Fouras

Automated Android Accessibility Assessment

Master's thesis in computer science and engineering

Hannes Häggander

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Master's thesis 2025

Fouras

Automated Android Accessibility Assessment

Hannes Häggander

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Fouras
Hannes Häggander

© Hannes Häggander, 2025.

Supervisor: Linda Erlenhov, Department of Computer Science and Engineering
Examiner: Gregory Gay, Department of Computer Science and Engineering

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2025

Hannes Häggander
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This thesis investigates systematic methods and tools applicable to improve the accessibility feature compliance testing of Android applications. Focusing on how software engineers can better implement design their applications to simplify testing and verify compliance with European accessibility standards, such as EN 301 549 [1] and WCAG 2.1 [2]. We aspired to assist developers and organizations in improving their Android application accessibility support and promote digital inclusivity for impaired users relying on accessibility features.

We examined the capabilities and limitations of current accessibility verification tools to suggest strategies to incorporate validation tests into the development workflow. We identify accessibility verification challenges and offer automated validation methods where applicable to reduce manual labor. Our research was performed with an iterative design-based approach, resulting in a developer-approved framework that offers systematic design approaches and automated tests to simplify the compliance process. The framework provides developers and organizations with solutions to gradually improve accessibility compliance.

To ensure industry relevance, we continuously obtained feedback from senior professional Android developers throughout the study to improve the implementation of accessibility features in the development process. According to prior research, Android accessibility features are rare in Android applications. With this study, we seek to change that and provide an equal application experience for all users.

Keywords: Android, application, mobile, development, accessibility, inclusion, automation, testing, verification, tooling

Acknowledgements

Thank you, supervisor Linda Erlenhov, for providing valuable guidance, feedback, and insights throughout this thesis. Linda's expertise in research design has been especially helpful.

Thank you, Gregory Gay, for your insights and academic assistance throughout the thesis process and for connecting me with supervisor Linda.

Thank you to all the interviewees who contributed valuable insights into the practical aspects of professional development, enhancing the industry relevance of this work.

Lastly, thank you to my wonderful family for supporting me and my efforts throughout this thesis.

Hannes Häggander, Gothenburg, July 2025

Contents

List of Figures **xiii**

1 Introduction **1**
1.1 Problem statement . . . 1
1.2 Purpose of the study . . . 2
1.3 Significance of the study . . . 2
1.4 Research questions . . . 3
1.4.1 Research question 1 (RQ1) . . . 3
1.4.2 Research question 2 (RQ2) . . . 4
1.4.3 Research question 3 (RQ3) . . . 4

2 Background **5**
2.1 Application accessibility problem . . . 5
2.2 Accessibility standards . . . 6
2.2.1 Web Content Accessibility Guidelines 2.1 . . . 6
2.2.2 EN 301 549 standard . . . 8
2.2.3 The European Accessibility Act . . . 8
2.2.4 Google Android guidelines . . . 9
2.3 Accessibility feature testing . . . 10
2.3.1 Manual accessibility testing . . . 10
2.3.2 Automated accessibility testing . . . 10

3 Related work **11**
3.1 Accessibility feature neglect . . . 11
3.2 Android accessibility challenges . . . 11
3.3 Improving accessibility adoption . . . 12
3.4 Automated accessibility detection . . . 12
3.5 Lack of accessibility testing tools . . . 13
3.6 Summary . . . 13

4 Method **15**
4.1 Problem . . . 15
4.2 Solution . . . 15
4.3 Evaluation . . . 16
4.4 Validation . . . 17
4.5 Research design, methods, and procedures . . . 17
4.5.1 Design science . . . 18

4.5.2 Insights to improve app accessibility 18
4.6 Thesis design science research 19
4.7 Semi-structured interviews 19
4.8 Thematic analysis 19

5 Findings 21
5.1 First cycle 21
5.1.1 Requirements 21
5.1.2 Problem 23
5.1.2.1 Public transport application 23
5.1.2.2 Popular music streaming application 24
5.1.2.3 Digital healthcare application 24
5.1.2.4 Our sample application 25
5.1.3 Solution 26
5.1.4 Validation 27
5.1.5 Findings from interviews 27
5.1.6 Implementation 29
5.1.6.1 Non-text content 30
5.1.6.2 Contrast (minimum) and Contrast (Enhanced) . . . 33
5.1.7 Evaluation 35
5.1.7.1 Automated accessibility validation using frameworks (RQ1) 35
5.1.7.2 Increasing accessibility with technical solutions (RQ2) 36
5.2 Second cycle 36
5.2.1 Problem 36
5.2.2 Workshop 37
5.2.3 Solution 38
5.2.3.1 Contrast checker 39
5.2.3.2 Screenshot testing exploration 44
5.2.3.3 Orientation validation 45
5.2.4 Validation 46
5.2.5 Evaluation 47
5.2.5.1 Promoting accessibility adoption by increasing developer awareness (RQ3) 47

6 Fouras framework 49
6.1 Systematic accessibility design 50
6.2 Accessibility feature testing 50
6.3 Compliance deviation detection 51
6.4 Expanding the framework 51
6.5 Fouras Android implementation 52

7 Discussion 53
7.1 Validating accessibility standards (RQ1) 53
7.2 Improving developer awareness (RQ2) 54
7.3 Promote accessibility adoption (RQ3) 55
7.4 Evaluating framework efficacy 55

7.5 Research implications . . . 56
7.6 Artificial intelligence assistance . . . 56
7.7 Future work . . . 57
7.7.1 Expand compliance tests suite . . . 57
7.7.2 Validate in production . . . 57
7.8 Threats to validity . . . 57
7.8.1 Construct validity . . . 58
7.8.2 Internal validity . . . 58
7.8.3 External validity . . . 59
7.8.4 Reliability . . . 59

8 Conclusion **61**

Bibliography **63**

A Appendix 1 **I**
A.1 Cycle 1 . . . I
A.1.1 Interview structure . . . I
A.1.2 Application layouts . . . III
A.2 Cycle 2 . . . X
A.2.1 Workshop structure . . . X
A.2.1.1 Introduction . . . X
A.2.1.2 Requirements (EN 301 549) . . . X
A.2.1.3 WCAG 2.1 . . . X

List of Figures

5.1 Visual representation of contrast ratios to white (#FFFFFF) text. . . 22
5.2 Västtrafik accessibility scanner result. 23
5.3 Spotify accessibility scanner result. 24
5.4 Kry accessibility scanner result. 25

A.1 Testing application authentication screen with credentials login IV
A.2 Testing application overview screen with preset products V
A.3 Testing application product screen showcasing one of the products . . VI
A.4 Testing application cart screen showing two items to be purchased . . VII
A.5 Testing application receipt screen, indicating a successful purchase . . VIII
A.6 Testing application profile view, allowing the user to logout IX

1
Introduction

1.1 Problem statement

Smartphone applications (apps) have transformed how we interact with services by making it easier to perform everyday tasks such as banking, identification, communication, and other utilities seamlessly using apps. However, applications are rarely accessible to impaired users who rely on accessibility features to use their smartphones. Application developers must be inclusive when they develop applications so that everyone, including those with impairments, may take part and have the opportunity to perform everyday tasks. To provide a universally accessible and inclusive user experience, app designers and developers should design and incorporate features that support and include users with specific needs from the early stages of app development.

According to the European Commission, one in five Europeans have disabilities [3]. Research on visual impairments [4] indicates that approximately 17% of the global population could benefit from enhanced visual accessibility features. The Android operating system developer parent company, Google, began an initiative called the Next Billion Users (NBU) [5], aiming to improve technological accessibility in developing nations. The NBU initiative seeks to be more inclusive and provide an inclusive digital environment [6].

Given the low adoption rate in applications, the availability of accessibility standards and verification tools does not seem to encourage developers to consider accessibility concerns. For example, Alshayban et al. found that, on average, 22.81% (with a standard deviation of 11.61%) of UI elements related to text contrast do not meet accessibility guidelines [7]. The general lack of accessibility support limits users who need to use applications, making it difficult or impossible for them to use the product. The European Commission seeks to remedy this issue by putting accessibility feature support into law by 2025 for smartphones [8].

Findings in previous research indicate that the reason for lacking accessibility features in applications is due to developers being unaware of accessibility features entirely and the accessibility standards [7]. This thesis seeks to address the possibilities of current accessibility validation tooling and custom tooling to improve developer awareness of the European Commission standards by providing systematic approaches and technical solutions to improve awareness by educating developers

by validating applications against European regulations and providing feedback to developers with the ambition of improving Android application accessibility support.

Our top priority is to improve applications and provide an inclusive experience to all users, which is the ethical way to approach development. Including more users in an application also offers an economic incentive, given that the user base will grow due to allowing more users to use the product.

1.2 Purpose of the study

We determined how developers use current accessibility validation tooling and created a framework for organizations to comply with European Union accessibility standard EN 301 549 [1]. We also explored how to improve developer accessibility feature awareness and minimize the manual labor process of verifying compliance by providing automated accessibility feature testing approaches in Android applications.

The outcome of our research is a framework that assists organizations with practical technical implementations and systematic approaches to comply with accessibility standards and requirements. We explored ways to improve developers' awareness of accessibility features by providing strategies and had experienced developers validate their usefulness. By using this framework, developers can iteratively implement the provided framework strategies for their existing Android projects, which will warn developers about missing accessibility implementations. Providing warnings will, in turn, improve developer awareness about accessibility feature support.

Implementing the framework will enhance accessibility and promote inclusivity in Android application development. By providing developers with systematic approaches and automated tooling that validates accessibility support, we reduce tedious manual labor and simplify the process of accessibility feature testing.

1.3 Significance of the study

Applications that offer services to the public sector must follow the European Commission standards in 2025. Improving developer knowledge about accessibility requirements via common warnings and failing automated tests where accessibility compliance fails during development contributes to improved developer awareness and improves the inclusive experience for impaired users. All developers should seek to better their applications to comply with accessibility expectations, allowing everyone to have an equivalent experience. We believe that in terms of economic factors supporting accessibility will, in theory, providing an inclusive product should also improve the economic aspect of any organization, given that additional customers can use the application.

Our research on accessibility assessments in mobile applications indicates a shortcoming in existing applications. Given that we have had challenges finding research on approaches to automate accessibility testing. Our intent with this thesis is to educate and provide insights to contribute to the automated accessibility research with our findings.

Our framework will primarily be useful to developers in implementing accessibility features by applying a systematic approach that complies with regulations and applying technical solutions when creating custom assessment tooling for their application purposes. It is difficult and time-consuming to retroactively explore every layout in an existing application manually; by using automated assessment tools, we estimate that the required effort needed to validate accessibility features throughout the application will decrease significantly.

As evident by prior research [7] [9], accessibility testing is lacking in Android application development. Our ambition is to create a framework that, when applied to the development process, will improve developer awareness and create a more inclusive application experience for users who need accessibility assistance. Technology should be available to everyone. Custom automated tests that validate accessibility throughout a changing code base will ensure that developers consider accessibility support throughout the development effort.

1.4 Research questions

1.4.1 Research question 1 (RQ1)

Which accessibility standards can we validate using automated tests for Android applications by using existing testing frameworks, such as Espresso User Interface Tests [10]?

Automated testing is a common practice in software development to verify functionality as the project evolves. This concept should ideally extend to accessibility testing, too. Accessibility tests should validate compliance to established standards from the European Commission from the World Wide Web Consortium (WCAG) [2] [1].

Creating unit tests to validate functionality in isolated environments is common in Android development, and we want to explore how unit testing can verify accessibility compliance with standards. However, accessibility support can involve user interfaces, and testing layout elements must behave in an expected way and provide expected properties; we want to explore the possibilities of verifying accessibility standards using this type of test.

1.4.2 Research question 2 (RQ2)

How can technical solutions contribute to improving Android developer awareness and adoption of accessibility features?

We explored current technological solutions that enhance developer awareness. We used current tools and other potential strategies that developers can use. For developers to recognize the importance of accessibility features in Android applications, they must understand what they are, how to implement them, and how they support users who rely on them. Our suggested improvements align with current development practices in Android application development. Our goal was to provide a framework that is familiar to developers when they initiate their accessibility support efforts in practice.

1.4.3 Research question 3 (RQ3)

To what extent can Android frameworks improve developer awareness and promote accessibility adoption?

Android frameworks seek to lower the initial knowledge requirement of accessibility features by providing tools and guidelines for developers to implement, allowing for quicker adoption of accessibility features. We investigate how such frameworks focused on accessibility features influence developers' adoption of accessibility features and by implementing features improving developers' awareness of accessibility features.

2

Background

2.1 Application accessibility problem

Accessibility features are tools within the mobile operating system that allow users to alter how they interact with the device to improve the user experience. For example, accessibility features allow users to change the default text font size to improve readability for visually impaired users [11]. For some users, these features are essential to use their device, such as blind users having the option to have the screen's contents read out to them using the TalkBack feature [12].

Smartphone applications have made everyday digital tasks more available and accessible to most users. However, using applications can be challenging for impaired users who rely upon the application supporting accessibility features, and many applications do not sufficiently support them. Providing accessibility features to users with disabilities, such as partial blindness, near-sightedness, or limited motor functions, will allow them to have an equal application experience. Ehrlich et al. [4] stated that the issue stems from Android application developers not considering accessibility during development. This oversight excludes impaired users from using the application effectively.

The low adoption rate of accessibility features in most applications [7] needs refactoring to the project to provide sufficient support to their users. Retroactively implementing accessibility features late during development will most likely require labor-intensive efforts. Planning for accessibility features by creating and implementing design systems compliant with regulations early on during development makes implementing accessibility features considerably easier than updating every layout element with an existing application. There is always a trade-off when allocating development resources, and allocating resources to provide proper support to those needing it should be prioritized.

Developers and organizations should address this oversight early on during the development cycle rather than later to avoid a scenario requiring much work. Society is becoming increasingly reliant on technical solutions to perform everyday tasks. Even ordering food at certain restaurants requires operating a smartphone. Being inclusive to all users is a matter of fairness, and by supporting accessibility features, all users have an equal opportunity to function in our increasingly technically dependent society.

For developers to create an application that suits their own needs as a user is inherently easier than understanding the needs of others. According to Ehrlich et al., [4] and Alshayban et al. [7], state that one of the predominant issues of accessibility adoption is that developers are unaware of accessibility features entirely. Assuming this is valid, developers do not consider accessibility features when developing applications, designing, or testing the application using accessibility evaluation tools. This results in disabled users not being supported when using the application.

In our research, we seek to identify problems in current accessibility feature validation and provide solutions to lessen the accessibility feature knowledge gap by providing insights to developers. We will gain insights by using existing accessibility validation frameworks, creating automated tests and tooling that identifies implementation compliance of accessibility features and standards by Google [13] and the World Wide Web Consortium [2] which is adopted by the Swedish agency for digital government [14].

2.2 Accessibility standards

The European Union, the Swedish governments, and the company developing the Android platform provide accessibility standards and guidelines. These standards and guidelines will form the basis of our framework, and our validation strategies will comply with them.

2.2.1 Web Content Accessibility Guidelines 2.1

The Web Content Accessibility Guidelines 2.1 standards cover four areas in terms of usability and accessibility needs. The web consortium has organized the standards into the following categories: perceivable, operable, understandable, and robust.

Perceivable
The standards emphasize the need for content to be perceivable for users, including adjustments specific to mobile usage. For the user experience to be perceivable with small screens, the information should be tailored for mobile use, for example, having fewer modules presented and images than an equivalent desktop website.

Input areas should be considered and usable for low-vision users without zooming in. However, users should have the option for additional zoom; users should be able to increase the size of the text up to 200%.

Smartphones and their applications must function in various levels of lighting exposure, including exposure to direct sunlight. This environmental factor requires the design of interface elements with high contrast ratios to be visible. They must ensure a clear contrast between interface components to support usability and accessibility, aligning with guidelines for visual presentation. These design considerations

are essential for maintaining readability and usability for users, especially those with low vision.

Operable

When smartphones moved away from physical keyboards in favor of touchscreen interfaces, the need for precise touch inputs became challenging for impaired users. WCAG 2.1 guidelines state that input areas for on-screen elements should have a minimum of 9 square millimeters to assist users who find it challenging to perform precise interactions.

On-screen interactions and gestures, such as long presses or double taps, can also be difficult for impaired users to perform. Best practices suggest that interactions should be simple to enhance usability. This guideline is essential for making accessibility features like the screen reader functionality known as TalkBack [12] easier to operate for those who rely on its functionality. By making interactions with the application more simple and accessible, designing is good for every user, regardless of their physical capabilities.

Better yet, enabling one-handed usability of the application by placing intractable elements to be easily accessed provides benefits to users with disabilities who cannot use both hands simultaneously. Simply positioning interactive components on the screen's lower sections allows the thumb to reach easily, making it easier for all users to interact with the app. Such design practices enhance usability for individuals with physical limitations and improve the overall user experience.

Understandable

Supporting multiple screen orientations, such as portrait and landscape mode, is essential when mounting devices to assistive equipment, such as wheelchairs. The assistive technologies must function seamlessly across all orientations.

Consistency in layout operational changes in the app is key for intuitive navigation. For example, suppose a layout displays items A, B, and C in a specific order. In that case, this order should remain consistent after the app's configuration changes, such as orientation changes. This principle should apply to all orientations. Consistent is particularly advantageous for users with low vision or cognitive challenges, as they may utilize zoomed-in or scaled-up interfaces, limiting the visible layout area.

Improvements for screen readers, such as TalkBack, will make the layout more understandable to the impaired user. For example, grouping multiple layout elements within a single container and having the container be interactable rather than the inner content of the container improves usability for individuals with dexterity challenges. Given that the input of the container provides a larger area rather than multiple small interactable elements. If they are separated elements, the screen reader will read out all of the elements and potentially confuse the user if the buttons read the same when the functionality is the same, making it seem like the elements read multiple times.

Marking interactive elements using contrasting colors and shapes representing intent or other visual cues is essential to making the layout understandable. Given the clear separation between layout elements, this differentiating design will benefit all users, especially the visually impaired, as it is easier to separate the elements.

Robust
When requesting information from users, offering the most appropriate input method for the specific input is beneficial. For example, display a numeric-only virtual keyboard for inputs requiring numerical input to minimize input errors and reduce confusion. The guidelines seek to limit invalid manual data entry by utilizing constrained input methods like menus, radio buttons, and checkboxes whenever applicable. These simple data input approaches will streamline the process and enhance the user experience by reducing faulty options and the likelihood of errors.

2.2.2 EN 301 549 standard

The European standard EN 301 549 [1] lists accessibility requirements for information and communication technology products and services, including 162 essential requirements for mobile applications. Similarly to the previously mentioned WCAG 2.1. guidelines, EN 301 549 standards groups the requirements into four categories: perceivable, operable, understandable, and robust. This standard refers to WCAG 2.1 several times but mandates achieving Level AA requirements rather than the minimum Level A requirements.

The standards also provide complementary directives on what the application needs to achieve. For example, users should be able to activate accessibility features without relying on others to do it for them; if a user cannot see, they should have the possibility to enable the TalkBack feature even when it is disabled. Other requirements emphasize providing alternative ways of interacting with devices. For instance, when using a mobile gesture like pinch-to-zoom, there must be an alternative method to achieve the same result as the pinch-to-zoom action.

Some requirements offer alternatives for specific implementation requirements. For example, when legally binding documents are to be signed, the user must either have the option to reverse the choice, check for errors before submitting, or confirm the action before submission.

2.2.3 The European Accessibility Act

Europe has decided on a directive to require accessibility support into law in all European Union member states [3]. The directive will require countries to put accessibility feature requirements for public products and services, such as public transport, banking services, and smartphones, to be accessible to individuals with disabilities into law by June 28, 2025. While there are no provided requirements or

details for compliance, the European Union allows countries to identify their implementation of the law. The Swedish Agency for Digital Government (DIGG) states that implementing WCAG 2.1 and EN 301 549 will be mandatory for organizations in Sweden [15].

The European Union estimates that at least 87 million people will benefit from accessibility features in digital applications, marking the significance of this directive. According to the European Accessibility Act Questions and Answers [3], mobile services are among the services and products covered by the Act. However, the standard requirements specifying the mobile application accessibility feature requirement and what needs to be supported are vague. The Act states:

> *The European Accessibility Act identifies the product features and service features that must be accessible for persons with disabilities. The Act uses functional EU accessibility requirements. It does not impose detailed technical restrictions to make products and services accessible. This allows room for innovation and flexibility*, European Commission

While the answer describes expectations such as alternatives to speech for operating a device, magnification options, and compatibility with assistive devices for web navigation, the requirements still need detailed technical definitions. This ambiguity leaves individual countries (and developers) responsible for determining how to meet these general requirements.

2.2.4 Google Android guidelines

Google offers many resources aimed at incorporating accessibility features for Android applications. These resources encompass broad principles and specific guidance tailored for Android application development [16]. In their instructional video, Make Your Android App More Accessible [17], they claim that the integration of accessibility features not only benefits users with disabilities but also enhances the overall user experience of the application. This material also states that key accessibility principles align with the Web Content Accessibility Guidelines (WCAG). Such as the importance of adapting to varying environmental brightness levels, verifying sufficient contrast ratios of elements [18], and providing sufficient input areas. However, it is uncertain if the accessibility suggestions comply with upcoming European regulations.

These guidelines provide implementation-specific details, as opposed to the European guidelines. For example, text smaller than 18pt in regular style or 14pt in bold must exhibit a minimum contrast ratio of 4.5:1, similar to Level AA compliance in WCAG 2.1. Developers are encouraged to utilize tools such as the Accessibility Scanner app [19] for verifying compliance with these color contrast ratios. Additionally, input fields must maintain a minimum size of 48 by 48 density-independent pixels and content descriptions to comply with other accessibility features such as

screen readers. These detailed implementations are helpful to developers, given that they give domain-specific advice rather than regulatory expectations.

2.3 Accessibility feature testing

We explored manual and automated accessibility testing for Android development. Manual testing involves using accessibility tools like TalkBack or the Accessibility Scanner to mimic user interactions to ensure that the implemented accessibility feature works as expected. Automated testing with frameworks like Espresso programmatically verifies accessibility features by setting up an environment, interacting with the layout, and verifying that the layout is in an expected state.

2.3.1 Manual accessibility testing

Google provides tools for testing accessibility features and guides on manually and automatically testing them. Manual testing is performed by enabling the accessibility services and trying the accessibility features like the screen reader TalkBack [12]. There are tools like the accessibility scanner that let developers know about the elements that lack accessibility support. The Accessibility Test Framework [20] is integrated into the development environment and provides feedback to the developer and suggestions on how to fix them.

When developers create an application and run a manual test using the accessibility scanner, they can create reports from the accessibility testing framework. The report covers the identified lacking accessibility features, such as sufficient input areas, elements contrast checks, and missing content labels.

2.3.2 Automated accessibility testing

The automated testing framework developed by Google, named Espresso [10], allows developers to configure automated tests that programmatically interact with user interfaces in Android applications. By enabling accessibility feature checks when developing the interface tests and interacting with the layouts using automated actions, the test will check for accessibility features while running. The accessibility features test is configurable to limit testing only to verify a subset of the accessibility checks to make it easier to gradually test the layouts rather than reporting on all of the errors at once. However, Google states that automated testing is unlikely to solve all of the accessibility problems that users face and pushes for manual testing by users who need the accessibility features to verify that they provide value for them [21].

3
Related work

This section summarizes our research findings on accessibility in Android applications and previous efforts to improve accessibility feature validation.

3.1 Accessibility feature neglect

Prior studies indicate that the vast majority of publicly available Android application projects do not support accessibility features [7] [9]. These studies indicate that applications that fail to implement accessibility features limit impaired users' ability to operate the application; for example, Vendome et al. [9] found that users failed to perform precision clicks on small layout elements on their mobile devices and that elderly users are especially susceptible to physical challenges when interacting with applications.

A 2020 study found that even applications with billions of downloads needed to improve their accessibility support, partly due to the same reason of increasing interaction areas of their interactable layout elements [22].

3.2 Android accessibility challenges

Research identified that one of the leading causes for the lack of accessibility feature adoption in Android applications is that developers are unaware of accessibility features and that the options for assessing accessibility compatibility through verification with tools or automated testing are limited. Developers encounter hurdles when making an effort to support accessibility features as a result of limited options of tools designed for accessibility assessment through testing.

Researchers claim that current accessibility testing tools for Android developers are lacking compared to tools available for other platforms, such as web development [23]. Arguing for more usable tooling that simplifies testing as current tooling for Android accessibility compliance requires manual labor.

Previous research efforts to address automated accessibility testing approaches resulted in a testing tool that randomly navigated an application for a set amount of time, evaluated and identified layouts lacking accessibility features, and then created

a report of the findings [24]. The tests identified errors by taking screenshots and analyzing the image for accessibility errors, such as color contrasts. The researchers stated that the lack of tooling is partly due to the lack of a clear definition of how to evaluate sufficient accessibility support, making it difficult to provide a unified framework that effectively evaluates the accessibility of an application.

Visually impaired users face additional challenges when encountering generated content, such as ads [25]. Ziyao et al. found that ads can cause issues for screen readers, such as Talkback used in Android, when layouts are not labeled and often require an unintuitive number of actions to remove. They compared several ad libraries and found that 84% of ads have accessibility issues when using these tools. Among the best-performing were ads for native Android applications when compared to others. They interviewed 15 blind users to gain insight into their experience navigating applications with ads. These users strongly disliked their experience of interacting with ads, often opting to purchase a premium version of the app that completely removed the ads. Researchers found that the visually impaired experienced significant improvement when using labeling and elements designed for mobile use.

3.3 Improving accessibility adoption

Researchers stress the importance of developers familiarizing themselves with accessibility features and then supporting them [7]. Another identified reason is that organizations neglect to allocate the necessary time and resources to implement accessibility support into their products. Another study advocates fixing these errors during the development cycle and ensuring that apps adhere to a minimum accessibility standard to provide an inclusive user experience for all [22]. Research shows that the most commonly supported accessibility features relate to assistance to visually impaired users [9] [7], also stating that the number of users needing these kinds of accessibility features is growing every year.

However, we found mixed approaches to improving developer awareness of accessibility features in prior research. Some researchers suggest that providing developers with hints and warnings during project development when accessibility features are unsupported improves the overall adoption rate [9]. Others argue that presenting developers with hints and warnings on accessibility improvements can overwhelm developers, discouraging them from supporting accessibility features [23].

3.4 Automated accessibility detection

In research on automating the detection of failing accessibility implementations, Abdulaziz Alshayban and Sam Malek created a screenshot tool to detect text scaling issues in Android applications [26]. By analysing complaints in application reviews and social media, they identified that some layouts become difficult or impossible

to use when changing the size of texts. Issues ranged from unresponsive layouts to missing, overlapping, cropped, or truncated views, resulting in users missing out on information that was accessible to those using the default text size. They concluded that integrating accessibility should be a consideration throughout the application development process to test for these types of issues.

On the opposite side of not providing accessibility to those who rely upon them is the problem of providing information or functions that are not available without accessibility features. Forough Mehralian et. al. created a detection tool called oversight [27] that identifies overly accessible layouts that could lead to security risks, where accessibility tools allow bad actors to bypass security features. Their research provides an approach to automate the process of analyzing application layout files for issues, such as accessibility tools that allow interactions with hidden or disabled elements that are not accessible without programmatically altering their state.

3.5 Lack of accessibility testing tools

For other platforms, such as web development, Mehan Tafreshipour et al. [28] researched the existing tools' ability to identify accessibility issues and created a framework that introduced bugs to websites by mutating the website, causing accessibility errors. They then assessed the performance of accessibility testing tools to verify compliance with the guidelines in WCAG 2.1 [2]. However, accessibility tools failed to identify half of the accessibility errors created by their framework, indicating that web accessibility testing tools are not consistently capable of identifying accessibility issues.

3.6 Summary

Several studies have identified the lack of accessibility feature adoption in Android applications and the lack of tooling for identifying and validating accessibility feature support in Android applications. Most Android applications, ranging from open-source projects to apps with over a billion downloads, face challenges supporting accessibility features. The need for accessibility features in Applications is growing, and supporting them would invite many users to use the applications. There have been efforts to automate the manual process of validating accessibility features. Developer awareness is potentially a major issue, but research needs to provide a clear path to improving it.

4
Method

This chapter describes our research procedures, providing an overview of the methodologies and techniques used. We applied the design science research guidelines following Knauss [29] thesis structure recommendations. This research structure is appealing to us as it provides a cycle-based product development approach suitable for iterative software development practices, and we validated our progress with feedback from industry professionals.

4.1 Problem

At the start of every development cycle, we investigated an identified problem related to each research question.

We identified which accessibility standards we can automate using user interface testing in Android to answer research question 1 concerning automated validation. Previous research identified the most common accessibility needs. With that in mind, we read all the 162 standards applicable to mobile applications in EN 301 549 [1] and WCAG [2] and determine which of the standards we identified to have potential to be validated using automated test validation. We also strived to be efficient with our limited time by providing automated solutions to accessibility standards that provided the most value, meaning that we prioritized implementing solutions for the most used accessibility features that also required low implementation effort.

For the second and third research questions regarding developer awareness and adoption of accessibility features, we gathered information and feedback from industry professionals to understand their experience and knowledge of accessibility. To understand how or if they support accessibility features in their applications and how viable they assessed our solutions to be in an industrial setting.

4.2 Solution

We examined and prioritized the EN 301 549 standards applicable to mobile applications by estimating each standard's value while keeping the standards' popularity, identified by prior research and data, in mind. Accessibility feature support like visual assistance is estimated to be the most dominant impairment that developers

discuss in online forums [9], and compared it to the expected development effort required to automate it. We explored how to automate a compliance validation solution and created a practical prototype implementation that verified the expected outcome. By creating a framework of solutions for validating the standards, we aimed to create a practical approach for developers and assist their accessibility compliance efforts. By exploring ways to systematically design or automate the process of ensuring layouts comply with the regulations to reduce the manual labor of current tooling.

We then decided which prioritized standards to focus on in the first cycle. Our selection was determined by our estimated time to develop a compliance validation solution. When two standards roughly provided the same value to the user, we prioritized the one we estimated to take less time to develop.

At the start of our first cycle we created an Android application to implement and test our solutions in an isolated environment, modify the business logic, and alter layouts to verify that our solutions provided valid feedback in different scenarios. This application would ensure that our framework provided valid feedback when it detected non-compliance with standards. The European Accessibility Act does not have strict technical requirements. However, the Swedish Agency for Digital Government states that mobile applications must comply with EN 301 549 by law [30]. We designed our framework to comply with EN 301 549 standards, which often refer to WCAG 2.1 [2], by creating automated or systematic solutions that verify the guidelines.

4.3 Evaluation

When we finished developing a standards-compliant solution to an identified problem, we turned to industry professionals, asking for their input or to evaluate the industrial viability of our solution and validate and finalize the cycle. All developers who participated had ten years or more of experience in Android development working at several companies.

In our first cycle, we conducted semi-structured interviews about developers' general understanding of accessibility features, new regulations, the state of their current projects' accessibility complaint validation processes, and potential improvements to further accessibility feature support in their application. Our questions to the participants, found in appendix A.1.1, focused on answering our second research question of understanding how technical solutions contribute to their current development efforts. We interviewed three developers with several years of experience in Android development experience of published applications. We invited many more developers to participate but either got no initial response or the correspondence stopped when attempting to schedule a time for an interview. We acknowledge that three participants are on the lower end of the desired number of participants we hoped to be a part of the interview. However, we found that the participants' answers provided

valuable insights into accessibility feature testing and allowed us to identify patterns of problems with accessibility feature development within a production environment.

At the end of our second cycle, we invited professionals to participate in a workshop where we presented our findings and solutions for improving accessibility during the development process. Throughout this workshop, we asked the participants to discuss and evaluate our findings by providing feedback regarding the solution's anticipated usefulness in their current development processes. Four developments participated, more than the interviews but less than we had hoped. However, only having four participants allowed the opportunity to have detailed discussions with them to share their development experiences, so we encouraged discussions about the presented solutions, where the participants had the opportunity to share their anticipated viability of the presented solutions in an application development process.

4.4 Validation

By speaking to industry professionals, we validated our solutions and assumptions about integrating our work into industry development processes. We questioned them about the viability of integrating our solutions into their development process. We continuously improved our solutions by listening to feedback regarding potential challenges, such as anticipated difficulties of implementing or validating, that the developers foresaw based on their experience within the industry. We deemed our results to be successful if the developers agreed that our solutions provided value and simplified their accessibility compliance efforts by using or taking inspiration from our provided solutions.

4.5 Research design, methods, and procedures

Gaining a well-informed understanding of current practices of accessibility testing will allow our framework to build upon up-to-date solutions to assess Android accessibility features. At this early stage, the objective is to gather information from research papers, conference papers, Accessibility standards requirements, and Google's official resources about the offered integration tooling. The standards that we will base the framework on are the Swedish government guidelines "Digital tillgänglighet," which adopts EN 301 549 and WCAG 2.1.

Throughout our research, we implemented our framework in a local Android application project to validate our findings and gain a practical understanding of what it is like to implement accessibility features in a project. We also developed automated approaches, such as user interface tests, to validate accessibility standards using available tools, and created custom tooling where available tools failed to solve our problem.

4.5.1 Design science

This research is performed with design science methodologies as we sought to contribute to a solution for a particular problem by using iterative development [31]. Runeson et al. [32] state that design science research should address real practice problems and understand the problem first and foremost rather than focusing heavily on a solution. Understanding and continuously improving is a well-suited strategy for resolving issues in software engineering. Iterative development is a natural process for software developers, given the standard continuous improvement practices within the industry. Performing design science research provides a structured approach to bridging the process of creating a solution for an identified industry problem and academic research using iterative development cycles [29].

Our focus will be on understanding accessibility needs, finding technical solutions for testing accessibility compliance, preferably by automated tests to verify continual compliance, and creating a collection of solutions combined into a framework that simplifies the regulation compliance verification process for organizations and Android developers who need to provide an equal opportunity to use the application regardless of ability. The framework will initially provide support to verify a limited number of EN 301 549 and World Wide Web Consortium (WCAG) 2.1 [33] standards and follow Google's guidelines [13] and then support more features with every iteration.

Professional developers are to be interviewed and provide feedback on our progress so that we may understand and accommodate their needs; by doing so, we address our second research question of how technical solutions may improve developer awareness of accessibility features. In every research cycle, we will improve the framework to provide a good developer experience and improve awareness about the need for accessibility testing through our feedback to developers through our implementations. We want a developer experience with minimal friction so that developers can easily get going testing their Android applications.

4.5.2 Insights to improve app accessibility

We combined our findings from developing the framework and interviewing developers to propose solutions that verify compliance of accessibility features during Android application development. Our approach seeks to automate as much as possible by structuring the layouts in a way that, by default, provides fewer options, but is inherently compliant and allows developers to change settings with intent for edge cases. We requested that developers evaluate our implementations in terms of how useful they found them and if they believed it would achieve our goal of improving developer awareness.

4.6 Thesis design science research

There is a disconnect between the goals of industry needs and academia regarding outcomes of thesis research [29]. Design science is a practical approach for thesis research that allows software solutions that resolve specific problems in addition to contributing to academic research. Knauss [29] has supervised multiple thesis projects and provides a framework composed of concrete advice based on insight and experience to perform research and solve problems with software successfully iteratively, satisfying industry and academia wants.

4.7 Semi-structured interviews

A semi-structured interview is a qualitative data-gathering research method that combines structured and unstructured questions. The interviewer follows a set of open-ended questions that allow flexible follow-up questions to explore interesting topics that arise when the interviewee has given their answer [34]. This data-gathering approach is practical in understanding the interviewee's perspective [35] while staying focused on one singular subject.

4.8 Thematic analysis

Braun and Clarke [36] provide a guide to applying thematic analysis in their research paper. Thematic analysis provides a flexible method for qualitative research by identifying and analyzing patterns or themes within a dataset. The purpose of this analysis is to organize and describe the data in detail by interpreting the responses, identifying commonalities, and uncovering themes within the data. Unlike other qualitative methods tied to specific theories, thematic analysis does not require a pre-existing theoretical framework. This approach suits our semi-structured approach, where points of interest were discussed with little structure regarding follow-up questions, as conversations differed based on responses.

The proposed process of conducting thematic analysis according to Braun and Clarke consists of six recursive phases:

1. Familiarize ourselves with the gathered data.
2. Identify labels for the response data (codes).
3. Determine themes by combining labels/codes into patterns.
4. Reviewing and refining the patterns to highlight insights and validate the themes.
5. Understand, define the themes, and ensure that they are unique and support the research questions.
6. Produce a report that highlights findings and their significance to the research topic.

5 Findings

This section is documentation of our work throughout the thesis.

5.1 First cycle

During the first cycle, we focused our efforts on the first research question regarding current frameworks (1.4.1) and the second research question about improving developer awareness (1.4.2). We identified what standards could be validated using existing accessibility validation tooling and interviewed industry professionals about their knowledge and exposure to accessibility features in general. Questions that we asked developers can be found in appendix A.1.1.

5.1.1 Requirements

Given that all requirements hold equal importance, we read the EN 301 549 requirements and selected a subset to start the development of our framework that verifies that the application supported the requirements. We continuously added more requirements throughout our development cycle iterations to expand support of more requirements and to have the opportunity for feedback on the current offering at the end of every cycle. We selected the requirements in part due to their popularity within prior research and our subjective expectations of ease of implementation founded on our experience with Android development [7] [9]. These are the requirements that we decided to start with:

1. Non-text content
EN 301 549 11.1.1.1.1
WCAG 2.1 1.1.1 Non-text content (Level A)
"All non-text content that is presented to the user has a text alternative that serves the equivalent purpose, except for the situations listed below. Controls, Input, Time-Based Media, Test, Sensory, CAPTCHA, Decoration, Formatting, Invisible", WCAG 2.1

2. Contrast (minimum)
EN 301 549 10.1.4.3
WCAG 2.1 1.4.3 (Level AA)
"The visual presentation of text and images of text has a contrast ratio of at

least 4.5:1, except for the following: large text, incidental, logotypes", WCAG 2.1

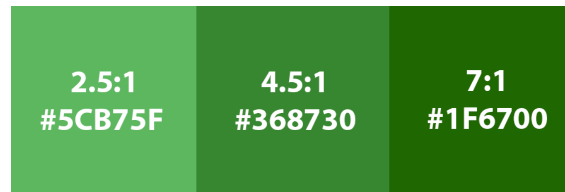

Figure 5.1: Visual representation of contrast ratios to white (#FFFFFF) text.

We explored how current tooling supports accessibility guideline validation and how developers are informed when accessibility compliance errors occur. Using our custom Android application, we created and tested layouts with EN 301 549 guideline-compliant and non-compliant layouts and then used Google's accessibility scanner tool to the layouts, verifying that the result reflected the compliance requirements of EN 301 549. Our application provided an isolated environment where we had control over the layouts, allowing us to modify the layouts to try different layout configurations.

The application we created contained a product purchasing flow. The layouts used different layout elements, like input fields, images of differing sizes, and interactable elements of different colors and input areas. The initial implementation was compliant with EN 301 549. Then, we modified the layouts to become increasingly non-compliant and investigated at what point the accessibility scanner reported the errors. The first version of the application included the following layouts.

1. Authentication screen inputs for username, password, and an intractable login button. With this layout, we tested the validation of supporting scalable texts in containers and provided a sufficient area for large text input fields for users to interact with. See figure in appendix A.1

2. Product overview screen with several listed items in a vertical grid structure. The product page also contains a small intractable button that leads to the user profile page. See figure in appendix A.2

3. Cart view where products are presented in a list with additional detailed information and a total sum of all the products currently in the cart. See figure A.4

4. Receipt view indicating the finalization of a purchase, letting the user know the completed order, and a button navigating back to the product overview. See figure A.5

5. Profile page where the user can return to the authentication screen. See figure A.6

5.1.2 Problem

We used the accessibility validation tool Google Accessibility Scanner on public Swedish applications to examine if findings from prior research on large applications lacking accessibility feature support also applied to the Swedish market. We selected a small sample of one regional, national, and global application to represent their respective reach. The orange boxes shown in the provided figures for each example indicate errors found by the accessibility scanner.

5.1.2.1 Public transport application

When we used the accessibility scanner on Gothenburg's regional public transport application, the scanner identified several errors and warnings on every screen. The main issues were contrast problems, scalable text within fixed layout containers that can lead to the text exceeding the size of the container, and interaction buttons sharing the exact description, which may cause problems for accessibility features like TalkBack that rely on descriptions that differentiate actionable items.

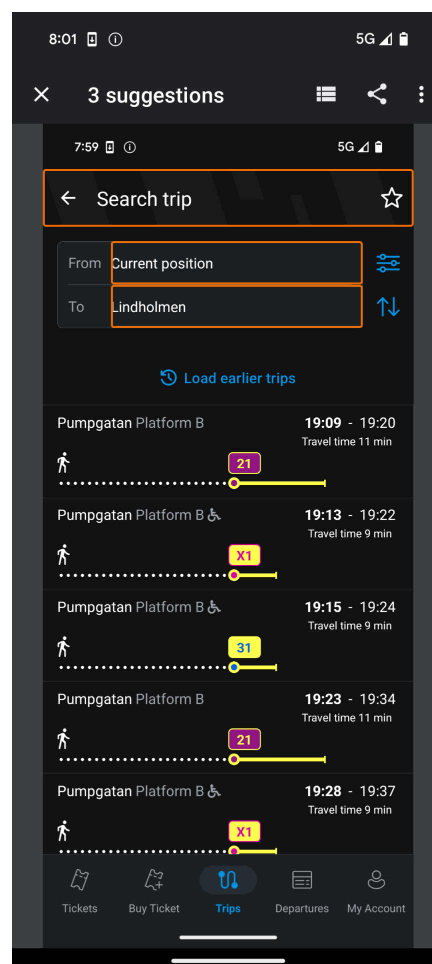

Figure 5.2: Västtrafik accessibility scanner result.

5.1.2.2 Popular music streaming application

Using the scanner on a massively popular music streaming application, Spotify's search screen, almost every layout element is identified to suffer from some accessibility errors, according to the result. According to the results, errors such as insufficient contrasts on generated content and small input areas are prevalent. These results indicate that even larger applications with large amounts of resources seem to have neglected accessibility requirements, at least according to the results from the accessibility scanner.

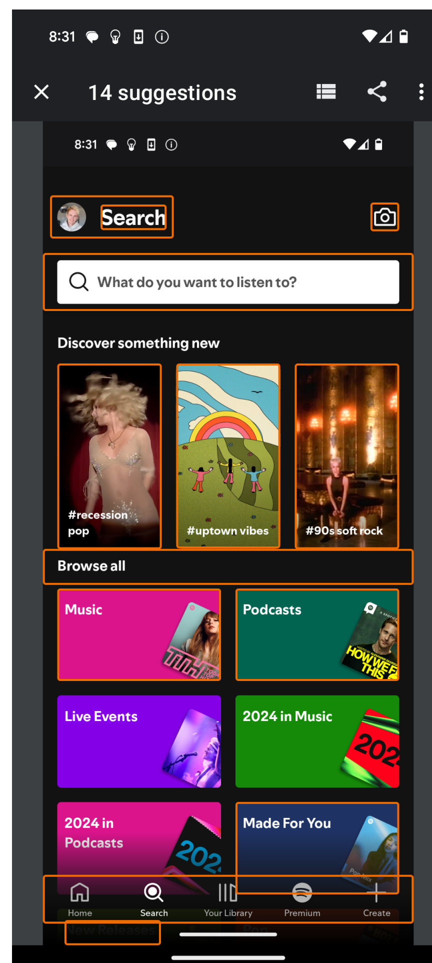

Figure 5.3: Spotify accessibility scanner result.

5.1.2.3 Digital healthcare application

According to the accessibility scanner results, the national digital health care application Kry also fails to address accessibility needs. Given the importance of health care, we believe that this type of government-subsidized application would be required to achieve accessibility needs, as the government regulates accessibility requirements of public applications. This application is of extra importance when considering that the users of this application are assumingly in need of medical attention, albeit non-emergencies.

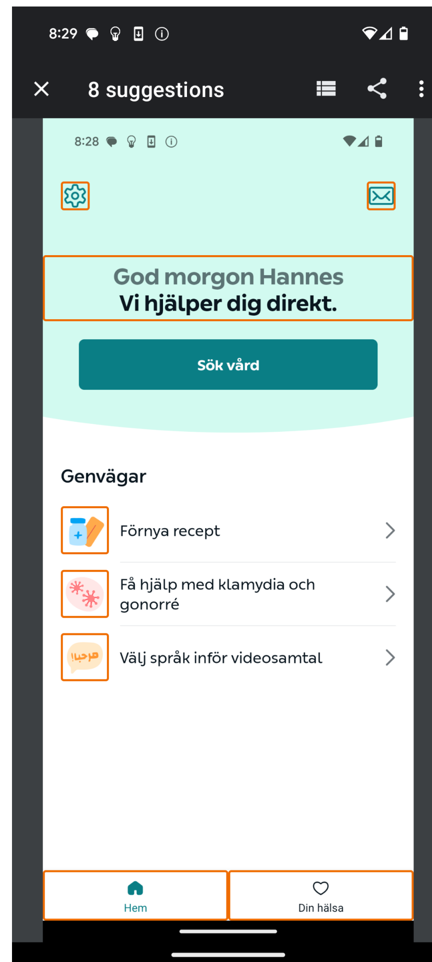

Figure 5.4: Kry accessibility scanner result.

5.1.2.4 Our sample application

Google's Accessibility Scanner tool identified missing image labels for image icons, such as the profile navigation button on the product overview page and the back navigation button on the profile page. However, the scanner did not identify missing labels of product images. Standards state that providing an image description is required for non-decorative images. However, given the intent uncertainty, we expect the tool to identify this as a potential error. There is a need to identify missing labels for all images with unknown informative or decorative intent. We decided to address this issue in our framework and identify empty descriptions by using static code analysis tools such as Lint to inform the developer about missing labels in layouts unless images are explicitly configured to be decorative or informative.

The TalkBack feature worked as expected in our demo application without additional development effort. However, the features and scope of our application layouts are simpler than those found in large-scale applications, making it easier for TalkBack to operate correctly.

We browsed for developer tools when we developed our application in the integrated development environment, Android Studio. We found the UI (User Interface) Check Mode [37] analysis tool feature. This tool allowed us to see several common screen sizes within the editor and provided information about the lack of accessibility support without running the application. This feature is similar to what we planned to implement in an automated testing environment. However, the tool only runs in a Compose Preview environment within the editor and not in a testing environment. When we applied this tool to our project, the UI checker identified contrast and size issues for buttons that were too wide on larger screens, such as tablet devices. Even though we could not get this functionality to report errors in a testing environment, we found this tool helpful, albeit with additional manual labor, for the developer to perform a preliminary check in isolation when constructing layouts.

Unfortunately, the automated Espresso test library alternative, AccessibilityChecks [38], did not perform feedback similar to the UI Check Mode. We could not identify how and what accessibility implementations the AccessibilityChecks library validated against. Ideally, we wanted to configure the checks to validate compliance with European standards.

5.1.3 Solution

Our approach to increase Android developer awareness was to give feedback when the implementation was not compliant with European standards. We found that the result of applying the Accessibility Scanner to our application was insufficient to accomplish our desired result. Our framework needed to provide useful and accurate feedback to the developer regarding what the framework was verifying. We also needed to avoid false positives that give developers a false confirmation of accessibility compliance.

For example, according to Android learning material provided by the Google Android team [39], the accessibility scanner is to identify color contrasts of 4.5:1, just like the requirements for WCAG 2.1, but when we applied the accessibility scanner to a layout with the foreground button color (#C1E8FF) and the background color (#F2FBFF), the contrast ratio was 1.23:1, it passed the test even though this is not compliant with WCAG [2] expectations of 4.5:1 minimum color-to-contrast ratio. These tools give developers a false sense of achieving compliance when several faults are present in the layout.

We also wanted to reduce the manual labor required when verifying compliance using accessibility tooling. For example, testing with the Accessibility Scanner is performed by manually opening the application under test with the Accessibility Scanner enabled and navigating through the application the developer wants to evaluate. This manual testing process is tedious as a part of the development process when repeated for every update to the layouts. We explored automated systems

by creating design requirements and implementations, allowing automated testing of validating compliance by verifying against the systems we created.

5.1.4 Validation

We believe that reducing the burden of manual labor is essential for developers to validate layout changes continuously. We aimed to create systematic solutions that assist the developer with accessibility compliance using automated instrumentation testing. Having experienced available accessibility compliance tools, we identified a few possibilities for automating the testing process. Automating the testing will allow developers to get warnings when changes quickly cause layouts to fail compliance.

We aimed to make our framework appealing to developers with automation and a design system approach in mind. To understand developer needs, we interviewed several senior Android developers for their general knowledge of accessibility features and their opinions on what type of feedback and configuration options developers want for this framework—allowing developers to configure what kinds of compliance tests to apply and how to get informed when tests fail.

5.1.5 Findings from interviews

We conducted several semi-structured interviews with professional developers to ask about their knowledge of upcoming European accessibility requirements, existing standards, experiences with accessibility features, and usage of accessibility testing tools. Our goal was to gain insights into their general understanding of accessibility, how the developers implement new testing processes, and to find potential solutions for automating the accessibility verification process with which the developers had first-hand experience.

Table 5.1: Overview of interview participant's experiences with accessibility features and understanding of upcoming regulations.

Experience	Accessibility experience	Regulation aware
Android Developer - 13 years	Yes	No
Android Developer - 12 years	Yes	No
Android Developer - 8 years	No	No

Details of the interview questions are provided in Appendix (A.1.1). We constructed the questions to be open to allow developers to share their experiences. We asked follow-up questions when the developers mentioned interesting matters, making for a deeper exploration of relevant topics to our research.

Using the thematic analysis process detailed in section 4.8 to identify patterns and themes based on the data we collected. By familiarizing ourselves with the transcripts of the interviews we identified that developers are unaware of the European Union Accessibility Act, unaware of what is required to comply with guidelines, awareness often comes from practical experience at work, some knows of the features but has no practical first hand experience, and that staying updated on accessibility advancements has not been a priority. We also identified that developers primarily implement accessibility when management or marketing requires it to promote the application or when achieving certifications is necessary, such as Google's requirements. However, if there are no requirements to support accessibility, developers rarely take it upon themselves to implement them. Accessibility implementations are rare for designers and developers to include in their development practices, leading to the fact that when sudden requirements arise, there is a need to update the application, which is challenging retroactively. Testing that the application is compliant with accessibility needs is often a manual task performed by developers and quality assurance personnel, rather than being part of the automated testing process. However, there is a general desire among developers to automate the process and add static code analysis, such as lint checks. Identifying issues early is welcome, but requiring fixes until they are resolved and preventing the process from continuing is not.

We identified five themes from these points:

1. Limited developer awareness: Developers are often unaware of requirements and guidelines. Awareness comes from practice, not a prioritized skill.
2. External requirement-driven development: Developers implement accessibility features when required to do so. If there are no requirements, it is rare for developers to implement accessibility features on their initiative.
3. Reactive accessibility implementation over intentional process: Accessibility compliance is not a part of the development process, but instead implemented reactively when there are sudden requirements to comply with.
4. Primarily manual testing: Verifying applications' compliance with accessibility requires manual testing and not automated approaches to validate implementations.
5. Desire for integrating accessibility compliance tooling: There is a desire to automate the process and integrate tooling that provides warnings, rather than prohibiting development.

As anticipated from researching prior studies, the interviewees had limited knowledge of accessibility features and how to test them [7] [9]. The interviewees mentioned that the most common reason for them to learn about accessibility features was a result of internal company pressures, such as failing certification tests or management requirements, rather than from their exploration of accessibility.

All the participants knew about the new European accessibility requirements, but none knew of the criteria for accessibility compliance. One interviewee noted that

when the General Data Protection Regulation (GDPR) became mandatory a few years back, their management gave short notice to make significant changes to the project to comply with the regulation. The interviewee thought the European Accessibility Act (EAA) requirements would be handled similarly at their current company. Making for a stressful time for developers and providing them with good tools would greatly help. As mentioned previously, the motivation for adopting accessibility seemed to stem from management and design mandates, not from the participant's initiatives, resulting in unprepared developers.

One interviewee had utilized Google's Accessibility Scanner on a production application and discovered errors. The Accessibility Scanner found issues and notified the designers, who then corrected the design and prioritized the required changes for the developers to implement. During that time, the interviewee held frequent sessions with impaired users using the accessibility features. They found value in getting feedback from users who frequently use accessibility features to validate their implementation. According to the interviewee, this approach was very effective and highly recommended. We believe this is an excellent method to verify that accessibility features effectively support the intended users.

Another interviewee had experience with an automated monkey tester. A monkey tester is a tool that interacts with the layout randomly or follows predetermined coordinates on the device to check for the presence of accessibility features [40]. This accessibility requirement was part of the requirements to pass a test suite for the *Made for Google* [41] certification program for compatibility with Google devices. The interviewee said that most identified errors were related to missing image descriptions and missing labels for icons. However, the participants' most frequently mentioned testing methods were manual testing by designers and quality assurance, not automated approaches. The interviewees agreed that automating the verification process would be beneficial, providing developers instant feedback.

All interviewees favored lint-style warnings over strict errors when receiving feedback. However, they believed that once accessibility regulations took effect, receiving errors for non-compliant layouts would be beneficial, given the consequences of not complying with regulations. The interviewees noted that strict testing would likely cause developers to implement temporary fixes or workarounds to avoid errors, such as adding empty or non-descriptive descriptions to pass tests. They felt enforcing accessibility features would remain difficult until legal requirements are in effect, forcing the developers to comply with regulations.

5.1.6 Implementation

By applying the data points we collected from the interviews and having experienced the current tooling, Google Accessibility Scanner, TalkBack, and the standard automated accessibility testing approach, we determined the subset of requirements that would become the framework. We also wanted to create solutions that address

the issues that we identified with the Accessibility Scanner.

5.1.6.1 Non-text content

EN 301 549 11.1.1.1.1
WCAG 2.1 1.1.1 Non-text content (Level A)

For our systematic approach to validate this standard, we provided an implementation requiring developers to choose whether the image is informative or decorative. Depending on the choice, the developer must set content descriptions to informative images or ignore content descriptions when the image is decorative. There are exceptions where the requirement does not apply, such as when the non-text content is a verification test where a description would invalidate the implementation, like a captcha test where parsing information from the image is the test. An example of this implementation is found in figure 5.1.6.1

```
object Image {
    @Composable
    fun Decorative(
        painter: Painter,
        modifier: Modifier = Modifier,
        alignment: Alignment = Alignment.Center,
        contentScale: ContentScale = ContentScale.Fit,
        alpha: Float = DefaultAlpha,
        colorFilter: ColorFilter? = null,
    ) {
        androidx.compose.foundation.Image(
            painter = painter,
            contentDescription = null,
            modifier = modifier.semantics {
                testTag = "image.decorative"
                role = androidx.compose.ui.semantics.Role.Image
            },
            alignment = alignment,
            contentScale = contentScale,
            alpha = alpha,
            colorFilter = colorFilter,
        )
    }

    @Composable
    fun Informative(
        painter: Painter,
        @StringRes contentDescriptionRes: Int,
        modifier: Modifier = Modifier,
        alignment: Alignment = Alignment.Center,
```

```
        contentScale: ContentScale = ContentScale.Fit,
        alpha: Float = DefaultAlpha,
        colorFilter: ColorFilter? = null,
    ) {
        val contentDescription = stringResource(contentDescriptionRes)

        require(contentDescription.isNotBlank()) {
            "Content description resource value cannot be blank"
        }

        androidx.compose.foundation.Image(
            painter = painter,
            contentDescription = contentDescription,
            modifier = modifier.semantics {
                testTag = "image.informative.{$contentDescription}"
                role = androidx.compose.ui.semantics.Role.Image
            },
            alignment = alignment,
            contentScale = contentScale,
            alpha = alpha,
            colorFilter = colorFilter,
        )
    }
}
```

This implementation is a wrapper of the material design image Composable implementation as provided by the Android framework. This implementation forces developers to implement the correct accessibility implementation and comply with requirements. One developer noted during the interview that the material design system should support accessibility requirements by design, such as contrasting colors compliance. The same developer also had experience implementing accessibility features in a production environment, and designers were the ones who decided on the priority. If alternatives for images are a part of the design specification, it will further the accessibility efforts developers will be required to implement the correct layout and comply with regulations by default. Another developer mentioned during their interview that, given the chance, developers will attempt to find tricks to make the implementation as uncomplicated as possible. With this opinionated approach, the content description must be a translatable local resource and not be allowed to be empty, solving two of the workaround tricks that the developer experienced when testing the accessibility features in the past.

The semantic tags are added to the implementation to allow for detection in automated testing. By providing the test tags, we can search for all layout nodes, identify images in the view that do not implement these tags, and offer the option to fail the test should any other type of image implementation not follow these opinionated image implementations.

5.1.6.2 Contrast (minimum) and Contrast (Enhanced)

EN 301 549 10.1.4.3
WCAG 2.1 1.4.3 (Level AA)

Contrast (Enhanced)
WCAG 2.1 1.4.6 (Level AAA)

Color systems like Material Design [42] do allow designers to provide background colors and text colors that provide sufficient contrast to one another. Implementing this kind of color system where the colors have an accompanying color makes complying with color contrast requirements an easier task for developers to implement the correct color, leaving little reason for automating tests to verify the colors. Given that there are exceptions to this requirement where decorative elements that incidentally do not provide an accepted color contrast are allowed, it should not be an issue.

There are options to add opinionated custom text implementation with the correct configuration, similar to implementing accessible images in the previous requirement (5.1.6.1). These opinionated implementations force developers to comply with regulations but limit their flexibility.

```
@Composable
fun ContrastText(
    text: String,
    modifier: Modifier = Modifier,
    fontSize: TextUnit = TextUnit.Unspecified,
    contrastType: ContrastType,
    fontStyle: FontStyle? = null,
    fontWeight: FontWeight? = null,
    fontFamily: FontFamily? = null,
    letterSpacing: TextUnit = TextUnit.Unspecified,
    textDecoration: TextDecoration? = null,
    textAlign: TextAlign? = null,
    lineHeight: TextUnit = TextUnit.Unspecified,
    overflow: TextOverflow = TextOverflow.Clip,
    softWrap: Boolean = true,
    maxLines: Int = Int.MAX_VALUE,
    minLines: Int = 1,
    onTextLayout: ((TextLayoutResult) -> Unit)? = null,
    style: TextStyle = LocalTextStyle.current
) {
    Text(
        text = text,
        modifier = modifier.then(Modifier.semantics {
            testTag = "text.${contrastType.name}.{$text}"
        }),
        color = with(MaterialTheme.colorScheme) {
```

```
            when (contrastType) {
                ContrastType.Primary -> onPrimary
                ContrastType.Secondary -> onSecondary
                ContrastType.Tertiary -> onTertiary
                ContrastType.Surface -> onSurface
                ContrastType.Background -> onBackground
            }
        },
        fontSize = fontSize,
        fontStyle = fontStyle,
        fontWeight = fontWeight,
        fontFamily = fontFamily,
        letterSpacing = letterSpacing,
        textDecoration = textDecoration,
        textAlign = textAlign,
        lineHeight = lineHeight,
        overflow = overflow,
        softWrap = softWrap,
        maxLines = maxLines,
        minLines = minLines,
        onTextLayout = onTextLayout,
        style = style,
    )
}

enum class ContrastType {
    Primary,
    Secondary,
    Tertiary,
    Surface,
    Background,
}
```

This approach does not allow the developer to configure the text to any color and instead specify what material design color to use. This forced implementation will require a compliant configuration of the underlying container.

An alternative approach to validate that the contrast is valid for text fields is to create a lint rule that performs a static code analysis to determine that the contrast ratio complies with a sufficient colors. We gathered that developers prefer lint warnings to be preferable from the interviews. However, we found this type of implementation to be complicated to implement and the provided implementation we propose will ensure usage of material design and in turn with the standard where a lint solution could be non-compliant.

5.1.7 Evaluation

We found it challenging to understand what the tools were verifying against when using the automated testing tools. There are limited configuration options available to developers to fit their needs. The option offered to the tester is to include or exclude an entire layout. This lack of configuration and transparency makes it hard to operate these tools to any benefit regarding verifying layouts with EN 301 549 standards.

The developer interviews on accessibility features summarize that automated processes for accessibility compliance are not as prevalent in practice as expected. Automated tests are standard within the software engineering industry. We find it odd that accessibility tests are treated differently than other areas of application development. Almost all interviewees used or had their designers and quality assurance teams use manual testing tools to verify accessibility features. With our attempts to create systems and automated verification tests, we seek to minimize the time required for manual testing.

5.1.7.1 Automated accessibility validation using frameworks (RQ1)

Verifying compliance with EN 301 549 and WCAG 2.1 using the available tools requires manual labor. We identified that current automated accessibility checks lack configuration options for developers to customize their testing strategy. It is also challenging to determine if the layouts are passing or failing the requirements set by the European Union by only applying the automated tools.

Verifying accessibility functionality is not an intended use case for automated testing Espresso frameworks. Espresso tests intended use case is confirming that layout elements are interactable and display the correct information in specific scenarios. Our attempts to validate accessibility standards using these tools have been inadequate, which resulted in us pivoting to design system compliance tests and opinionated implementation layout elements to enforce regulation-compliant implementations in combination with lint warnings when straying away from the intended design system.

Establishing a comprehensive design system is more reliable to ensure accessibility compliance than creating automated tests. Implementing compliant layout components combined with lint warnings to guide developers to compliance with the design system is practical. An added benefit of Lint warnings is allowing developers to customize when to address or ignore warnings and errors. Currently, involving manual testing and validation, preferably by including impaired users who regularly use and depend on accessibility features, seems to be the most effective approach to validating an application's accessibility.

5.1.7.2 Increasing accessibility with technical solutions (RQ2)

We identified a few patterns in the interview results:

- Developers typically avoid accessibility features unless required by designers or management.

- All interviewees preferred lint-style warnings when discussing compliance requirements due to the flexibility. They explained that strict enforcement might lead developers to circumvent tests—such as adding empty labels to descriptions of informative images if there is no requirement to ensure these features function correctly.

- Developers who had implemented accessibility features in previous roles did not continue with the practice when changing to a new position where it was not mandatory.

As discussed in our findings to Research Question 1, we recommend creating a design system that enforces accessibility standards by creating custom components and providing static analysis lint warnings when developers use non-compliant implementations. This approach will ensure developers' awareness of correctly implementing the accessibility features.

Ultimately, the implementation of accessibility features is a team effort. All team members should share responsibility for understanding accessibility requirements. A comprehensive design system will bridge the concept and implementation. Using a unified design framework that inherently complies with accessibility standards. With a carefully considered design system, we can improve developer accessibility awareness and adoption rates, as developers are informed and required to implement according to the design system.

5.2 Second cycle

This cycle was built upon the findings from the first cycle. Our goal was to address the third research question (1.4.3) by exploring how an Android framework could enhance developer awareness of accessibility features and encourage developers to include them to the development process.

5.2.1 Problem

To address the challenges of low awareness and adoption of accessibility features in Android applications, we wanted to find a way to continuously inform developers about accessibility compliance throughout the development by validating the application using technical solutions. We also needed to expand our framework to

accommodate the changes and ask developers if they would consider our approach in an industry environment. Our reason for involving developers in the research process was to present our technical solutions to them and verify that we created a framework developers would find useful. We talked to developers and asked them to evaluate our framework and provide ideas for improvement to our solutions, allowing us to refine our framework to improve the developer experience.

During the interviews, we found that developers generally only had a surface-level understanding of the accessibility requirements set to become law in 2025. We introduced the developers to EN 301 549 and WCAG 2.1 and showcased our proposed solutions to validate compliance. We showed the developers an implementation and provided a practical use case for integrating feature accessibility validation. We wanted to encourage validating accessibility features during development and considering how accessibility feature requirements apply to their applications.

5.2.2 Workshop

We organized a workshop with several experienced Android developers to share their insights into the European Accessibility Act (EAA) requirements that will apply to Android applications. We gathered information about where they stand with their efforts and discussed the challenges they encountered in complying with the requirements. We wanted to understand how they approached accessibility compliance in general when developing industry applications. We got feedback and insights into the status and needs of the industry. Then, we incorporated their feedback to refine our framework and accommodate developer needs.

For this study, our criteria for a senior Android developer is defined as a developer with a decade or more of Android development experience and holds architectural responsibilities for their respective projects. We defined an Android developer as having 2 to 9 years of experience and fewer responsibilities than senior developers.

Table 5.2: Overview of workshop participants' requirement to support accessibility features and current progress of European Accessibility Act (EAA) compliance.

Current role	Accessibility required	EAA compliance status
Senior Android developer	No	Initiated
Senior Android developer	No	No
Senior Android developer	No	No
Android developer	No	No

In an ideal situation, we would have had a larger sample size, allowing us to obtain more developer experiences and feedback. Three of the four participants were interviewed individually in the last cycle, but this workshop enabled them to share and discuss their feedback with each other. However, we found that only one of the

four participants had initiated their compliance efforts despite having previously encountered accessibility requirements in their careers. Despite prior experience, accessibility is often seemingly neglected. Three out of the four participants worked at the same organization but in different departments. Interestingly, they did not share requirements placed upon them regarding accessibility, as one had initiated the process of compliance while the other had not.

The developers appreciated our custom accessibility-compliant layout elements approach to make compliant integration across the application easier. Participants mentioned that this method allows for the iterative implementation of accessibility features, which aligns with our goal of achieving accessibility compliance through common developer practices of iterative improvement. By validating individual layout elements and enabling developers to integrate our components as needed, we can validate for compliance without testing the entire screen simultaneously.

One developer mentioned that their company had contacted DIGG regarding web application accessibility compliance but had yet to ask DIGG to review the mobile application. The developers expressed interest in implementing our layout elements into their application and validating them by DIGG. Another developer highlighted the challenges of maintaining opinionated accessibility layout containers that replace common implementations, suggesting that lint checks for non-compliant contrast may be more practical. We implemented such lint checks and evaluated the container maintenance issue for a better solution. We want to create approaches that enforce compliance without adding extra work and limiting design options.

One of the developers pointed out that screenshot testing could apply to testing contrasts of items within the layout; we explored this further and determined that screenshot testing will work in some cases but cannot ensure contrast compliance to the degree that we find acceptable.

5.2.3 Solution

Interviews revealed that developers were unlikely to consider accessibility support unless it was a requirement specified by other parties, such as the design department and management. We created automated checks that validated the compliance of the material design system with accessibility standards, for additional detail, see the implementation section. These automated tests will serve as a practical starting point for developers and designers to simplify the process of complying with the upcoming regulations by design. Ideally, these automated tests will always succeed in a correctly implemented material design system, but we want to provide automated checks for when the application inevitably changes.

With feedback from the workshop session, we refined the existing framework. We expanded the accessibility features, focusing on implementing support to validate more requirements. We also explored additional test strategies to validate layout

compliance programmatically.

5.2.3.1 Contrast checker

Workshop participants indicated that manually implementing and maintaining contrast compliance checks required much manual labor, and automating the process could significantly streamline development. We created an example of a user interface test that validates compliance with EN 301 549 contrast requirements, ensuring all material design system colors meet the required contrast ratio of 4.5:1 (and 7:1 for AAA-level compliance).

```
@RunWith(AndroidJUnit4::class)
class ColorSchemeContrastCheck {

    companion object {
        private const val MIN_CONTRAST_RATIO = 4.5
        private const val ENHANCED_CONTRAST_RATIO = 7.0
    }

    @get:Rule
    val composeTestRule = createComposeRule()

    @Test
    fun lightModeEN301439ContrastCheck() {
        composeTestRule.setContent {
            FourasTheme(darkTheme = false) {
                RunContrastChecks(isDarkMode = false)
            }
        }
    }

    @Test
    fun darkModeEN301439ContrastCheck() {
        composeTestRule.setContent {
            FourasTheme(darkTheme = true) {
                RunContrastChecks(isDarkMode = true)
            }
        }
    }

    @Composable
    private fun RunContrastChecks(isDarkMode: Boolean) {
        with(MaterialTheme.colorScheme) {
            ensureCompliantContrastRatio(
                color1 = primary,
                color2 = onPrimary,
```

```
            message = "Primary and onPrimary",
            isDarkMode = isDarkMode,
        )
        ensureCompliantContrastRatio(
            color1 = secondary,
            color2 = onSecondary,
            message = "secondary and onSecondary",
            isDarkMode = isDarkMode,
        )
        ensureCompliantContrastRatio(
            color1 = tertiary,
            color2 = onTertiary,
            message = "tertiary and onTertiary",
            isDarkMode = isDarkMode,
        )
        ensureCompliantContrastRatio(
            color1 = background,
            color2 = onBackground,
            message = "background and onBackground",
            isDarkMode = isDarkMode,
        )
        ensureCompliantContrastRatio(
            color1 = surface,
            color2 = onSurface,
            message = "surface and onSurface",
            isDarkMode = isDarkMode,
        )
    }
}

private fun ensureCompliantContrastRatio(
    color1: Color,
    color2: Color,
    message: String,
    isDarkMode: Boolean,
    enhanced: Boolean = false,
) {
    // formula as provided by WCAG 2.1
    val luminance1 = color1.luminance().plus(0.05)
    val luminance2 = color2.luminance().plus(0.05)
    val lighterColor = max(luminance1, luminance2)
    val darkerColor = min(luminance1, luminance2)
    val contrast = lighterColor / darkerColor
    val requiredRatio = if (enhanced) ENHANCED_CONTRAST_RATIO
        else MIN_CONTRAST_RATIO,
```

```
        require(
            value = contrast >= requiredRatio,
            lazyMessage = {
                "[${if (isDarkMode) "Dark" else "Light"} mode] $message
                insufficient contrast ($contrast) < $requiredRatio:1,
                not EN 301 439 compliant"
            }
        )
    }
}
```

The provided tests support both light and dark mode. This test threw an error if the color contrasts were non-compliant, with messages like:

secondary and onSecondary insufficient contrast (3.1) < 4.5:1, not EN 301 439 compliant.

During the workshop, we discussed calculating the contrast ratio within a lint check by identifying the container and content colors of any given text. However, we discovered that the color value is unreachable during a static lint check. We created a lint rule that warned developers when specifying a color to text composable within our opinionated implementations.

```
/**
 * Configure a column with a surface background and content color
 * based on the contrast type. Helpful to ensure that colors
 * provide the design system contrast compliant colors.
 */
@Composable
fun SurfaceColumn(
    contrastType: ContrastType,
    modifier: Modifier = Modifier,
    verticalScrollState: ScrollState = rememberScrollState(),
    verticalArrangement: Arrangement.Vertical = Arrangement.Top,
    horizontalAlignment: Alignment.Horizontal = Alignment.Start,
    content: @Composable ColumnScope.() -> Unit,

    ) {
    val backgroundColor = when (contrastType) {
        ContrastType.Primary -> MaterialTheme.colorScheme.primary
        ContrastType.Secondary -> MaterialTheme.colorScheme.secondary
        ContrastType.Tertiary -> MaterialTheme.colorScheme.tertiary
        ContrastType.Surface -> MaterialTheme.colorScheme.surface
        ContrastType.Background -> MaterialTheme.colorScheme.background
    }
```

```
    val contentColor = when (contrastType) {
        ContrastType.Primary -> MaterialTheme.colorScheme.onPrimary
        ContrastType.Secondary -> MaterialTheme.colorScheme.onSecondary
        ContrastType.Tertiary -> MaterialTheme.colorScheme.onTertiary
        ContrastType.Surface -> MaterialTheme.colorScheme.onSurface
        ContrastType.Background -> MaterialTheme.colorScheme.onBackground
    }

    Surface(
        modifier = modifier,
        color = backgroundColor,
        contentColor = contentColor,
    ) {
        Column(
            modifier = Modifier.verticalScroll(verticalScrollState),
            verticalArrangement = verticalArrangement,
            horizontalAlignment = horizontalAlignment,
            content = content,
        )
    }
}
```

This lint warning will alert developers if custom colors are set within the scope of components, warning that the color may break compliance with standards. We discovered that because the colors are a reference value, the actual color value required for comparison is unreachable during a static lint check. We want the warning to be more specific but cannot validate whether the color breaks the contrast.

```
public class ColorContrastDetector extends Detector implements SourceCodeScanner{

    private static final Implementation IMPLEMENTATION = new Implementation(
            ColorContrastDetector.class,
            Scope.JAVA_FILE_SCOPE
    );

    final static Issue ISSUE = Issue
            .create(
                    "ColorContrastCheck",
                    "Checks text color contrast",
                    "Checks text color compliance to EN 301 549",
                    Category.A11Y,
                    5,
                    Severity.ERROR,
                    IMPLEMENTATION
            )
            .setAndroidSpecific(true);
```

```
@Override
public @Nullable List<Class<? extends UElement>> getApplicableUastTypes() {
    return Collections.singletonList(UCallExpression.class);
}

@Override
public @Nullable UElementHandler createUastHandler(
    @NotNull JavaContext context
) {
    return new UElementHandler() {
        @Override
        public void visitCallExpression(@NotNull UCallExpression node) {
            String methodName = node.getMethodName();
            if ("SurfaceColumn".equals(methodName)) {
                for (UExpression argument : node.getValueArguments()) {
                    if (argument instanceof ULambdaExpression) {
                        argument.accept(textColorVisitor(context));
                    }
                }
            }
        }
    };
}

private static AbstractUastVisitor textColorVisitor(JavaContext context) {
    return new AbstractUastVisitor() {
        @Override
        public boolean visitCallExpression(@NotNull UCallExpression node) {
            String calleeName = node.getMethodName();
            if ("Text".equals(calleeName)) {
                for (UExpression argument : node.getValueArguments()) {
                    var isColorScheme = argument.asSourceString()
                            .startsWith("MaterialTheme.colorScheme.");
                    if (!isColorScheme) {
                        continue;
                    }

                    context.report(
                            ISSUE,
                            node,
                            context.getCallLocation(
                                    node,
                                    false,
                                    true
                            ),
```

```
                                "Setting custom text colors in SurfaceColumn " +
                                  "may result in insufficient contrast according" +
                                  " to EN 301 549 accessibility standards.",
                                null
                        );

                    }
                }
                return super.visitCallExpression(node);
            }
        };
    }
}
```

During the workshop, one of the participants proposed calculating the contrast ratio within a lint check by identifying any container background color and the applied text color. However, as mentioned previously, we cannot get the resource value in the lint environment. We created a lint rule that warned developers when specifying a color-to-text composable within our opinionated implementations. This type of warning is the next best thing to the initial idea. Remember that this lint check only supports one type of layout and is not designed to be applied everywhere. Should developers find this lint check useful, the implementation can extend to check additional methods.

5.2.3.2 Screenshot testing exploration

Screenshot testing compares one screenshot of a prior reference state and another of the modified visual state. These tests verify that layouts are similar to prior layouts to avoid accidental layout changes. We wanted to check if this test could validate contrast checks. We will use this differently, using only the image representing the test state and disregarding the old state to evaluate color contrasts.

We searched for official tools that Google supports to compare screenshots but found no stable releases that provide screenshot-testing functionality. We found stable alternatives, such as Paparazzi [43], from a reputable company and explored the possibilities with their library.

We used the paparazzi screenshot test library by implementing two tests: The first takes a screenshot of a contrast-compliant layout. The second one presents a representation of a non-compliant layout. The challenging part is that we need to process the created images and determine the contrast of the elements.

```
class ContrastScreenshotTest {
    @get:Rule
    val paparazzi: Paparazzi = Paparazzi()
```

```
    @Test
    fun hasCompliantColors() {
        paparazzi.snapshot(name = "SurfaceColumn_ComplaintColors") {
            FourasTheme {
                SurfaceColumn(
                    contrastType = ContrastType.Background,
                ) {
                    Text(text = "test text")
                }
            }
        }
    }
}
```

This test generates an image of the SurfaceColumn implementation. Removing the expected background color from the generated image will compare the contrast of all the remaining colors in the image to the background color. The visual representation will pass if it complies with the WCAG 2.1 standard. However, we realized this approach has issues and could provide false positives in scenarios, such as when there are no colors left when removing the background from the image. Given this uncertainty, we decided that applying screenshot testing is not suitable for our purposes.

5.2.3.3 Orientation validation

EN 301 549 10.1.3.4
WCAG 2.1. 1.3.4 Orientation

The application must comply with the orientation level AA requirement and offer multiple orientation alternatives for layouts, except for layouts where a fixed orientation is essential for the application to function. Verifying that the content provides the same functionality for changes in screen orientation can be done with automated instrumentation tests.

```
@RunWith(AndroidJUnit4::class)
class OrientationTest {

   @get:Rule
   val composeTestRule: AndroidComposeTestRule
     <ActivityScenarioRule<MainActivity>, MainActivity>
       = createAndroidComposeRule(MainActivity::class.java)

   @Test
   fun verifyOrientationChangesAvailable() {
      composeTestRule.onOrientation(
```

```
            onPortrait = {
                onNodeWithText("Fouras").assertIsDisplayed()
            },
            onLandscape = {
                onNodeWithText("Fouras").assertIsDisplayed()
            },
        )
    }

  private fun <R : Activity, T : ComponentActivity>
        AndroidComposeTestRule<ActivityScenarioRule<R>, T>.onOrientation(
      onPortrait: AndroidComposeTestRule<ActivityScenarioRule<R>, T>.() -> Unit,
      onLandscape: AndroidComposeTestRule<ActivityScenarioRule<R>, T>.() -> Unit,
    ) {
      activityRule.scenario.onActivity { activity ->
        activity.requestedOrientation = ActivityInfo.SCREEN_ORIENTATION_PORTRAIT
      }
      waitForIdle()
      onPortrait(this)

      activityRule.scenario.onActivity { activity ->
        activity.requestedOrientation = ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE
      }
      waitForIdle()
      onLandscape(this)
    }
}
```

This test will check that the main activity displays the application name in portrait and landscape mode. The orientation callback will allow the developer to change the layout if necessary.

5.2.4 Validation

We presented the proposed solutions of the framework to experienced Android developers during a workshop to validate our implementations. Throughout the workshop, we asked the participants to validate our technical solutions of automated tests that verify design system resources to ensure guideline-compliant layout elements. We asked the participants to evaluate the technical aspects of these implementations based on ease of implementation and whether they would provide value to their accessibility compliance efforts. We also asked them to anticipate potential challenges and other benefits of integrating such accessibility features into their existing application development workflows. The immediate feedback from this session was positive. The participating developers recognized the potential of the presented solutions and were interested in trying them as they considered compliance processes

in preparation for upcoming regulations.

5.2.5 Evaluation

To evaluate whether Android frameworks can improve developer accessibility awareness, we aim for developers to find value in our framework. One of the participants in the workshop we conducted mentioned that their company had initiated discussions with designers on the topic of improving accessibility in preparation for regulations and was happy and willing to try our implementation to start the process. During our interviews and workshops, we identified that developers prefer design system implementation solutions and found that automated tests are beneficial but require significant effort and maintenance. We focused on improving the current offering and implementing suggestions from the workshop during this cycle. We learned that providing flexible solutions that verify compliance and functionality, rather than forcing developers to apply solutions that provide optionality, is preferable to strict implementations.

We aim for our findings to serve as a starting point for developers on their accessibility compliance journey. We find that providing developers with a helpful framework that supports the right amount of flexibility will improve developer accessibility efforts, thereby enhancing awareness by keeping developers informed about their choices that affect compliance. Finding the balance between rigidity and flexibility is difficult, but in instances where optionality is available, our framework should trust developers to consider compliance. During the workshop, one of the participants mentioned that implementing checks for contrasting colors would be a massive undertaking given the custom elements but noted that he had noticed invalid configurations of color usage for color contrasts throughout the application. We oriented our solutions to avoid the requirements of implementing automated tests and rather use design system solutions that provide the correct contrasts by default (with the option to change them manually, providing flexibility), removing the need for automated tests.

5.2.5.1 Promoting accessibility adoption by increasing developer awareness (RQ3)

We explored how accessibility compliance testing impacts developers to promote adoption. It is possible to create custom tooling that verifies compliance within a design system approach, where layout elements are restricted but thoroughly tested for compliance. Following a design system that implements compliance by design and informs developers that if they deviate from the intended design, they will receive a notification to consider the change more thoroughly.

We have identified that developers primarily want to be informed and follow a structure and will not implement accessibility features if they are not required. One interviewee mentioned that they had requirements for a previous project. However,

when starting at a new company without requirements, the interviewee did not put additional effort into ensuring accessibility compliance. Indicating that if accessibility is not a requirement, accessibility compliance is at risk of not being considered by developers.

We believe the key to promoting accessibility adoption and awareness among developers is creating a design system that aligns with their development style. Applying a carefully considered design system, composed of compliant layout elements and warnings when deviations occur, creates a robust system for informing developers and improving awareness of accessibility compliance. Creating automated tests is a major undertaking but a viable option if implemented correctly to ensure that colors and layouts comply with requirements. Relying on a design system provides predefined resources to use but relies on developers to implement them correctly. Our proposed solution is to provide implementations that default to the correct resources while allowing developers to override the opinionated design if required for edge cases.

6

Fouras framework

This section is a summary of the capabilities of the Fouras framework.

The European Commission and the Swedish Agency for Digital Government DIGG set regulations for all public applications in 2025. This framework provides practical examples and strategies to achieve accessibility compliance with a subset of EN 301 549 and WCAG 2.1 standards that will apply in Sweden. This framework primarily targets organizations and developers who produce Android applications that must add or improve accessibility features.

The following requirements have been examined:

- Non-text content
  EN 301 549 11.1.1.1.1
  WCAG 2.1 1.1.1

- Contrast
  EN 301 549 10.1.4.3
  WCAG 2.1 1.4.3 (Level AA)
  WCAG 2.1 1.4.6 (Level AAA)

- Orientation
  EN 301 549 10.1.3.4
  WCAG 2.1. 1.3.4

Throughout this study, we have identified technical and systematic approaches to validate accessibility regulations through automation, tooling, and custom design systems. Our approach differs from other available tooling in that our examples allow implementing compliance checks incrementally rather than reporting all issues simultaneously. This incremental approach allows developers to selectively approach by prioritizing compliance testing where it is most needed.

We recognize that this subset is limited, but we believe that extrapolating the concepts of our findings can become a foundation for further development in regulatory accessibility compliance validation.

6.1 Systematic accessibility design

We found that a strong systematic design approach to accessibility requirement compliance is key to providing developers with the tools to support accessibility features. The design system intends to provide developers with compliant layout components to create layouts. To achieve a systematic design approach, developers, quality assurance, designers, and management must work together to ensure that accessibility features are considered and enforced throughout development. Management needs to consider accessibility requirements when presenting new features. Designers must consider accessibility requirements when designing application layouts. Developers need to understand the requirements and be vigilant by identifying lacking accessibility needs when they present themselves and quickly inform other parties when layouts fail to comply with accessibility standards.

In Android applications, a common way to structure design resources is to use the material design system. This design system requires the developers to consider combinations of colors, such as background colors, and what color can be atop the background color. Our automated approach assists developers in validating that the colors are compliant with contrast standards; our framework provides practical implementations that automate testing a range of colors in the design system to comply with contrast requirements of 4.5:1 and 7:1 to comply with WCAG 2.1.

The framework also provides an automated test approach to validate that the layouts present the same information in multiple orientations as required by WCAG 2.1. Designers and developers should expand the design system with layout elements that accommodate the standards' requirements. The custom layout elements we suggest are opinionated implementations of existing material design layouts that require the developer to specify the use case to avoid accidental non-compliant implementations. The automated tests will provide immediate feedback to the developers if their layouts are non-compliant, and lint checks will provide warnings if the developer implements features according to the standards.

We recommend implementing the material design system and selectively providing compliance contrasting colors throughout the application. We also suggest creating warnings that trigger when incorrect use of the design system to prevent non-compliant layouts.

6.2 Accessibility feature testing

We found multiple approaches that create compliance validation, such as custom layouts with predetermined contrasting colors and providing warnings when providing alternative colors function well in conjunction with one another. These implementation warnings are available to developers during development and raise awareness about accessibility features when the implementation can become non-compliant.

This custom approach improves the available tooling offering when validating compliance with specific accessibility guidelines.

Automating the compliance validation process is challenging and will require maintenance and additional design limitations when developing applications. Similarly to regular quality assurance, testing requires partial manual labor. However, our framework aims to keep the required manual testing to a minimum. The developers we spoke with had experience with quality assurance engineers applying manual tools like Accessibility Scanner to validate that the application complies with the requirement standards. Allowing developers to test accessibility features with automation will reduce the required time for regression testing by continuously verifying accessibility compliance standards throughout development. We support including a focus group of impaired users who rely on the accessibility features in the testing process, if possible. Users who need the features are the ideal validation candidates, given that they are the target audience and can provide better feedback to the user experience than any framework can.

6.3 Compliance deviation detection

We recommend adding safety measures that prevent developers from deviating from the design system. By applying Lint, we created an implementation that detects when the implementation deviates from the intended use of the design system, thus possibly deviating from compliance. The warning we provide ensures that the developer is aware of compliance requirements when specifying colors to preconfigured layout element implementations so that they inherently implement compliant contrasting colors. We also discovered limitations of static code analysis in Android, such as being unable to resolve Android resources due to the reference-based resource system in Android. However, we discover an alternative approach to inform the developer about potentially breaking the compliance. This warning acts as a safeguard for developers to be mindful about specifying resources to compliant layouts.

6.4 Expanding the framework

Given the time constraints of this thesis, our initial implementation focused on a subset of accessibility guidelines that we determined would provide the most significant value to developers and designers in enhancing users' experiences with accessibility features. We determined our guideline selection strategy by prioritizing guidelines that address the most prevalent disabilities among mobile users, in combination with our subjectively estimated ease of implementation, to fit within our limited time.

Prior research indicates that visual impairments are among the most common disabilities affecting mobile interaction [**?**]. Therefore, we prioritized Android accessi-

bility guidelines that directly support users with visual impairments, such as programmatic support for ensuring sufficient color contrast, as outlined in the provided guidelines for calculating color contrast. This approach enabled us to establish a foundational framework for approaching accessibility testing. However, accessibility features require early consideration of implementation built for validation, which includes testing, making it easier for the development team to automate compliance tests.

6.5 Fouras Android implementation

The accessibility features discussed in this thesis are available for exploration in our public GitHub repository.

Github.com/HannesHaggander/fouras

7

Discussion

In this chapter we will discuss our key findings for each research question, our interpretation of the results and how our suggested framework differs from current development practices, and discuss the limitation of our research.

7.1 Validating accessibility standards (RQ1)

Our initial ambition was to utilize the current automated accessibility offering of tools to find techniques that validate accessibility standards that will apply within the European Union. We were surprised by how limited developers are regarding validating compliance with any specific requirement. Generic evaluation tools need manual guidance, and suggestions to resolve the issues are limited. We found that manual testing tools were the most common approach to validating accessibility requirements, according to experienced developers. However, all the developers we interviewed had little experience validating accessibility features. Shockingly, we learned that the developers did not use available accessibility validation tools to ensure accessibility compliance of their application. We also discovered that developers had yet to update themselves on upcoming regulation compliance details that will apply to their Android applications in 2025.

We hoped to validate as many accessibility standards as possible using the tooling. However, we found that the current offering needs more support for automation and configuration options to verify compliance with specific standards. We took it upon ourselves to use our expertise in Android development to provide a framework composed of approaches to implement a custom design system that provides opinionated implementations that comply with EN 301 549 standard and WCAG 2.1 accessibility requirements.

Our framework provides a set of automated tests, lint warnings, and systematic design implementations to guide developers into creating compliant applications. Given the differences in Android projects, we realized developers need custom solutions. Android applications are different, even though they follow similar approaches, making designing generic approaches challenging for all projects. Instead, we provide examples of implementations of creating compliant layout elements, validating compliance using automated tests, and validating that the design system is following expectations. This approach is suitable for developers given that most

forms of learning provide examples of solving a problem and let developers modify and apply our examples to meet their needs.

We interviewed developers and found they would be pleased with flexible implementations and lint rules that provide warnings rather than strict implementations that must follow the rules. This approach allows them to prioritize importance and deviate from the design system if needed. We do not want to force developers into a particular way of implementing. However, we want to display compliance regulation expectations and approaches to solve the problems they will face during development.

7.2 Improving developer awareness (RQ2)

When researching academic papers on accessibility testing in Android, we understood that the adoption of accessibility features in publicly available projects is extremely low and would need to improve. We realize that publicly available projects do not necessarily represent the accessibility support of industry applications. However, it indicates that public project developers seem to neglect accessibility.

We wanted to understand why developers neglect to implement accessibility features. Research suggests that the simple reason why developers do not consider accessibility is because they do not know of accessibility features. We confirmed this to be the case with the developers we interviewed and got to understand that accessibility is not prioritized by organizations or developers if not forced upon them.

During our interviews with developers, interestingly, they seemed to all have worked with some form of accessibility requirement before in their careers. However, they did not have any requirements to implement accessibility features at their current place of work. Curiously, developers who know of accessibility features seem to disregard testing for accessibility features if they are not required, further indicating that developers are only bothered with accessibility if not required to do so.

We asked developers about their preferred approach to improving awareness of accessibility features and found that flexible solutions better suit the needs of developers than strict implementations. A developer mentioned during the workshop that adding checks as lint rules to identify non-compliant layouts would be a good, flexible solution rather than forcing an opinionated implementation upon a product. We considered this when developing our framework, and when we presented our suggestions to the developers, they mentioned that they would consider trying our solutions as an initial step in their accessibility efforts. We find that the low adoption rate is not a matter of developers avoiding the implementation of accessibility features; rather, accessibility validation is a hurdle and lacks developer support. Projects typically apply Lint rules to verify code correctness, and we believe that adding additional accessibility checks will enhance developer adoption of accessibility features and general developer awareness of accessibility by reminding developers of compliance through warnings.

7.3 Promote accessibility adoption (RQ3)

Developers who participated in the workshop positively received our technical solutions to improve accessibility feature adoption. This validated our expectations that there is value in implementations that correctly notify developers and assist their accessibility feature improvement efforts during development. As found in prior research, most developers are entirely unaware of accessibility features, which is one of the primary reasons for accessibility feature negligence. We were positively surprised when the developers were excited to try parts of our framework added to their projects.

We have identified a key feature to allow the developers to cherry-pick where to start and incrementally increase the coverage rather than forcing every layout to comply with standards. Developers prefer an incremental coverage approach to improving accessibility support as it fits better when modifying large projects; given that large projects may require several corrections before sufficiently supporting accessibility features, doing such changes in small chunks is preferable. Developers prefer to make small changes and correct one layout at a time. Current tooling, like the Accessibility Scanner, allows the developer to evaluate one screen at a time but does not allow configuration of what to examine on the layout. We found that configuration is key and is necessary for developers to reduce friction and improve developer adoption.

7.4 Evaluating framework efficacy

Our framework applies to most projects, given that our approaches are flexible and can be modified to fit the current development flow. It is not easy to take the first step, but our framework can become a starting point for developers. During our workshop, where we presented our framework progress, the developers were happy to try some of our suggestions, especially implementing the opinionated image component. They told us that they were talking about accessibility feature support within the company and about starting the compliance process, and the developer found that our implementation would be a good first step to get them going with their accessibility efforts. We hope the company will try our implementation and provide feedback on its usefulness. Implementing the framework into a production application would be a great validation opportunity to compare our framework's efficacy to other available tooling.

We understand that the framework provides narrow use for complete compliance with standards, given that our offer only supports a few selected standards. Our ambition for the framework was to have it promote accessibility feature adoption and create a more inclusive digital environment for impaired users. We believe that developers can use the limited offer as a first step and then incrementally

add accessibility support themselves. Making an application compliant with new standards will require extensive time and effort. Finding somewhere to start can be challenging, and our framework provides a starting point.

7.5 Research implications

This study explores the current state of accessibility testing in Android applications. It establishes that accessibility validation tooling needs modifications to evaluate compliance with new European Union standards confidently by using current testing frameworks and automated testing (RQ1 1.4.1). Our framework provides developer-approved implementations of a subset of standards previously not supported by tooling, providing organizations with strategies to apply when new regulations apply in 2025.

The study also identifies technical solutions to improve developers' awareness of lacking accessibility feature support in their applications. It provides tools and Lint rules to warn developers when they deviate from the expected system design implementations (RQ2 1.4.2). This automatic process of informing developers will, in turn, improve developer awareness when potentially breaking compliance with accessibility standards to mitigate the issue of developer negligence to accessibility support. We believe that informing developers when performing changes to the application will reduce the effort needed for regression testing to verify the accessibility standard compliance.

7.6 Artificial intelligence assistance

An aspect of accessibility features that we did not cover in the framework is artificial intelligence (AI) advancements. Android recently received updates for features that implement artificial intelligence into the operating system. The premise is similar to Google's Accessibility Scanner; the AI will scan and describe images to the user using an artificial voice rather than relying on developers to implement labels for TalkBack to use [44]. Removing the need for developers to provide descriptions has the potential to be a huge improvement for inclusion and allow impaired users to use applications without relying upon special implementations. However, until AI improves to the point of perfection, there is still a need for intended accessibility support by application developers. However, this kind of technology is promising when accessibility support is missing.

AI is known for making incorrect assessments, potentially resulting in an incorrect interpretation of images, which could be worse than no description for an impaired user, depending on the perceived usefulness. Then again, AI is advancing rapidly, and companies are expanding their reach efforts, making it challenging to evaluate the future benefits of this technology. Until AI provides equal assistance to impaired users as accessibility support, developers should provide accessibility support where

possible and have AI assistance as a secondary option. We assume that Google, which manages the Android SDK development and heavily invests in AI development, will insert AI into Android to improve the experience for disabled users.

7.7 Future work

This section covers how the Fouras framework can expand to provide more coverage and improve accessibility requirements.

7.7.1 Expand compliance tests suite

We want to improve the framework test compliance offering and provide a comprehensive test suite for EN 301 549 and WCAG 2.1 standards to support Android developers to comply with the regulations in Sweden. Providing a comprehensive testing framework is a massive undertaking, given that our limited offering requires time and effort. However, given our indications from developers within the industry, this kind of framework will be highly valuable to organizations once the regulations become law in 2025.

7.7.2 Validate in production

We have proved that our theoretical implementations of accessibility feature support work in an isolated environment, but hidden issues may only present themselves when implemented into a production application environment. Our framework would do well in production, given that the solutions are flexible and configurable to accommodate project needs. However, we can only ensure usefulness once we test the framework in its intended setting.

Retroactively adding accessibility support is a necessary and daunting task for many organizations. Most Android projects contain legacy code, which our up-to-date solutions do not apply to. Larger organizations often have a lengthy evaluation process before implementing new working methods. Then again, accessibility is seemingly not prioritized today but will have to be in 2025 due to regulations, meaning that unprepared companies may need to rush decisions to comply. Companies must start considering accessibility and may be eager to find solutions for their accessibility problems, and that is where solutions like ours benefit the developers.

7.8 Threats to validity

In this section, we discuss the trustworthiness and validity of our research by addressing potential biases regarding construct, internal, and external validity concerns. We

followed the proposed structure provided by Runeson and Höst [34].

7.8.1 Construct validity

We conducted our interviews and the workshop, expecting interviewees and attendees to have sufficient knowledge of Android development in areas such as test automation and a general understanding of creating design elements using Jetpack Compose to comment on our suggested implementations. We assumed this knowledge because both aspects of modern Android development are standard practices within the industry. We believe that the participants had experience with these practices after our discussions, given that we found their feedback reasonable. Nevertheless, it may be an oversight on our part, and we should have asked questions about their knowledge of the standard practices in our introductory questions.

During the workshop, a few attendees were naturally quieter than others. We made an effort to receive information from all attendees by asking for their opinions. However, we only managed to get them to participate in discussions when we asked for their opinion. We expected differing levels of participation within the workshop group, but ideally, being able to promote more discussion where everyone is participating could lead to more exciting insights and feedback on our results.

As a result of applying a design research methodology, we iteratively received feedback from developers and improved our solutions to accommodate them. These developers are not necessarily representative of developers in general, given that we were happy to include any experienced developer who wanted to participate in the study. Had we interviewed other developers with different experiences, they would most likely have given us feedback that differed from the feedback we received, potentially leading to a result different from the one we have now.

7.8.2 Internal validity

A significant threat is that we accepted the inclusion of any experienced Android developer to participate in the study with only one criterion of several years of domain knowledge. The majority of the developers we included in the study worked at the same company, making their opinions potentially apply to the same project rather than providing a general approach that applies to different project structures. We did ask several developers to participate in interviews but very few replied.

Another threat is that we failed to use the current accessibility tools sufficiently, even though we did our best to use and implement them properly. Our findings indicated that these tools could not identify standards to a sufficient degree, but we recognize that our knowledge of these tools is limited and that the tools are more capable than we found them to be.

We also find that there is inherent researcher bias in the analysis of the workshop and interviews. Semi-structured interviews offer in-depth questioning of topics of interest to each participant, but the result is inherently complicated to replicate, as there is no rigid structure. Follow-up questions are unique to every discussion, and it is possible that other researchers, with their perspectives, experiences, and interpretations, will conduct the interviews differently or draw different conclusions from the discussions. The subjective interpretation of qualitative data could influence the findings, and we acknowledge this as a limitation.

7.8.3 External validity

Our framework only provides Android development examples, which will not apply to other platforms, given that they are platform-specific implementations. However, the new regulations for accessibility require other platforms to implement the same standards, and our approaches may be generalizable when translated into domain-specific approaches for different platforms. The most similar platform to Android is iOS, as both target native development for different mobile platforms. Our approaches should also apply to iOS, but the implementation details are different, given that iOS does use Swift, not Kotlin. Multiple frameworks target both platforms, known as multiplatform frameworks, such as Kotlin multiplatform and Flutter. Our findings are also generalizable to these, but implementation details differ.

Our framework provides a small number of accessibility checks. Acknowledge the uncertainty of whether total coverage of all accessibility requirements is possible or if supplementing with other means is required to achieve total regulatory compliance. Even so, we found that developers found value in our solutions as an introduction to accessibility features. Automating some tests and raising developers' awareness of accessibility are successes.

The validation process of our framework was performed on our sample application, which we acknowledge is simple in comparison to large-scale applications and perhaps proves challenging to adapt to our suggested changes. Large-scale applications may face challenges, such as using old legacy code in conjunction with modern code standards. However, our approach is flexible and can partially apply to verify only certain aspects of compliance if needed, without requiring developers to implement every test. If the tests need to be modified to accommodate larger applications, we assign that task to the developers, as every application implementation requirement is unique.

7.8.4 Reliability

We gathered all of our industry information data using semi-structured approaches, which makes it difficult to replicate the exact results when others perform the same task. However, applying semi-structured techniques yields better results as we take

advantage of individual experiences rather than a strict interview format that assumes our questions will result in the best possible reflection of their experiences.

Our experience in Android development gave us a significant advantage in understanding developer needs and providing appropriate implementations that reflect those needs. These insights guided us in the interview setting, allowing us to ask questions that brought up relevant discussion topics during the semi-structured interview format, considerably improving the implementation suggestions and their usefulness to developers.

8
Conclusion

We have identified that Android application developers have not implemented accessibility feature support in their applications and are unprepared to comply with new European accessibility regulations. In prior research, our experience and the data we collected throughout this thesis indicate that accessibility features are rarely implemented within the industry. This negligence makes applications challenging to operate for impaired users relying on accessibility tools to interact with their Android smartphone(s). To address this deficiency, we have created a framework that provides regulation-compliant examples of solutions to assist developers in ensuring accessibility compliance with the new European Accessibility Act (EAA) during development. The framework provides Android developers with insights on how to implement support for accessibility features into their projects and examples of how to validate these features using automated testing methods when used in conjunction with design systems [3].

We have examined the current state of automated accessibility feature testing and found that it needs improvement to validate specific standard requirements in EN 301 549 [1] and WCAG 2.1 [2]. Our outcomes are practical approaches to a subset of accessibility requirements. During the development of this framework, we had professional Android developers provide their feedback on our results to ensure industrial relevance. We aspire to offer an introductory step for developers and organizations in their accessibility feature support journey.

Throughout our research, our ambition has been to improve Android applications' adoption of accessibility features. With our guidelines on what to consider during development, practical code examples of our design elements in conjunction with tests that verify compliance, and strategies to maximize the possibility of allowing impaired users to operate Android applications more easily. By improving accessibility features, we ultimately seek to promote digital inclusivity and equal opportunity for all users.

Bibliography

[1] E. T. S. Institute. Accessibility requirements for ict products and services. [Online]. Available: https://www.etsi.org/deliver/etsi_en/301500_301599/301549/03.02.01_60/en_301549v030201p.pdf

[2] W. W. W. C. (W3C). (2023-09-21) Web content accessibility guidelines (wcag) 2.1. [Online]. Available: https://www.w3.org/TR/WCAG21/

[3] E. Commission. European accessibility act: Q & a. [Online]. Available: https://ec.europa.eu/social/main.jsp?catId=1202&intPageId=5581&langId=en

[4] J. R. Ehrlich, A. Agarwal, C. Young, J. Lee, and D. E. Bloom, "The prevalence of vision impairment and blindness among older adults in india: findings from the longitudinal ageing study in india," *Nat Aging*, vol. 2, no. 11, pp. 1000–1007, 2022, pMID: 37118083; PMCID: PMC10148950. Erratum in: Nat Aging. 2023 Dec;3(12):1602. [Online]. Available: https://www.nature.com/articles/s43587-022-00298-6

[5] Google. (2023) Next billion users. [Online]. Available: https://blog.google/technology/next-billion-users/

[6] ——. (2022) An anthology of insights, for a more inclusive internet. [Online]. Available: https://blog.google/technology/next-billion-users/anthology-insights-more-inclusive-internet/

[7] A. Alshayban, I. Ahmed, and S. Malek, "Accessibility issues in android apps: state of affairs, sentiments, and ways forward," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1323–1334. [Online]. Available: https://doi.org/10.1145/3377811.3380392

[8] E. Commission. European accessibility act. [Online]. Available: https://ec.europa.eu/social/main.jsp?catId=1202

[9] C. Vendome, D. Solano, S. Liñán, and M. Linares-Vásquez, "Can everyone use my app? an empirical study on accessibility in android apps," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2019, pp. 41–52.

[10] Google. (2024) Espresso. [Online]. Available: https://developer.android.com/training/testing/espresso

[11] Google. (2025) Android accessibility overview. [Online]. Available: https://support.google.com/accessibility/android/answer/6006564

[12] ——. (2024) Use talkback gestures. [Online]. Available: https://support.google.com/accessibility/android/answer/6151827?sjid=14246681936652185547-EU

[13] Android Developers. (2024) Making apps more accessible. [Online]. Available: https://developer.android.com/guide/topics/ui/accessibility/apps

[14] DIGG - Myndigheten för digital förvaltning. (2024) Digital tillgänglighet. [Online]. Available: https://www.digg.se/kunskap-och-stod/digital-tillganglighet

[15] D. M. för digital förvaltning. Det här är en 301 549 och wcag. [Online]. Available: https://www.digg.se/webbriktlinjer/lagar-och-krav/det-har-ar-en-301-549-och-wcag

[16] Google. (2023-05-09) Build accessible apps. [Online]. Available: https://developer.android.com/guide/topics/ui/accessibility

[17] ——. (2023-05-09) Make your android app more accessible. [Online]. Available: https://developer.android.com/courses/pathways/make-your-android-app-accessible

[18] ——. (2024-01-03) Make your android app more accessible. [Online]. Available: https://developer.android.com/guide/topics/ui/accessibility/apps

[19] Google. (2024) Accessibility scanner. [Online]. Available: https://play.google.com/store/apps/details?id=com.google.android.apps.accessibility.auditor

[20] Google. (2024) Accessibility test framework for android. [Online]. Available: https://github.com/google/Accessibility-Test-Framework-for-Android

[21] ——. (2024) Test your app's accessibility. [Online]. Available: https://developer.android.com/guide/topics/ui/accessibility/testing

[22] P. Acosta-Vargas, L. Salvador-Ullauri, J. Jadán-Guerrero, C. Guevara, S. Sanchez-Gordon, T. Calle-Jimenez, P. Lara-Alvarez, A. Medina, and I. L. Nunes, "Accessibility assessment in mobile applications for android," in *Advances in Human Factors and Systems Interaction*, I. L. Nunes, Ed. Cham: Springer International Publishing, 2020, pp. 279–288.

[23] S. Kashif, M. Ahmad, M. Shahid, M. A. Habib, and K. Rizwan, "Development of an automated accessibility evaluation plugin tool for mobile applications," in *2022 3rd International Conference on Innovations in Computer Science & Software Engineering (ICONICS)*, 2022, pp. 1–10.

[24] M. M. Eler, J. M. Rojas, Y. Ge, and G. Fraser, "Automated accessibility testing of mobile apps," in *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, 2018, pp. 116–126.

[25] M. Tafreshipour, A. Deshpande, F. Mehralian, I. Ahmed, and S. Malek, "Ma11y: A mutation framework for web accessibility testing," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2024. New York, NY, USA: Association for Computing Machinery, 2024, p. 100–111. [Online]. Available: https://doi.org/10.1145/3650212.3652113

[26] A. Alshayban and S. Malek, "Accessitext: automated detection of text accessibility issues in android apps," ser. ESEC/FSE 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 984–995. [Online]. Available: https://doi.org/10.1145/3540250.3549118

[27] F. Mehralian, N. Salehnamadi, S. F. Huq, and S. Malek, "Too much accessibility is harmful! automated detection and analysis of overly accessible elements in mobile apps," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '22. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3551349.3560424

[28] M. Tafreshipour, A. Deshpande, F. Mehralian, I. Ahmed, and S. Malek, “Ma11y: A mutation framework for web accessibility testing,” in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2024. New York, NY, USA: Association for Computing Machinery, 2024, p. 100–111. [Online]. Available: https://doi.org/10.1145/3650212.3652113

[29] E. Knauss, “Constructive master’s thesis work in industry: Guidelines for applying design science research,” in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, 2021, pp. 110–121.

[30] D. M. för digital förvaltning. Webbtillgänglighet: ska vi följa wcag 2.1 eller en 301 549? [Online]. Available: https://www.digg.se/om-oss/nyheter/digital-tillganglighet/nyheter/2024-08-05-webbtillganglighet-ska-vi-folja-wcag-2.1-eller-en-301-549

[31] A. R. Hevner, S. T. March, J. Park, and S. Ram, “Design science in information systems research.” *MIS Quarterly*, vol. 28, no. 1, pp. 75 – 105, 2004. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=bsu&AN=12581935&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[32] P. Runeson, E. Engström, and M.-A. Storey, *The Design Science Paradigm as a Frame for Empirical Software Engineering.* Germany: Springer, 2020, pp. 127–147.

[33] W. W. W. C. (W3C). (2008-12-08) Mobile accessibility at w3c. [Online]. Available: https://www.w3.org/TR/WCAG20/

[34] P. Runeson and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, Apr. 2009. [Online]. Available: https://doi.org/10.1007/s10664-008-9102-8

[35] O. A. Adeoye-Olatunde and N. L. Olenik, “Research and scholarly methods: Semi-structured interviews,” *JACCP: JOURNAL OF THE AMERICAN COLLEGE OF CLINICAL PHARMACY*, vol. 4, no. 10, pp. 1358–1367, 2021. [Online]. Available: https://accpjournals.onlinelibrary.wiley.com/doi/abs/10.1002/jac5.1441

[36] V. Braun and V. C. and, “Using thematic analysis in psychology,” *Qualitative Research in Psychology*, vol. 3, no. 2, pp. 77–101, 2006. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1191/1478088706qp063oa

[37] Google. (2024) Test your app’s accessibility. [Online]. Available: https://developer.android.com/guide/topics/ui/accessibility/testing#analysis

[38] ——. Accessibilitychecks. [Online]. Available: https://developer.android.com/reference/androidx/test/espresso/accessibility/AccessibilityChecks

[39] A. Developers. (2021) Color contrast - accessibility on android. [Online]. Available: https://www.youtube.com/watch?v=RHHpljSTDxA

[40] Google. Ui/application exerciser monkey. [Online]. Available: https://developer.android.com/studio/test/other-testing-tools/monkey

[41] ——. Made for google. [Online]. Available: https://get.google.com/madefor/

[42] ——. The color system. [Online]. Available: https://m2.material.io/design/color/the-color-system.html#color-theme-creation
[43] Square. Paparazzi. [Online]. Available: https://cashapp.github.io/paparazzi/
[44] Google. What's new with talkback 14.2. [Online]. Available: https://support.google.com/accessibility/android/answer/14788172?hl=en
[45] Adobe. (2024) Adobe firefly. [Online]. Available: https://www.adobe.com/products/firefly.html

A

Appendix 1

A.1 Cycle 1

A.1.1 Interview structure

This is the interview structure that we used for the semi-structured interviews that was held at the start of the first cycle to better understand what experienced developers knowledge of accessibility features and tool implementation in Android.

Thesis introduction
This thesis is about improving the accessibility of mobile applications by investigating why the adoption rate of accessibility tools is low and how we can improve it as developers. Ultimately we want applications to comply with accessibility guidelines set by governments and companies to make the experience for impaired users better.

The goal of this interview is to understand Android developers general knowledge of accessibility features, how to implement them and how to verify them using testing. By understanding developers better and the challenges they encounter we aim to identify gaps in current accessibility implementation development processes. We want to understand the developers current workflow of accessibility implementation and ultimately learn and create a framework that can improve other developers.

Consent Interviewee is asked for permission to record the interview and notifying them that the interviewee and their answers will be anonymous and not shared with any other party.

Interviewee background

1. Please tell us what it is that you do in your current role.

2. How long have you been working as a professional software engineer?

Understanding of accessibility

3. Are you aware of the The European Accessibility Act 2025?

4. Do you know of accessibility features in (Android) applications?

5. Do you know of the accessibility guidelines provided by Google or the World Wide Web Consortium?

Follow-up:
- How did you first become aware of (Android) accessibility features?
- Have you had requirements from designers or management to implement accessibility features?
- In your opinion; What are the most important accessibility features?
- Have you reviewed any accessibility resources or guidelines from Google or other sources? If so, which ones?
- Do you believe that developers should be informed about lacking accessibility features in their application by the publishing platform?

Implementation challenges

6. Have you verified accessibility features using manual or automated testing before?

7. What tools and processes were used for such tests?

8. Have you used any automated tools or testing frameworks for accessibility in Android development?

9. If you were tasked with integrating accessibility feature testing into your Android development process, how would you approach it?

Follow up:
- What challenges do you typically face when implementing accessibility features?
- How do you stay updated on the latest accessibility guidelines and features?
- Have you implemented tests that verify accessibility features previously?

Developer processes

10. Do you believe automated testing to be helpful to verify functionality?

11. How do you typically integrate new tools or practices (such as automated testing) into your development workflow?

12. Would you find value in a framework that provides actionable feedback on accessibility issues in your Android applications?

13. What do you think could be done to raise overall awareness about accessibility among Android developers?

14. Based on your experience, do clients or management typically request accessibility features in Android applications?
15. Are you currently using any tooling to verify accessibility features?

Follow up:
- What factors do you consider when deciding upon implementing a new tool or process?
- What type of feedback do you find most useful from a testing framework?
- Are you a part of any communities that are beneficial to accessibility testing?
- Have you worked at any company that specifically asks for accessibility feature requirements?
- Do you believe that understanding the importance of accessibility features to impaired users would promote implementing them?

Closing

16. Are there any methods or resources that you would like to share that could assist other developers to implement accessibility feature testing more effectively?

A.1.2 Application layouts

This section showcases the application with intentional accessibility errors that was used to verify automated tests on. Note that all images are created by me using artificial intelligence tool provided by Adobe Firefly [45] and thereby not subject to copyright and safe to use.

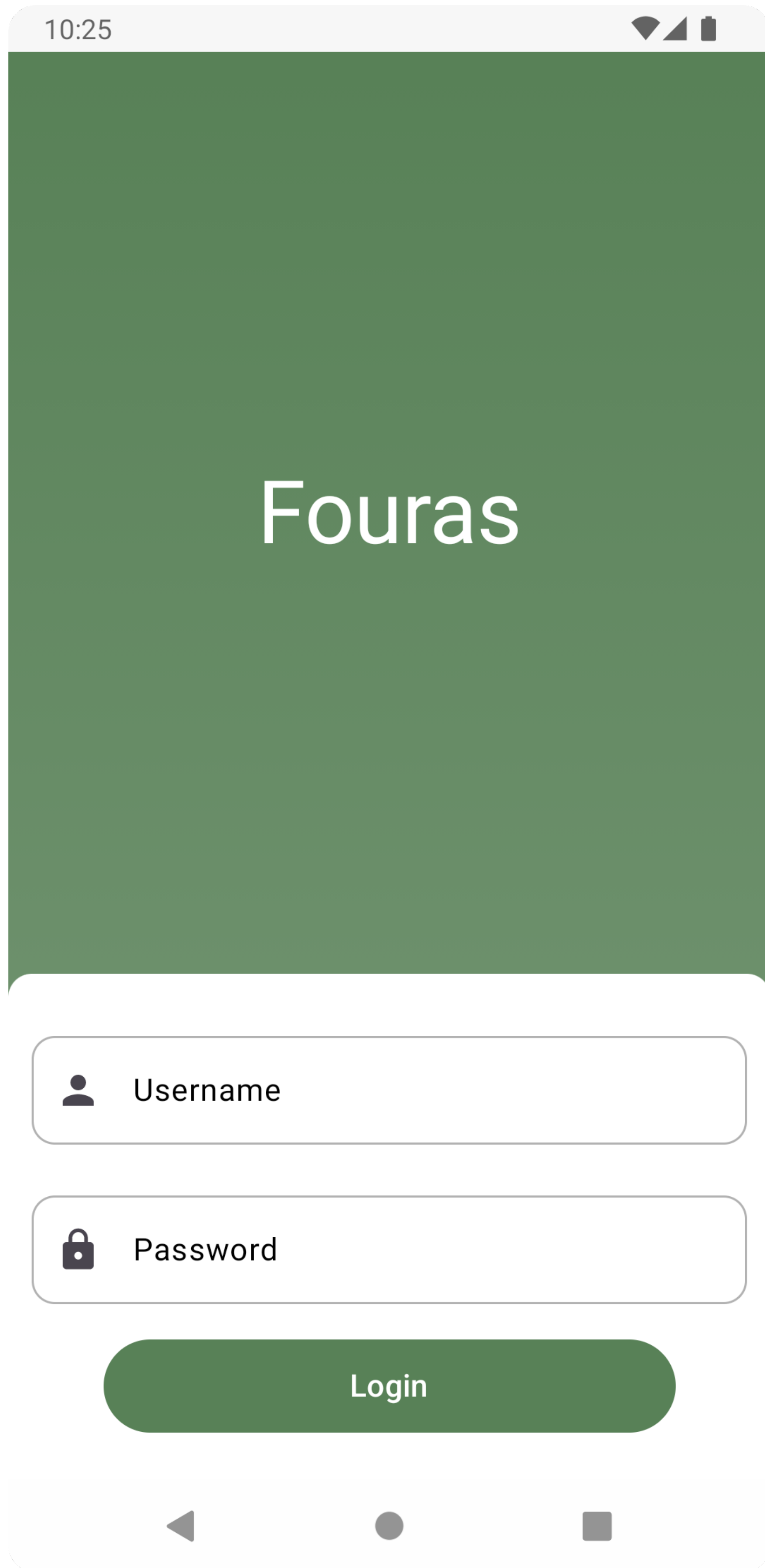

Figure A.1: Testing application authentication screen with credentials login

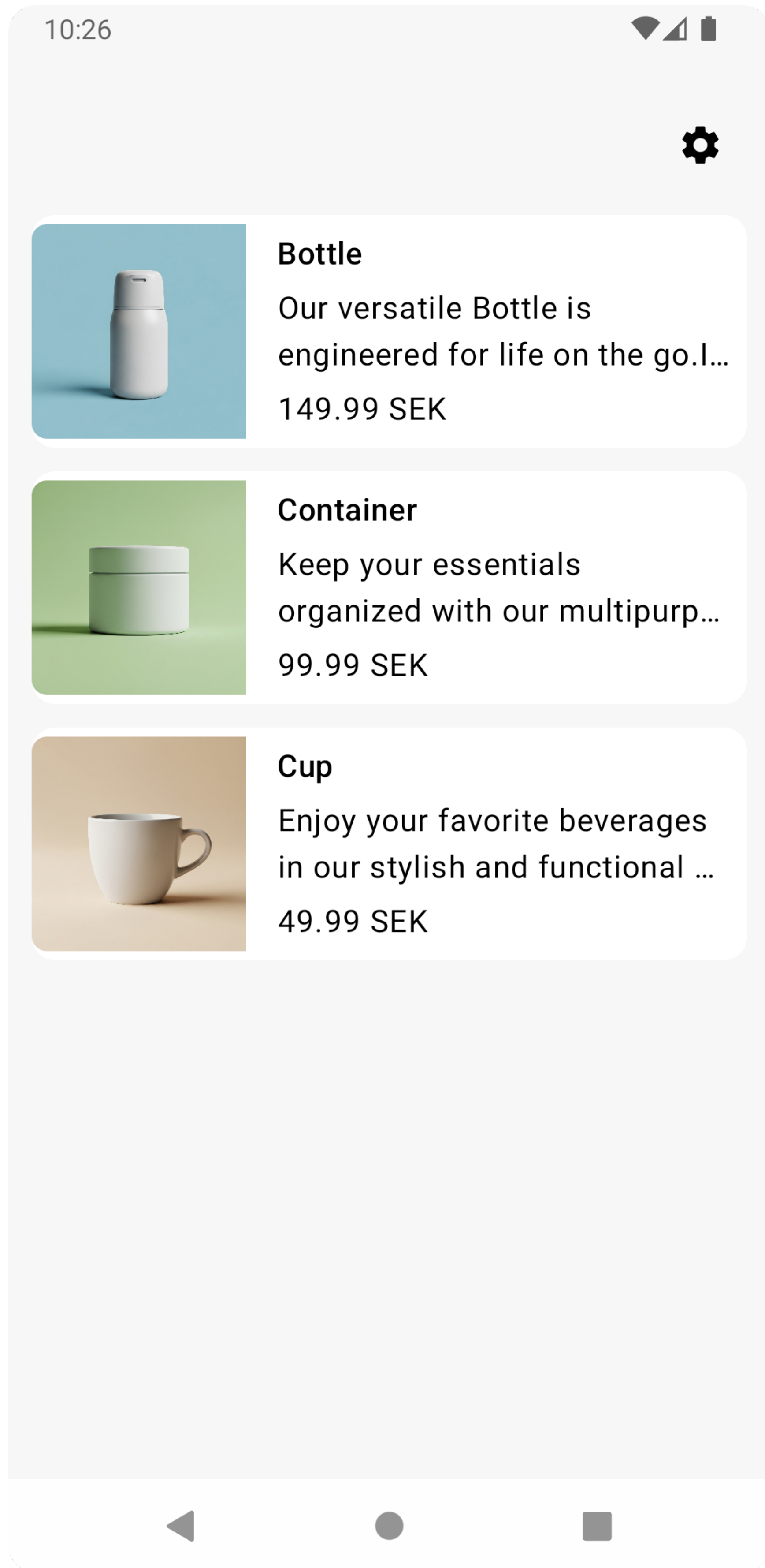

Figure A.2: Testing application overview screen with preset products

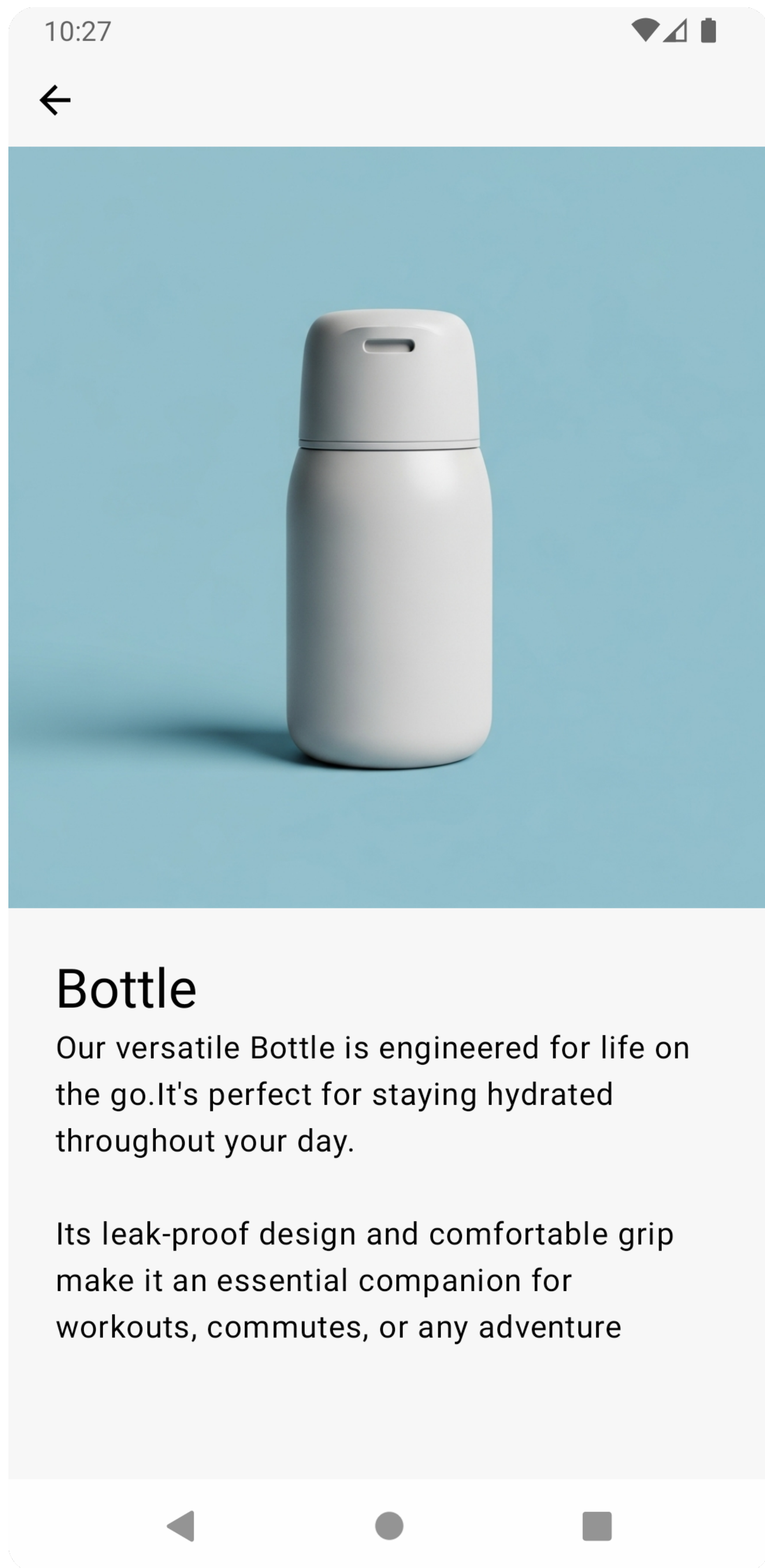

Figure A.3: Testing application product screen showcasing one of the products

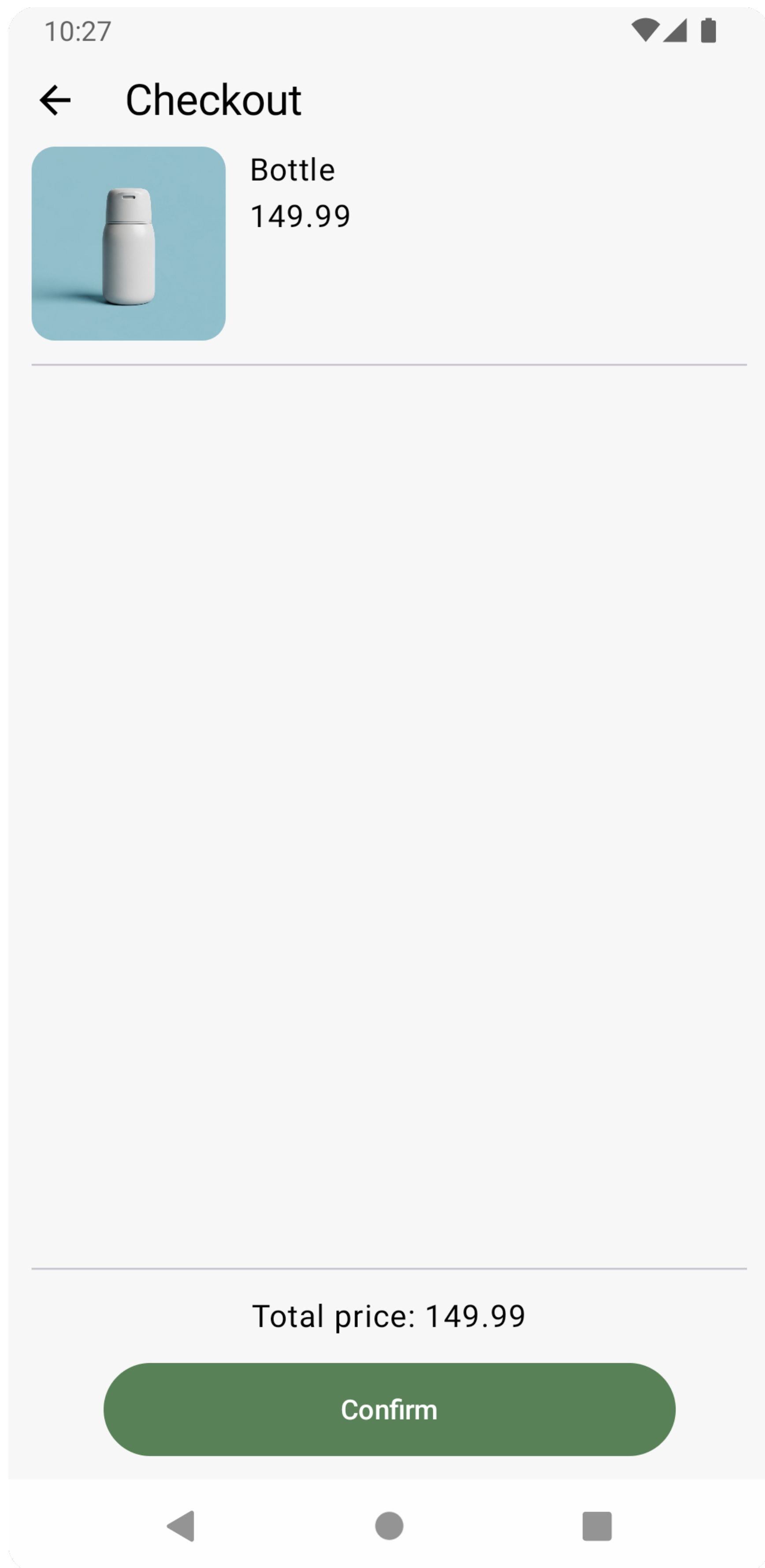

Figure A.4: Testing application cart screen showing two items to be purchased

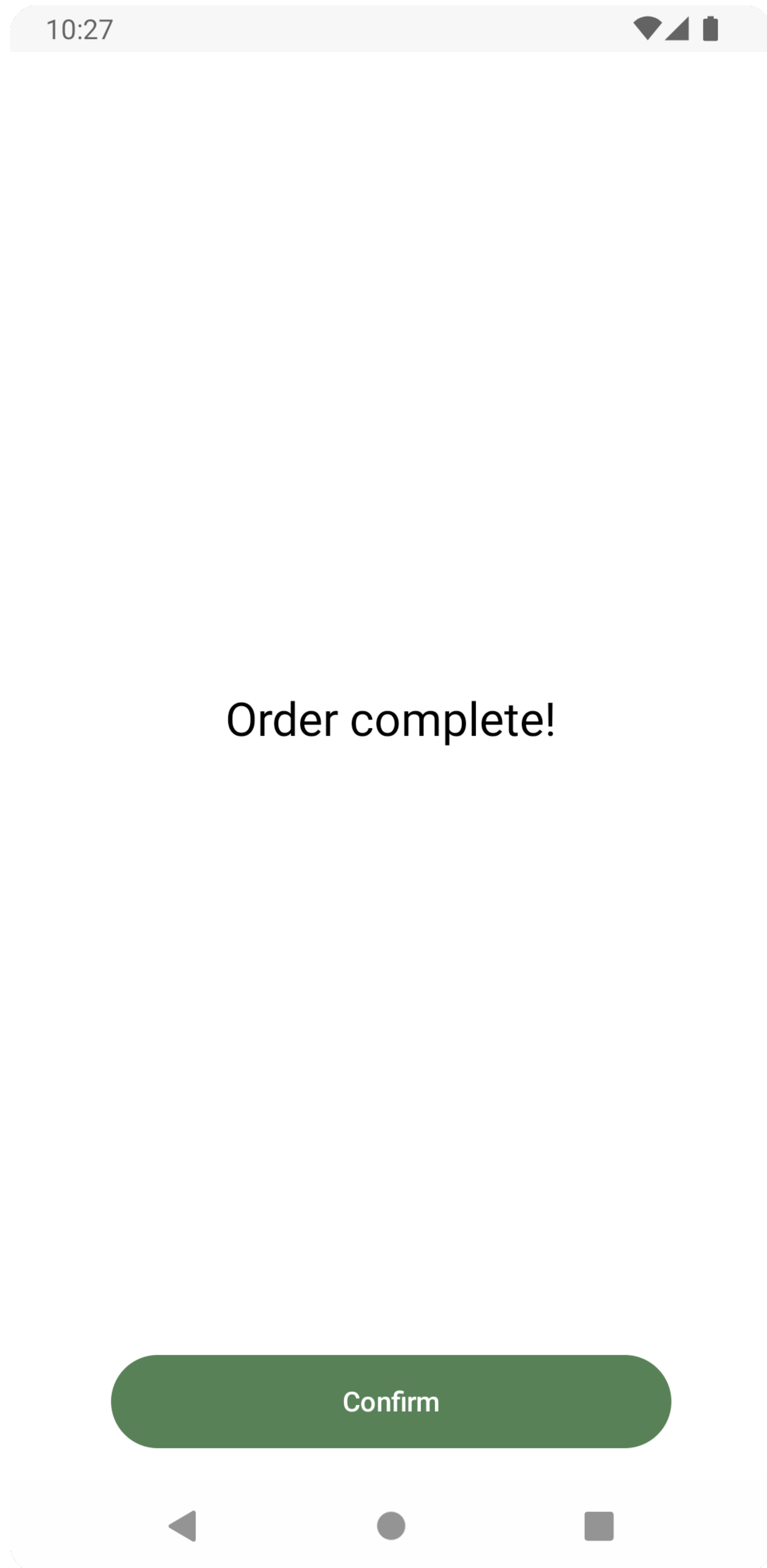

Figure A.5: Testing application receipt screen, indicating a successful purchase

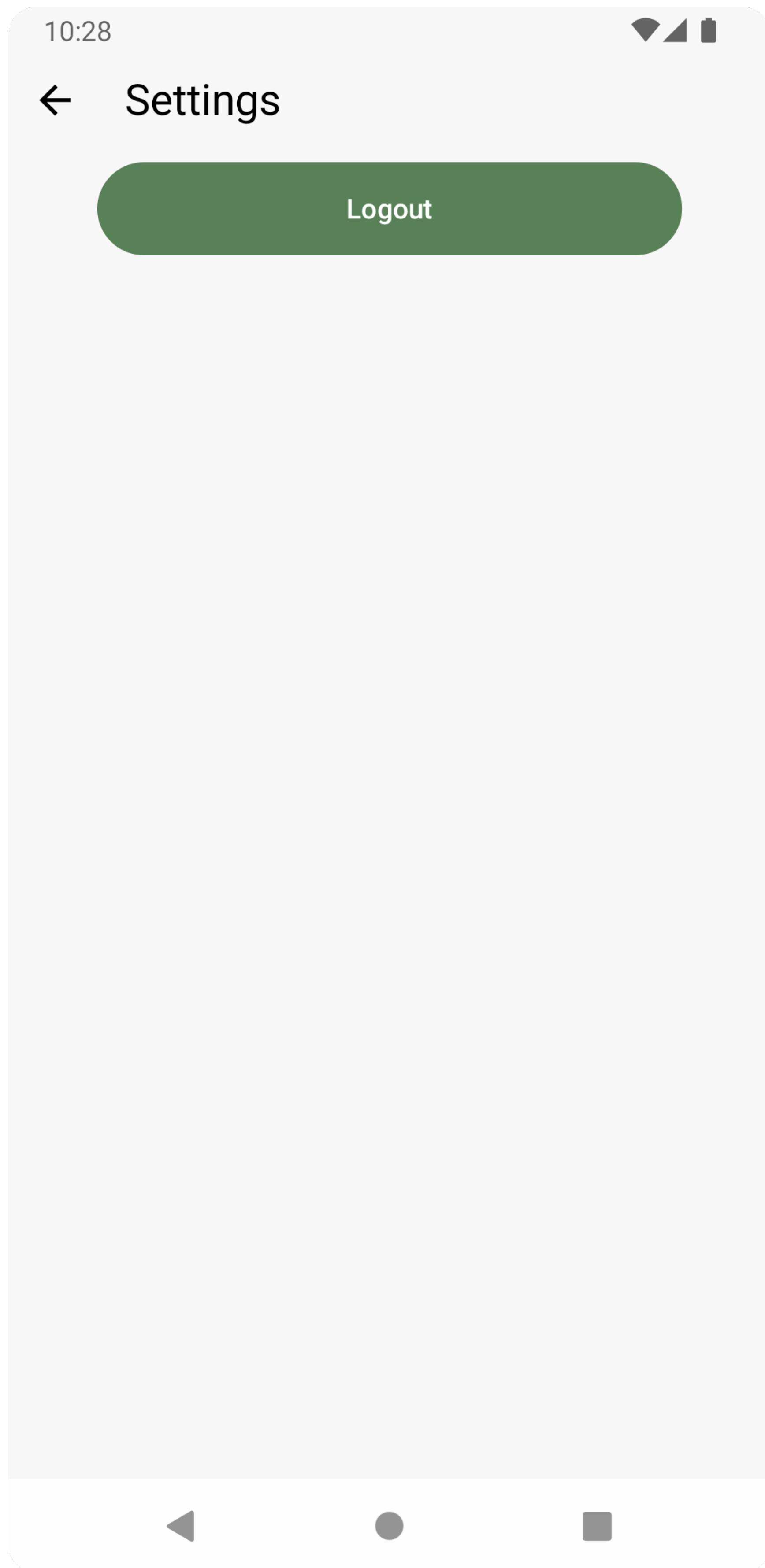

Figure A.6: Testing application profile view, allowing the user to logout

A.2 Cycle 2

A.2.1 Workshop structure

This is the structure we used for the group of developers to get feedback of our current implementations and modify them as needed for the second cycle.

A.2.1.1 Introduction

The European Accessibility Act will come into law in 2025, but there is seemingly a low adoption rate of accessibility features in Android applications. Current tooling like the Accessibility Scanner requires manual work on a compiled application. We have been creating a framework with the goal of ensuring that accessibility features are implemented by automated methods and/or systematic approaches.

A.2.1.2 Requirements (EN 301 549)

We will now inform you about the requirements from the European Union and what regulations will apply to Sweden. The Agency for Digital Government (DIGG) has specified that implementing the accessibility standard EN 301 549 will make an app compliant with upcoming regulations. This standard includes 162 requirements, some of which reference the World Wide Web Consortium's Web Content Accessibility Guidelines (WCAG) 2.1.

A.2.1.3 WCAG 2.1

The WCAG 2.1 guidelines are divided into three levels of compliance:

1. Level A: Basic requirements for accessibility.
2. Level AA: Addresses the biggest and most common barriers to impaired users.
3. Level AAA: The highest level of accessibility compliance.

To meet the EN 301 549 standard, an application must achieve Level AA compliance where applicable.

To provide an example of a level A requirement in WCAG 2.1. here is how non-text content requirement. Non-text content, such as images. are separated into different categories; decorative images and informative images. Decorative images should not have a description as they do not convey essential information. However, informative images must include a description of the content that the image depicts, such images include graphs or charts.

Our proposed solution to this requirement is to require developers to specifying whether images are decorative or informative instead of using the default image class. Linting tools is used to detect when a default image is used and prompt the developer to provide the appropriate implementation instead. This proactive approach ensures that all images meet accessibility standards during the development

process.

What do you think of this kind of opinionated composables approach?

Another requirement is ensuring that layered content provides a sufficient contrast between text and background colors. For normal text the ratio requires at least a 4.5:1 ratio, while larger texts requires a 3:1 ratio.

Our approach to this is to verify that the design system provides colors that offer a sufficient contrast ratio to begin with. In Android a common approach is to implement material design themes that specify the content and the color that can be placed atop that color, e.g. background and onBackground. If required, adding automated tests to verify the contrast ratios of the color theme can be created, but these types of tests should ideally be verified by designers rather than developers. Developers need to be vigilant in implementing the colors according to the design system. We did attempt to add lint checks to verify the contrast colors but that is seemingly complex given the structure of nestled composables.

What do you think of this kind design responsibility approach?