid or native), independent of the platform in which web-hybrid applications have been developed.

It is worth noting that the inspections were performed using only one screen reader in the Android, and that different results could be obtained with evaluation in other screen readers. However, we can point to the fact that the accessibility limitations encountered in the native Android applications were due to limitations in the way in which the platforms implements such components. At the moment, they do not offer accessibility features that could be used by either TalkBack or other screen reader used at Android.

Despite the advantages in accessibility provision of hybrid applications, this paper has not approached issues related to the efficiency and processing performance of those apps. Due to the use of the WebView component as an intermediary, hybrid apps can lag in performance in particular contexts, such as real-time systems, which were not the focus of this paper.

The lack of well-defined accessibility guidelines for mobile applications, be it for native of web-based apps, still imposes important barriers for developers due to the lack of standardisation of interface components. While more mature accessibility resources are not available for native components,

TABLE VIII
ACCESSIBILITY INSPECTION OF CONTROLS, FORMS AND FUNCTIONALITIES COMPONENTS

Component Interface	Native (without Web)	Native (with Web)	Hybrid (Cordova)	Hybrid (Appcelerator)	Hybrid (Ionic)
Checkbox	3.3.2 - P	3.3.2 - P	3.3.2 - P	3.3.2 - P	3.3.2 - P
Radio button	3.3.2 - P	3.3.2 - P	3.3.2 - P	3.3.2 - P	3.3.2 - P
Button	4.1.2 - P	4.1.2 - P	4.1.2 - P	4.1.2 - P	4.1.2 - P
Drop-down list	3.3.2 - P	3.3.2 - P	3.3.2 - P	3.3.2 - P	3.3.2 - P
Text field	3.2.2 - P	3.2.2 - P	3.2.2 - P	3.2.2 - P	3.2.2 - P
	3.3.1 - P	3.3.1 - P	3.3.1 - P	3.3.1 - P	3.3.1 - P
	3.3.2 - P	3.3.2 - P	3.3.2 - P	3.3.2 - P	3.3.2 - P
	4.1.2 - P	4.1.2 - P	4.1.2 - P	4.1.2 - P	4.1.2 - P

3.2.2 On Input, 3.3.1 Error Identification, 3.3.2 Labels or Instructions, 4.1.2 Name, Role, Value

developers should seek alternative solutions to make their apps accessible to people with visual disabilities.

VI. CONCLUSIONS AND FUTURE WORK

The research study presented in this paper involved the implementation of 30 interface components in five different platforms to develop mobile applications for Android. This was made by means of the implementation of five different prototypes, employing the best available accessibility techniques to those interface components. The study then involved the accessibility inspection of the implemented prototypes using a screen reader.

The results from a previous study from the authors [8] had shown that hybrid applications developed with Apache Cordova and native applications with embedded web components were more compatible with WCAG 2.0 success criteria than applications with purely native Android interface components. In the present paper, we have extended the analyses to include other platforms to generate hybrid applications, namely Appcelerator Titanium and Ionic. The results from the present study have confirmed that hybrid applications, even in other platforms, were indeed more compatible with accessibility requirements, with no differences encountered in terms of accessibility compatibility in Apache Cordova, Ionic or Appcelerator Titanium. Despite having similar results in terms of accessibility, the development of native apps with embedded web components may have negative impacts on the overall performance of the app.

With regards to the development of purely native Android applications using interface components available in Android Studio, the results indicate that developers should be cautious when implementing them. The accessibility inspections revealed that several of those components may have problems providing accessibility features, particularly in content with tables, headings, and multimedia. Developers concerned with the accessibility of their apps should be aware of those shortcomings and propose alternative solutions to avoid accessibility problems that could prevent disabled users from using their features.

As to the developers of Android interface components, it would be important to consider those results to improve the provision of accessibility features that developers and designers could use to make more accessibility interactive systems in native Android apps.

Despite the discussions about the accessibility in hybrid and native applications, we acknowledge that the choice from developers need to take other factors into account, which were not in the scope of the present paper. As pointed out by Bosnic *et al.* [2], the development of hybrid and native mobile applications have particular advantages and disadvantages. While hybrid development approaches enable the generation of applications for multiple platforms, native development can be more suitable for more complex development and have superior processing performance.

Although there are several recommendations to make mobile applications more accessible, it is very important to advance in the definition of a more consolidated set of accessibility guidelines for mobile devices. Those guidelines need to take into consideration the specificities of each type of user and the different technologies used in the development of such applications. Even though several studies have used adaptations of existing guidelines for other platforms, Costa *et al.* [12] alert that the use of accessibility recommendations tailored to specific devices and contexts can yield significantly better results. The results from the present paper reinforce this need and provide some leads to guide future research in the area.

As future work, we intend to select a sample of mobile applications from Google's Play Store that were built using the different platforms investigated in this study, and to carry out tests with disabled users performing tasks with those applications. Following this, we intend to identify the main problems encountered by those users on native and hybrid apps, in order to help consolidate specific guidelines and recommendations for those platforms.

In future works, we also intend to perform accessibility inspections using other less-frequently used screen readers in Android, and to analyse the differences between the accessibility of native and web-hybrid applications in the Apple's iOS platform.

ACKNOWLEDGEMENTS

We thank CNPq (proc. 448521/2014-8) and FAPEMIG for the financial support to this project. We also thank the

anonymous reviewers for their thoughtful contributions to this work.

REFERENCES

[1] AppBrain, "Number of Android applications", Available online at https://www.appbrain.com/stats/number-of-android-apps, last accessed on 30th January 2018, 2017.

[2] S. Bosnic, I. Papp, and S. Novak "The development of hybrid mobile applications with Apache Cordova", In: Telecommunications Forum (TELFOR), 2016 24th, pages 1-4, IEEE, 2016.

[3] Brazil, "eMAG - Accessibility Model for Electronic Government" (in Portuguese) - Version 3.1,Available online at http://emag.governoeletronico.gov.br/, last accessed on 30th January 2017, 2014.

[4] R. Budiu, "Mobile: Native apps, web apps, and hybrid apps", Available online at https://www.nngroup.com/articles/mobile-native-apps/, last accessed on 30th January 2018, Nielsen Norman Group, 2013.

[5] B. Caldwell, M. Cooper, L. G. Reid, and G. Vanderheiden, "Web Content Accessibility Guidelines (WCAG) 2.0", available online at https://www.w3.org/TR/WCAG20/, last accessed on 30th January 2018, World Wide Web Consortium (W3C), 2008.

[6] L. P. Carvalho, B. P. M. Peruzza, F. Santos, L. P. Ferreira, A. P. Freire, "Accessible Smart Cities?: Inspecting the Accessibility of Brazilian Municipalities' Mobile Applications", In: Proceedings of the 15th Brazilian Symposium on Human Factors in Computing Systems, IHC '16, São Paulo, Brazil, Article 17, 2016.

[7] M. C. N. Carvalho, F. S. Dias, A. G. S. Reis, A. P. Freire, "Accessibility and Usability Problems Encountered on Websites and Applications in Mobile", In: Proceedings of the 33rd ACM/SIGAPP Symposium On Applied Computing, Pau, France, p. 2022-2029, 2018.

[8] L. P. Carvalho, A. P. Freire, "Native or Web-Hybrid Apps? An Analysis of the Adequacy for Accessibility of Android Interface Components Used with Screen Readers". In: Proceedings of the XVI Brazilian Symposium on Human Factors in Computing Systems, IHC '17, Joinville, SC, p. 362–371, 2017.

[9] V. Casadei, T. Granollers, and L. Zaina, "Investigating accessibility issues of UI mobile design patterns in online communities: a virtual ethnographic study", In: Proceedings of the XVI Brazilian Symposium on Human Factors in Computing Systems (IHC 2017). ACM, New York, NY, USA, Article 33, 10 pages, 2017.

[10] S. Chiti and B. Leporini, "Accessibility of Android-based Mobile Devices: A Prototype to Investigate Interaction with Blind Users", In: Proceedings of the 13th International Conference on Computers Helping People with Special Needs - Volume Part II, ICCHP'12, pages 607-614, Berlin, Heidelberg. Springer-Verlag, 2012.

[11] R. Clegg-Vinell, C. Bailey, and V. Gkatzidou, "Investigating the appropriateness and relevance of mobile web accessibility guidelines", In: Proceedings of the 11th Web for All Conference, W4A '14, pp. 38:1-38:4, New York, NY, USA. ACM, 2014.

[12] D. Costa, L. Carriço, C. and Duarte, "The Differences in Accessibility of TV and Desktop Web Applications from the Perspective of Automated Evaluation". Procedia Computer Science, vol. 67, pages 388–396, 2015.

[13] A. P. Freire, "Disabled people and the Web: User-based measurement of accessibility", PhD thesis, University of York, Department of Computer Science, Available online at http://etheses.whiterose.ac.uk/id/eprint/3873, last accessed on 30th January 2018, 2012.

[14] R. J. P. Damaceno, J. C. Braga, and J. P. M. Chalco, "Mobile Device Accessibility for the Visually Impaired: Problems Mapping and Empirical Study of Touch Screen Gestures", In: Proceedings of the 15th Brazilian Symposium on Human Factors in Computing Systems (IHC '16). ACM, New York, NY, USA, Article 2, 10 pages, 2016.

[15] International Standardisation Organisation, "ISO 9241-11: Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs) - Part 11: Guidance on Usability", 1998.

[16] International Standardisation Organisation, "ISO 9241-171: Ergonomics of Human-System Interaction - Part 171: Guidance on Software Accessibility", 2008.

[17] S. K. Kane, J. O. Wobbrock, and R. E. Ladner, "Usable gestures for blind people: Understanding preference and performance", In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '11, pp 413-422, New York, NY, USA, ACM, 2011.

[18] B. Leporini, M. C. Buzzi, and M. Buzzi, "Interacting with Mobile Devices via VoiceOver: Usability and Accessibility Issues", In: Proceedings of the 24th Australian Computer-Human Interaction Conference, OzCHI '12, pages 339-348, New York, NY, USA, ACM, 2012.

[19] I. Malavolta, S. Ruberto, T. Soru, and V. Terragni, "Hybrid Mobile Apps in the Google Play Store: An Exploratory Investigation". In: Proceedings of the Second ACM International Conference on Mobile Software Engineering and Systems, MOBILESoft '15, pages 56-59, Piscataway, NJ, USA, IEEE Press, 2015.

[20] K. Park, T. Goh, and H. J. So, "Toward Accessible Mobile Application Design: Developing Mobile Application Accessibility Guidelines for People with Visual Impairment", In: Proceedings of HCI Korea, HCIK '15, pages 31-38, South Korea. Hanbit Media, 2014.

[21] L. S. G. Piccolo, E. M. de Menezes, and B. de Campos Buccolo, "Developing an Accessible Interaction Model for Touch Screen Mobile Devices: Preliminary Results", In: Proceedings of the 10th Brazilian Symposium on Human Factors in Computing Systems and the 5th Latin American Conference on Human-Computer Interaction, IHC+CLIHC '11, pp. 222-226, Porto Alegre, Brazil, Brazilian Computer Society, 2011.

[22] C. Power, A. P. Freire, H. Petrie, and D. Swallow, "Guidelines Are Only Half of the Story: Accessibility Problems Encountered by Blind Users on the Web", In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '12, pages 433-442, New York, NY, USA, ACM, 2012.

[23] L. C. Serra, L. P. Carvalho, Ferreira, J. B. S. Vaz, and A. P. Freire, "Accessibility Evaluation of E-Government Mobile Applications in Brazil". Procedia Computer Science, volume 67, pp, 348 – 357, 2015.

[24] C. F. da Silva, S. B. L. Ferreira, and J. F. M. Ramos. "WhatsApp accessibility from the perspective of visually impaired people", In: Proceedings of the 15th Brazilian Symposium on Human Factors in Computing Systems (IHC '16). ACM, New York, NY, USA, Article 11, 10 pages, 2016.

[25] Statista. "Number of smartphone users worldwide from 2014 to 2020 (in billions)". Available online at https://www.statista.com/statistics/330695/number-of- smartphone-users-worldwide/, last accessed on 30th January 2017. 2016.

[26] Statista. "Market share held by mobile operating systems in Brazil from January 2012 to December 2016", Available online at https://www.statista.com/statistics/262167/market-share-held-by-mobile-operating-systems-in-brazil/, last accessed on 30th January 2018. 2016.

[27] M. Shitkova, J. Holler, T. Heide, N. Clever, and J. Becker, "Towards Usability Guidelines for Mobile Websites and Applications", In: Wirtschaftsinformatik, pg 1603-1617, 2015.

[28] C. Siebra, T. Gouveia, J. Macedo, W. Correia, M. Penha, M. Anjos, F. Florentin, F. Q. B. Silva and A. L. M. Santos, "Observation Based Analysis on the Use of Mobile Applications for Visually Impaired Users", In: Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct, MobileHCI '16, pages 807-814, New York, NY, USA, ACM, 2016.

[29] World Wide Web Consortium, "Web Accessibility Initiative (WAI)". Available online at https://www.w3.org/WAI/, last accessed on 30th January 2017, 2008.

[30] WebAIM - Web Accessibility in Mind, "Screen Reader User Survey #7 Results". Available online at https://webaim.org/projects/screenreadersurvey7, last accessed 5 July 2018.