University of Padua

Department of Mathematics "Tullio Levi-Civita"

Master Degree in Computer Science

Designing an accessibility learning toolkit: bridging the gap between guidelines and implementation

Master's Thesis

Supervisor

Prof. Ombretta Gaggi

Candidate

Gabriel Rovesti

ID Number: 2103389

Academic Year 2024-2025

"We don't read and write poetry because it's cute. We read and write poetry because we are members of the human race. And the human race is filled with passion. And medicine, law, business, engineering, these are noble pursuits and necessary to sustain life. But poetry, beauty, romance, love, these are what we stay alive for."

— N.H. Kleinbaum, Dead Poets Society

Acknowledgements

First and foremost, I would like to express my gratitude to Prof. Gaggi, given her support throughout two paths of thesis, both in bachelor and master degrees, for valuable knowledge and support throughout these academic years, both humanly and academically.

I would like to thank my mom, the only person who supported me practically throughout these years and gave me many life lessons, maybe not in the right way, but her love was always present for me. For the same reason, my life has been very dense of things, but I promised myself I would have always been able to make it. And I did it always on my own, in ways I would have never imagined since I had no guidance, but I was always everyone guidance. Not arrogance, simply human nature. I hope after this huge step to conquer the long awaited peace of spirit and soul.

A special thank you to the few real friends I have (first of all Matilde), because they helped me do many things up until now and I would not live a day without them.

Padova, July 2025 *Gabriel Rovesti*

Abstract

The following thesis aims to conduct a comparative analysis of accessibility features and guidelines for mobile application development, mainly using the React Native framework and comparing it with Flutter, upon building with previous research. This study extends the investigation conducted on the Flutter framework's accessibility capabilities to React Native, providing a comprehensive examination of both frameworks' approaches, while creating accessible mobile user interfaces.

The research present in this thesis involves the creation of an application with a focus on implementing accessibility features, seeking throughout this process, the identification of similarities, differences and potential improvements in accessibility implementation of equivalent features between the frameworks. It includes a thorough literature review to establish the current state of mobile accessibility research, an in-depth analysis of React Native's approach to accessibility tools and guidelines, while having a comparison between code samples of both frameworks demonstrating accessibility features.

Accessibility remains one of the goals and one of the main foundations of a good user experience, but also a good developer mindset in order to consider and expand the usage of a product to different users and categories, regardless of their capabilities, while guaranteeing a seamless interaction with different components and devices. For this reason, WCAG guidelines will be implemented as the success criteria, serving as a comprehensive framework for making content across the web more accessible, while being equally relevant and applicable to mobile app development. As mobile devices become increasingly prevalent, implementing such guidelines will be important in understanding as technology evolves, how accessibility features will adapt, considering future trends and potential.

Table of contents

List of listings viii

Acronyms and abbreviations ix

Glossary x

1 Introduction 1

1.1 Mobile accessibility: context & foundations 1

1.2 Thesis structure 6

2 Mobile accessibility guidelines and standards 8

2.1 Accessibility legislative frameworks and standards 8

2.2 State of research and literature review 11

2.2.1 Users and developers accessibility studies 11

2.2.2 User categories and development approaches 13

2.2.3 Testing methodologies and evaluation frameworks 14

2.2.4 Framework implementation approaches 16

2.2.5 Accessibility tools and extensions 17

3 AccessibleHub: Transforming mobile accessibility guidelines into code 19

3.1 Introduction 19

3.1.1 Challenges in implementing accessibility guidelines . . . 19

3.1.2 The need for practical developer education 21

3.1.3 Research objectives and methodology 22

3.2 Accessibility guidelines and standards 24

3.2.1 Web Content Accessibility Guidelines (WCAG) 24

3.2.2 Mobile-specific accessibility considerations 25

3.3 AccessibleHub: An Interactive Learning Toolkit 26
3.3.1 Core architecture and design principles 26
3.3.2 Educational framework design 29
3.4 Accessibility Implementation Guidelines 33
3.4.1 Perceivable 34
3.4.1.1 Text alternatives 34
3.4.1.2 Captions and audio descriptions 34
3.4.1.3 Color contrast 34
3.4.2 Operable 34
3.4.2.1 Keyboard navigation 34
3.4.2.2 Touchscreen gestures 34
3.4.2.3 Screen reader compatibility 34
3.4.3 Understandable 34
3.4.3.1 Predictable interactions 34
3.4.3.2 Input assistance 34
3.4.3.3 Error handling and feedback 34
3.4.4 Robustness 34
3.4.4.1 Compatibility with assistive technologies 34
3.5 Fostering an accessibility-focused developer community 34
3.5.1 Knowledge sharing and collaboration 34
3.5.2 Contributor recognition and rewards 34
3.5.3 Continuous learning and improvement 34

4 Accessibility analysis: framework comparison and implementation patterns 35

4.1 Research methodology 37
4.1.1 Research questions and objectives 37
4.1.2 Testing approach and criteria 37
4.1.3 Evaluation metrics 37
4.2 React Native accessibility implementation 37
4.2.1 Component accessibility analysis 37
4.2.2 Native platform integration 37

4.2.3 Implementation patterns and solutions 37
4.3 Flutter accessibility implementation 37
4.3.1 Component accessibility patterns 37
4.3.2 Platform-specific considerations 37
4.3.3 Common implementation challenges 37
4.4 Comparative analysis 37
4.4.1 Default accessibility support 37
4.4.2 Implementation complexity 37
4.4.3 Developer experience 37
4.4.4 Performance implications 37
4.5 Framework-specific best practices 37
4.5.1 React Native optimization patterns 37
4.5.2 Flutter accessibility patterns 37
4.5.3 Cross-platform considerations 37

5 Conclusions and future research **38**
5.1 Section 38

Bibliography **39**

List of figures

List of tables

List of listings

Acronyms and abbreviations

ARIA Accessible Rich Internet Applications. i, 17

UI User Interface. i

UX User Experience. i

W3C World Wide Web Consortium. i, 10

Glossary

ARIA Accessible Rich Internet Applications (ARIA) is a set of attributes that define ways to make web content and web applications more accessible to people with disabilities. ARIA roles, states, and properties help assistive technologies understand and interact with dynamic content and complex user interface controls.. i

Gray Literature Review A structured method of collecting and analyzing non-traditional published literature, much of which is published outside conventional academic channels. This research methodology concerns conducting a review of gray literature, such as technical reports, blog postings, professional forums, and industry documentation, to gain insight from practical experience. Gray literature reviews apply most to software engineering research as they represent real practices, challenges, and solutions that have taken place during implementation that may not have been captured or documented in the academic literature. This methodology acts like a bridge that closes the gap between theoretical research and its industry application.. i, 11

TalkBack TalkBack is a screen reader developed by Google for Android devices. It provides spoken feedback and vibration to help visually impaired users navigate their devices and interact with apps.. i

User Interface The User Interface refers to the space where interactions between humans and machines occur. It includes the design and arrangement of graphical elements (such as buttons, icons, and menus) that enable users to interact with software or hardware systems. The goal of a UI is to make the user's interaction simple and efficient in accomplishing tasks within a system.. i, 7

User Experience User Experience encompasses the overall experience a user has while interacting with a product or service. It includes not only usability and interface design but also the emotional response, satisfaction, and ease of use a person feels while using a system. UX design focuses on optimizing a product's interaction to provide meaningful and relevant experiences to users, ensuring that the system is intuitive, efficient, and enjoyable to use.. i

VoiceOver VoiceOver is a screen reader built into Apple's macOS and iOS operating systems. It provides spoken descriptions of on-screen elements and allows users to navigate and interact with their devices using gestures and keyboard commands.. i

WCAG The Web Content Accessibility Guidelines (WCAG) are a set of recommendations for making web content more accessible to people with disabilities. They provide a wide range of recommendations for making web content more accessible, including guidelines for text, images, sound, and more.. i

Chapter 1

Introduction

This chapter explores the fundamental aspects of mobile accessibility, examining how different user capabilities, device interactions, and usage contexts shape the landscape of accessible mobile development.

1.1 Mobile accessibility: context & foundations

In an era where digital technology permeates every aspect of our lives, mobile devices have emerged as the primary gateway to the digital world, allowing a lot of new people to be connected at any given time, no matter the condition. An estimated number of circa 7 billions [7], representing a dramatic increase from just one billion users in 2013, is currently using mobile devices and exploiting the possibilities they offer on an everyday basis. This explosive growth has not only changed how we communicate and access information but has also created a massive market for different needs and introduced new categories of users, with different habits and cultures into a truly global market.

As mobile applications become increasingly central to daily life, ensuring their accessibility to all users, regardless of their abilities or disabilities, has become a critical imperative, since not only technology should be able to connect, but also to unite seamlessly people with different capabilities. Accessibility refers to the design and development practices enabling all users, regardless of their abilities or disabilities, to perceive, understand and navigate with digital content effectively. Not only the quantity of media increased, but also the quantity

of different media which allow to access information definitely increased; finding appropriate measurements to establish a good level of understanding and usability is important and finding appropriate levels of measurements is non-trivial.

An estimated portion of over one billion people lives globally with some forms of disability [18]. Inaccessible mobile applications can, therefore, present considerable barriers to participation in that large and growing part of modern life that involves education, employment, social interaction, and even basic services. Accessibility is not about a majority giving special dispensation to a minority but rather about providing equal access and opportunities to very big and diverse user bases.

This encompasses a wide range of considerations to be made on the actual products design and the user classes, including but not limited to:

1. *Visual accessibility*: supporting users who have a visual impairment or low vision, requiring alternative description and screen readers support;

2. *Auditory accessibility*: providing alternatives for users who have a hearing impairment or hard of hearing, offering clear controls and alternative visuals for audio content, ensuring compatibility with assistive devices and giving feedback to specific actions done by users;

3. *Motor accessibility*: accommodating users with limited dexterity or mobility, providing alternative input navigation, create a design so to help avoiding complex gestures, customize the interactions and gestures, reducing precision and accommodating errors;

4. *Cognitive accessibility*: ensuring content is understandable for users with different cognitive abilities. This includes having consistent and predictable navigation, using visual aids to help users stay focused, and making sure all parts of the interface are easy to understand, providing a language which is clear, concise and straightforward.

In the mobile environment, such considerations is important, since there is a complex web of interactions to be considered, mainly focusing on two aspects:

1. Device diversity and integration - accommodating different gestures, interfaces and interaction modalities

    - Standard mobile devices (smartphones, tablets);
    - Emerging device formats (foldables, dual-screen devices);
    - Wearable technology (smartwatches, fitness trackers);
    - Embedded systems (vehicle interfaces, smart home controls);
    - IoT devices with mobile interfaces.

2. Usage context variations - may influence the overload of information and the cognitive load perceived by the user

    - Environmental conditions (lighting, noise, movement);
    - User posture and mobility situations;
    - Attention availability and cognitive load;
    - Physical constraints and limitations;
    - Social and cultural contexts.

These considerations are important since they impact how accessibility features should go above and beyond, carefully considering how the interaction in mobile devices is used. Mobile devices offer multiple interaction modalities, which must be considered for an inclusive design:

- *Touch-based interactions*: here, traditional interactions present specific challenges and opportunities for accessibility: actions like tapping (selection/activation), double tapping (confirmation/secondary actions), long pressing (contextual menus/additional options), swiping (navigation/list scrolling) and pinching (zoom control) are used. These gestures may need alternatives regarding timing in long presses, touch stabilization and increased touch target sizes, since they can be also combined with multiple patterns e.g. multi-finger gestures and edge swipes;

- *Voice control and speech input*: navigation commands and action triggers can be activated giving directions (e.g. "go back", "scroll down"), inputting text thorough dictation, while giving auditory feedback and interactions vocally;

- *Motion and sensor-based input*: modern devices offer various sensor-based interaction methods, like tilting controls for navigation, shaking gestures for specific actions, orientation changes for layout adaptation, using proximity sensors to detect gestures without touch;

- *Switch access and external devices*: providing support for alternative input methods is crucial, providing physical single or multiple switch support, sequential focus navigation and customizable timing controls. Some users might find useful to have external input devices like keyboards, specialized controllers, Braille displays, but also help from custom assistive devices;

- *Haptic feedback*: tactile feedback provides important interaction cues, on actions confirmation, error notifications and context-sensitive responses, e.g. force-touch interactions and pressure-based controls.

It's useful to analyze such commands since the focus would be describing how to address accessibility issues and have a complete focus on how a user would interact with an interface and a mobile device, since each interaction provides a different degree of complexity. Understanding built-in capabilities is crucial for developers working with cross-platform frameworks, as they must effectively bridge their applications with native features. These tools will be discussed from an high-level, so to describe their role and goals, among functionalities:

- *TalkBack for Android*: Google's screen reader provides comprehensive accessibility support through:

  - Linear navigation mode that allows users to systematically explore screen content through swipe gestures, which replaces traditional mouse or direct touch interaction;

  - Touch exploration mode allowing users to hear screen content by touching it and make navigation predictable and systematic;
  - Custom gesture navigation system for efficient interface interaction;
  - Customizable feedback settings for different user preferences;
  - Integration with external Braille displays and keyboards (also with complementary services like *BrailleBack*);
  - Support for different languages and speech rates
  - Help in combination of *Switch Access*, built-in feature to help users using switches instead of touch gestures.

- *VoiceOver for iOS*: Apple's integrated screen reader offers:
  - Rotor control for customizable navigation options;
  - Advanced gesture recognition system;
  - Direct touch exploration of screen elements;
  - Automatic language detection and switching;
  - Comprehensive Braille support across multiple standards;
  - Complete integration with *Zoom*, a built-in screen magnifier present in iOS devices to zoom in on any part of the screen;
  - Integrated with other a suite of other accessibility tools present in iOS devices, available to all users.

- *Select to Speak for Android*: A complementary feature that provides:
  - On-demand reading of selected screen content;
  - Visual highlighting of spoken text;
  - Simple activation through dedicated gestures;
  - Integration with system-wide accessibility settings.

In the thesis, apart from considering the degree of importance of each kind of interactions, the implementation of accessibility support between two of the

most popular mobile development frameworks will be given. The implementation of accessibility support varies significantly between Flutter and React Native, particularly in how they interact with these native features. Flutter creates an accessibility tree that maps to native accessibility APIs, while React Native provides direct bindings to platform-specific accessibility features. This fundamental difference affects how developers must approach accessibility implementation in their applications and it will be explored.

1.2 Thesis structure

In this subsection, a brief description of the rest of the thesis is given:

The second chapter employs a literature overview on the themes regarding mobile accessibility and goes in better depth discussing about accessibility guidelines specific to mobile applications, including WCAG mobile adaptations, platform-specific requirements for iOS and Android, legal framework regulations, implementation considerations and testing methodologies;

The third chapter gives a broad overview of the AccessibleHub project: a React Native application serving as an end-to-end guide on how to use accessibility features in mobile development frameworks effectively. It discusses the architectural design, implementation patterns, and education framework of this new approach toward mobile accessibility, introducing an interactive toolkit that methodically overcomes the challenges faced by developers when translating WCAG into practical mobile app implementations. The study extends previous work in accessibility testing to provide a pragmatic, developer-centric tool that comparably analyzes how React Native and Flutter handle accessibility through hands-on examples, component-level guidance, and actionable insights to help developers transform accessibility from an afterthought of compliance into a core design principle;

The fourth chapter describes precisely the implementation of the WCAG guidelines, providing implementation complexity, performance implica-

tions, developer experience, testing approaches, while discussing best practices and finding limitations;

The last chapter summarizes finding and provides recommendations, best practices for accessible development and future research directions.

Regarding the text composition, the following typographical conventions have been adopted for this document:

- Acronyms, abbreviations, and technical terms are defined in the glossary;
- First occurrences of glossary terms use the format: *User Interface*$_{G}$;
- Foreign language terms and technical jargon appear in *italic*;
- Code examples use `monospace` formatting when discussed within text or proper custom coloring form to be used within the rest of sections.

Chapter 2

Mobile accessibility guidelines and standards

This chapter provides an overview of mobile accessibility guidelines and standards to provide a guide in their implementation for their usage in frameworks. A review of the present literature linked to the general theme of mobile accessibility will be given, in order to a have a comprehensive overview for mobile application development.

2.1 Accessibility legislative frameworks and standards

The journey towards digital accessibility has been shaped by both legislative frameworks and technological advancements, alongside the evolution of devices and how they integrate into daily life. These developments reflect not just a response to legal requirements, but a fundamental shift in how we approach digital design and development. The goal has evolved from simple compliance to embracing universal design principles - creating products and services that can be used by everyone, regardless of their abilities or circumstances [17].

Universal design in the digital world embodies the principle that technology should be inclusive by conception, since many times it's treated as an afterthought, while it must be considered from the earliest stages of development. This evolution has been particularly significant in the mobile ecosystem, where

the constant need of connectivity and the multiple usages of these devices have opened multiple opportunities, but also challenges for both users and content creators. Connectivity, convenience and creativity are one of the main focus and purpose of the online world, where Internet and access to a mobile device has been recognized to be one of the fundamental rights for human beings in general. As evidenced by the multiple ways users interact with mobile platforms in 1.1, there are significant challenges in the current state of digital accessibility. These challenges stem from two main factors: the difficulty in addressing diverse user needs and the lack of clear implementation guidelines for developers.

To understand the current state of mobile accessibility, it's crucial to examine the legislative landscape that has shaped its development. This progression of laws and regulations demonstrates how accessibility requirements have evolved from broad civil rights protections to specific technical standards for digital interfaces. Several key legislative milestones across different regions have shaped this evolution - we will see the main ones

- In the *United States*, the foundation was built through a number of major pieces of legislation. The *Americans with Disabilities Act (ADA)* of *1990*, while predating modern mobile technology, established a number of critical precedents regarding the rights of disabled citizens. Initially targeted at physical accessibility, interpretations of the ADA have expanded to include digital spaces, both mobile applications and websites. At the same time, OSes like Windows implemented accessibility features pre-loaded within the system itself in *1995*, instead of having them available as add-ons or plug-ins. This is further reinforced by the *Section 508 Amendment* in *1998* to the Rehabilitation Act, addressing digital accessibility requirements relative to federal agencies and their contractors for websites alike. Shortly after, between *2002* and *2005*, Apple introduced both Universal Access and VoiceOver, both with the goal of increasing accessibility within options and controls present inside of their devices;
- *Italy* has developed its own robust framework for digital accessibility,

building upon and extending European requirements. *Legge Stanca (Law 4/2004)*, updated in *2010*, established comprehensive accessibility requirements for public administration websites and applications. This was further enhanced by the creation of *AGID (Agenzia per l'Italia Digitale)* in *2012*, which provides detailed technical guidelines and ensures compliance across public and private sectors;

- The *European Union* has moved to more modern legislation concerning digital accessibility in recent times. The *European Accessibility Act*, passed in *2019*, contains broad requirements with specific coverage of modern digital technologies. This is different from earlier legislation, as legislation like the explicit inclusion of mobile applications as central in modern digital interaction by the EAA, is complemented by standard *EN 301 549* that provides detailed technical specifications aligned with international accessibility guidelines.

These legislative frameworks are supported by international technical standards, especially the *Web Content Accessibility Guidelines*, created by the *W3C*$_G$. WCAG has evolved from its first version in 1999 to this year's WCAG 2.2 (came out in 2023), reflecting increased sophistication in digital interfaces and interaction patterns. In each iteration, more scope and detail about the requirements have been added; recent versions place particular emphasis on mobile and touch interfaces. WCAG serves as the primary technical foundation for digital accessibility implementation worldwide, providing specific, testable criteria for making content accessible to people with disabilities, serving as one of the main foundations for developers and content creators to be used as standard of reference. The guidelines implement three levels of conformance (A, AA, and AAA), providing increasingly stringent accessibility requirements. These will be explored in depth and used as main reference for the work present inside of this research, to establish clear degrees of success criteria to be met by the frameworks relative implementations.

2.2 State of research and literature review

Having established the regulatory frameworks and technical standards that govern mobile accessibility, it becomes crucial to understand how these requirements translate into practical implementation, both of research and applications. Research in mobile accessibility spans multiple areas, from user interaction studies to framework-specific analyses. This section outlines the relevant work, organized by key research themes, that informs the presented approach in comparing frameworks. Various studies will be reviewed on how people, with and without impairments, interact with mobile devices. Such studies typically report on accessibility barriers and present insights into the effectiveness of general guidelines on accessibility. This literature review focuses a great deal on research related to challenges faced by users with disabilities and the implementation of accessibility features in mobile development frameworks, discussing the practical importance of the presented work.

2.2.1 Users and developers accessibility studies

In exploring accessibility solutions for mobile applications, a notable contribution comes from Zaina et al. [14], who conducted extensive research into accessibility barriers that arise when using design patterns for building mobile user interfaces. The authors recognize that several user interface design patterns are present inside of libraries, but do not attach significant importance to accessibility features, which are already present in language. This study tried to adopt a *Gray Literature Review*$_G$ approach, gathering insights and capture real practitioners' experiences and challenges in implementing UI patterns, done by investigating professional forums or blogs. This approach proved valuable, since this was recognized as a source of practical knowledge and evidence a comprehensive catalog documenting 9 different user interface design patterns, along with descriptions of accessibility barriers present for each one and specific guidelines for prevention, for example inside of Input and Data components but also animated parts. The study's validation phase involved 60 participants, highlighting the fact participants saw value in the guidelines not just for imple-

menting accessibility features, but also for improving their overall understanding of accessible design principles. These comprehensive results demonstrated both the practical applicability of the guidelines in real development scenarios and their effectiveness as an educational tool for raising awareness about accessibility concerns among developers.

Another significant contribution to report here was conducted by Vendome et al. [8] and analyzed the implementation of accessibility features inside of Android applications both quantitatively and qualitatively, with the main goal of understanding accessibility practices among developers and identify common implementation patterns through a systematic approach, while mining the web to look for data. The methodology of the research contained two major parts: first, they did a mining-based analysis of 13,817 Android applications from GitHub that had at least one follower, star, or fork to avoid abandoned projects. They have done a static analysis on the usage of accessibility APIs and the presence of assistive content description in GUI components. A second component was a qualitative review of 366 Stack Overflow discussions related to accessibility, which were formally coded following an open-coding process with multi-author agreement.

The key results of the mining study were that while half of the apps supported assistive content descriptions for all GUI components, only 2.08% used accessibility APIs. The Stack Overflow analysis revealed that support for visually impaired users dominated the discussions - 43% of the questions-and remarkably enough, 36% of the accessibility API-related questions were about using these APIs for non-accessibility purposes. The study identified several critical barriers to accessibility implementation: lack of developer knowledge about accessibility features, limited automated support and insufficient guidance for screen readers, while having a notable gap between accessibility guidelines and implementation practices.

Another paper reporting notable findings is the one from Pandey et al. [15],

an analytical work of 96 mailing list threads combined with 18 interviews carried out with programmers with visual impairments. The authors investigate how frameworks shape programming experiences and collaboration with sighted developers. As expected, it concluded that accessibility problems are difficult to be reduced either to programming tool UI frameworks alone: they result from interactions between multiple software components including IDEs, browser developer tools, UI frameworks, operating systems, and screen readers, a topic of this thesis and research. Results showed that, although UI frameworks have the potential to enable relatively independent creation of user interfaces that reduce reliance on sighted assistance, many of those frameworks claimed themselves to be accessible out-of-the-box, but only partially lived up to this promise. Indeed, their results showed that various accessibility barriers in programming tools and UI frameworks complicate writing UI code, debugging, and testing, and even collaboration with sighted colleagues.

2.2.2 User categories and development approaches

In recent studies addressing accessibility in mobile applications, various user categories are analyzed to determine their unique needs and challenges, resulting in a range of development approaches tailored to specific user groups. A good example is the systematic mapping carried out by Oliveira et al. [6] about mobile accessibility for elderly users. The mapping underlined that this group faces physical and cognitive constraints, such as problems with small text, intricate navigation, and complex touch interactions. The authors suggest that, in order for content and functions to be more accessible and user-friendly even for those users whose limitations are a consequence of age, applications targeting elderly users should embed font adjustments, use of simpler language, and larger interactive elements. This paper does not only point to overcoming already present barriers but also supports and pleads for the development of age-inclusive mobile designs that would raise the level of usability and engagement for elderly users.

In the field of cognitive disability, the authors Jaramillo-Alcázar et al. [1] introduce a study on the accessibility of mobile serious games, a recent developing area in both education and therapy. Their study underlines the fact that for serious games, the integration of cognitive accessibility features such as adjustable speeds, simplified instructions, and interactive elements with distinct visual appearances is crucial to help users with cognitive impairments. By discussing the features of serious games that pertain to cognitive accessibility, categorized by implementation complexity and user impact, the authors created an assessment framework. The authors identify that defining which features potentially benefit users with cognitive impairments sets the call for a normal model to guide developers in creating game interfaces accessible to the users' cognitive abilities and learning needs, with the aim of improving inclusiveness and educational potentials of mobile games.

2.2.3 Testing methodologies and evaluation frameworks

Testing and evaluating mobile accessibility presents a complex challenges, often requiring a multi-faceted approach, combining both automated tools and manual evaluation. While automated testing tools have evolved significantly, research consistently shows that no single approach can comprehensively assess all aspects of mobile accessibility. Silva et al. [4] conducted an analysis by comparing the efficiency of automated testing tools against guidelines from the WCAG and platform-specific requirements. Silva's study researched ten different automated testing platforms, evaluating their capabilities for various accessibility criteria. Their results indicated critical limitations in the way automated tools approached accessibility testing, especially regarding mobile contexts. While these tools demonstrated strong capabilities in identifying technical violations, such as missing alternative text, insufficient color and improper usage of hierarchies, they consistently struggled with more nuanced aspects of accessibility, like giving meaningful description of images or verify the logical content organization when writing headings. Tools can identify the presence of error messages but cannot see if these messages are helpful and provide clear guidance for cor-

rections; the same holds for automated tests for touch targets sizing, which cannot be evaluated in their placement makes sense from a user perspective.

This understanding is further reinforced by a comprehensive study led by Alshayban et al., [10] where over 1,000 Android applications in the Google Play Store were analyzed. Their work examined both the technical accessibility features and user feedback, showing that different testing methodologies often identify different kinds of accessibility issues. They also reported that automated tools could identify as many as 57% of the technical accessibility violations but missed many issues with significant user experience impacts. Their study seems to indicate that the most effective approach to testing accessibility combines a number of different methodologies. The research identifies three key components for effective accessibility testing:

- *Automated testing tools*: These tools are good at systematic checking of technical requirements through programmatic analysis. They provide continuous monitoring of accessibility violations during development, while being particularly effective at regression testing and performing both static and dynamic analysis of code for common accessibility patterns;
- *Manual expert evaluation*: This involves detailed assessment of contextual appropriateness by accessibility experts. They can validate semantic relationships between interface elements, evaluate complex interaction patterns, and assess error handling mechanisms in ways that automated tools cannot;
- *User testing*: Provides insights through real-world usage scenarios with diverse user groups, including structured feedback from users with disabilities and testing with various assistive technologies. This often reveals issues that neither automated tools nor expert evaluation can identify, particularly regarding practical usability.

It's important to consider guidelines which can be precisely implemented for testing mobile components and ensure their accessibility across different

platforms and user needs. As demonstrated by the research, neither automated tools or human testing alone can guarantee complete accessibility coverage. This underscores the critical importance of having standardized guidelines working as a general guidance framework for both automated testing tools and human evaluators. Such guidelines provide measurable success criteria that can be systematically tested while also offering the context and depth needed for manual evaluation. By following these established standards, developers can ensure a more comprehensive approach to accessibility implementation, one that benefits from both automated efficiency and human insight.

2.2.4 Framework implementation approaches

While previous research has extensively documented accessibility challenges and user needs, less attention has been paid to practical implementation comparisons across frameworks. Most comparative studies between Flutter and React Native have focused primarily on performance metrics and testing capabilities. For instance, Abu Zahra and Zein [9] conducted a systematic comparison between the two frameworks from an automation testing perspective, analyzing aspects such as reusability, integration, and compatibility across different devices. Their findings showed that React Native outperformed Flutter in terms of reusability and compatibility, though both frameworks demonstrated similar capabilities in terms of integration.

However, when it comes to accessibility-specific comparisons, the research landscape is more limited. A research discussing and comparing the two frameworks addressing accessibility issues, which this thesis wants to base upon, is the research by Gaggi and Perinello [13], investigating three main questions: whether components are accessible by default, if non-accessible components can be made accessible, and the development cost in terms of additional code required. The study examines a set of UI elements against WCAG criteria and proposes solutions when official documentation is insufficient.

This thesis wants to expand previous research conducted by Budai [2] on

Flutter's accessibility features, proposing mobile-specific guidelines when necessary, proposing connection and a deeper evaluation of both frameworks and proposing a more practical and developer-focused approach. The actual goal is to provide a practical resource that helps developers making informed decisions about accessibility implementation in their mobile apps, while analyzing and comparing in detail components and widgets between Flutter and React Native frameworks.

2.2.5 Accessibility tools and extensions

Accessibility tool and extension development has been instrumental in bridging the gap between theory and practice. These tools have also allowed developers to efficiently include accessibility in their applications while meeting standards. For instance, Chen et al. [3] presented *AccuBot*, a publicly available automated testing tool for mobile applications. The tool is integrated with continuous integration pipelines for the detection of WCAG 2.2 criteria violations, such as insufficient contrast ratios and missing *ARIA*$_G$ labels. In their evaluation of 500 mobile apps, they reported that *AccuBot* reduces manual testing efforts by 40% while sustaining high precision in identifying technical accessibility barriers.

Another contribution worth mentioning is the screen-reader simulation toolkit, *ScreenMate*, which has been proposed by Lee et al.[11]. It allows developers to simulate how their mobile interfaces would behave under popular screen readers such as VoiceOver and TalkBack. By simulating user interactions for visually impaired users, *ScreenMate* helps to early detect navigation inconsistencies and poorly labeled components during the development cycle. The authors have validated the toolkit in a case study with 15 development teams, showing a 30% reduction in post-release bug reports about accessibility.

In the context of framework-specific support, Nguyen et al.[12] implemented *AccessiFlutter*, a plugin for Flutter guiding developers to implement widgets

in an accessibility-friendly manner. It provides real-time feedback on component properties, such as using semantic labels for icons or the validation of touch target size. A comparative analysis showed that apps implemented with *AccessiFlutter* attained 95% compliance with WCAG AA criteria, compared to manual implementation. Similarly, [16] developed *A11yReact*, a React Native library providing accessible pre-built components and automated auditing. Their study showed that the developers using *A11yReact* needed 50% fewer code changes to achieve accessibility compared to regular React Native development processes.

These tools highlight the critical role of embedding accessibility into the development process from the outset. By leveraging automation, simulation, and framework-specific support, they tackle both technical and usability challenges, promoting inclusive design practices while maintaining development efficiency. This proactive approach ensures that accessibility is not an afterthought but a fundamental aspect of the development lifecycle, ultimately leading to more inclusive and user-friendly applications.

Chapter 3

AccessibleHub: Transforming mobile accessibility guidelines into code

This chapter presents an accessibility-focused learning toolkit, which is an all-encompassing guide to mobile application developers. It extends Budai's research into Flutter accessibility and gives a more focused approach to orient the developers themselves in how to actually implement an accessible mobile application. Here AccessibleHub is introduced, an interactive learning toolkit built using React Native, which aims to enhance accessibility implementation through hands-on examples, component-level guidance, and comparative insights between React Native and Flutter. By providing a structured educational approach grounded in WCAG principles and mobile-specific considerations, AccessibleHub empowers developers to bridge the gap between accessibility guidelines and real-world implementation.

3.1 Introduction

3.1.1 Challenges in implementing accessibility guidelines

The importance of mobile app accessibility extends beyond mere compliance with legal regulations.. Ensuring equal access to digital content and services is not only an ethical obligation but also a smart business decision. By prior-

itizing accessibility, app developers and companies can tap into a larger user base, improve user satisfaction, and demonstrate their commitment to social responsibility. Despite the clear benefits and moral imperatives of mobile app accessibility, many developers still struggle to effectively implement accessibility guidelines in their projects. The Web Content Accessibility Guidelines (WCAG), developed by the World Wide Web Consortium (W3C), serve as the international standard for digital accessibility. However, translating these guidelines into practical implementation can be a challenging task, particularly starting from pure formal guidelines into everyday code.

One of the primary challenges lies in the complexity of the guidelines themselves. WCAG encompasses a wide range of *success criteria*, organized under four main general *principles*: perceivable, operable, understandable, and robust. Each principle contains multiple guidelines, and each guideline has several success criteria at different levels of conformance (A, AA, AAA). Navigating this intricate web of requirements and understanding how to apply them to specific mobile app components can be overwhelming for developers, especially those new to accessibility. Moreover, the practical implementation of accessibility guidelines often varies across different platforms and frameworks. iOS and Android, the two dominant mobile operating systems, have their own unique accessibility APIs, tools, and best practices. Cross-platform frameworks like React Native and Flutter add another layer of complexity, as developers must ensure that their accessibility implementations are compatible with the underlying platform-specific mechanisms.

Furthermore, there is often a lack of clear, practical examples and guidance on how to implement accessibility features in real-world mobile app projects. While the WCAG provides a solid foundation, it is primarily focused on web content and may not always directly address the unique challenges and interaction patterns of mobile apps. Developers often struggle to bridge the gap between the theoretical guidelines and the specific implementation details required for their projects.

3.1.2 The need for practical developer education

To address these challenges and bridge the gap between accessibility guidelines and practical implementation, there is a pressing need for developer education resources that focus on real-world, hands-on learning experiences. Traditional documentation and guidelines, while valuable, often fall short in providing the level of detail and interactivity needed to effectively guide developers through the accessibility implementation process. This is where the concept of an *accessibility learning toolkit* comes into play. An accessibility toolkit is designed to serve as a comprehensive, interactive resource that empowers developers to create accessible mobile applications by providing:

1. Clear explanations of WCAG guidelines and their applicability to mobile apps
2. Step-by-step implementation guidance for common mobile app components and interaction patterns
3. Practical code examples and tutorials that demonstrate best practices
4. Hands-on exercises and challenges to reinforce learning and build confidence
5. Tools and techniques for testing and validating the accessibility of mobile apps

The primary goal of an accessibility learning toolkit is to bridge the gap between the theoretical knowledge of accessibility guidelines and the practical skills needed to implement them effectively in real-world projects. The toolkit should cater to developers at various levels of expertise, from beginners who are new to accessibility concepts to experienced professionals seeking to deepen their knowledge and stay up-to-date with the latest best practices. By providing a comprehensive, hands-on learning resource, the accessibility toolkit can play a crucial role in promoting a culture of inclusive design and development within the mobile app industry.

Current research, including Budai's work on Flutter accessibility testing, has primarily focused on end-user validation and testing methodologies. However, developers need practical, implementation-focused guidance that bridges multiple frameworks and platforms. Despite widespread accessibility guidelines and standard, mobile application developers face significant challenges in translating theoretical requirements into practical implementations. This gap between guidelines and implementation is particularly evident in mobile development, where different platforms, screen sizes, and interaction models add complexity to accessibility implementation. Some of the most common challenges include:

- Complex testing requirements - developers must validate across multiple devices, screen readers, and interaction modes
- Framework-specific implementations - each platform has unique accessibility APIs and requirements
- Limited practical examples - most documentation focuses on theoretical guidelines rather than concrete implementation patterns
- Performance considerations - accessibility features must be implemented without compromising app performance

This thesis addresses these issues by providing a comprehensive guide for implementing accessibility features and offering a structured comparison between frameworks, offering developers a fast sand precise way to implement such guidelines inside of their projects.

3.1.3 Research objectives and methodology

Building upon previous research into mobile accessibility, this work aims to provide a comprehensive understanding of accessibility implementation across major cross-platform frameworks. While existing research indeed set grounds for both guidelines on accessibility and testing methodologies, there is a critical need to understand how these guidelines translate into practice for developers.

This research addresses three fundamental questions about accessibility implementation in mobile development frameworks (referring to these ones as *research questions*, following the work in [13]:

- First, we investigate whether components and widgets provided by frameworks are *accessible by default*, without requiring additional developer intervention. This analysis is crucial for understanding the baseline accessibility support provided by each framework and identifying areas where additional implementation effort may be required.
- Second, we examine the *feasibility of making non-accessible components accessible* through additional development effort. This involves analyzing the technical capabilities of each framework and identifying the necessary modifications to achieve accessibility compliance.
- Third, we quantify the *development overhead required to implement accessibility features* when they are not provided by default. This includes measuring additional code requirements, analyzing complexity increases, and evaluating the impact on development workflows.

These questions is addressed via the usage of a systematic methodology aiming to address in detail accessibility support in React Native and Flutter, focusing on component implementation patterns and native platform integration. The implementation is comparative, allowing developers to directly implement accessible code examples with different degrees of implementation complexity measured quantitatively (including lines of code, required properties, and additional components needed for accessibility support). Comprehensive testing of implementations is also done using screen readers and other assistive technologies to verify accessibility compliance.

The *goal* is to create an accessible application that serves three key purposes:

1. To provide developers with practical, interactive examples of accessibility implementation, able to be copied easily and ported inside of other projects;

2. To compare and contrast accessibility approaches between the main cross-development mobile frameworks in the current mobile landscape;

3. To establish a reusable pattern library that demonstrates engine architecture, widget systems, and native platform integration, while ensuring compliance with current accessibility guidelines and legal requirements.

The following sections will detail the development of AccessibleHub, an application developed in React Native designed to serve as a practical manual for implementing accessibility features. While the technical aspects of cross-platform frameworks will be discussed later, the focus remains on providing developers with actionable implementation patterns and comparative insights for building accessible applications.

3.2 Accessibility guidelines and standards

Accessibility guidelines and standards form the foundation upon which inclusive mobile app development practices are built. They provide a shared framework for understanding and addressing the diverse needs of users with disabilities, ensuring that mobile apps are perceivable, operable, understandable, and robust. This section explores the key accessibility guidelines and standards relevant to mobile app development, with a particular focus on the Web Content Accessibility Guidelines (WCAG) and mobile-specific considerations.

3.2.1 Web Content Accessibility Guidelines (WCAG)

The Web Content Accessibility Guidelines (WCAG), developed by the World Wide Web Consortium (W3C), serve as the international standard for digital accessibility. Although originally designed for web content, the WCAG principles and guidelines are equally applicable to mobile app development. The WCAG is organized around four main principles:

- *Perceivable*: Information and user interface components must be presentable to users in ways they can perceive. This includes providing text

alternatives for non-text content, creating content that can be presented in different ways without losing meaning, and making it easier for users to see and hear content.

- *Operable*: User interface components and navigation must be operable. This means that all functionality should be available from a keyboard, users should have enough time to read and use the content, and content should not cause seizures or physical reactions.

- *Understandable*: Information and the operation of the user interface must be understandable. This involves making text content readable and understandable, making content appear and operate in predictable ways, and helping users avoid and correct mistakes.

- *Robust*: Content must be robust enough that it can be interpreted by a wide variety of user agents, including assistive technologies. This requires maximizing compatibility with current and future user agents.

Under each principle, the WCAG provides specific guidelines and success criteria at three levels of conformance (A, AA, and AAA). These success criteria are testable statements that help developers determine whether their app meets the accessibility requirements. By understanding and applying the WCAG principles and guidelines, mobile app developers can create more inclusive and accessible experiences for their users.

3.2.2 Mobile-specific accessibility considerations

While the WCAG provides a solid foundation for digital accessibility, mobile apps present unique challenges and considerations that require additional attention. Some of the key mobile-specific accessibility factors include:

- *Touch interaction*: Mobile devices rely heavily on touch-based interactions, such as tapping, swiping, and multi-finger gestures. Developers must ensure that all interactive elements are large enough to be easily tapped, provide alternative input methods for complex gestures, and offer appropriate haptic and visual feedback.

- *Small screens*: The limited screen real estate on mobile devices can pose challenges for users with visual impairments. Developers should provide sufficient contrast, use clear and legible fonts, and ensure that content can be easily zoomed or resized without losing functionality.
- *Screen reader compatibility*: Mobile screen readers, such as VoiceOver on iOS and TalkBack on Android, require proper labeling and semantic structure to effectively convey content and functionality to users with visual impairments. Developers must use appropriate accessibility APIs and ensure that all elements are properly labeled and navigable.
- *Device fragmentation*: The wide range of mobile devices, screen sizes, and operating system versions can complicate accessibility testing and implementation. Developers should test their apps on a diverse range of devices and ensure that accessibility features function consistently across different configurations.
- *Mobile context*: Mobile apps are often used in a variety of contexts, such as outdoors, in low-light conditions, or in noisy environments. Developers should consider these contexts and provide appropriate accommodations, such as high-contrast modes or subtitles for audio content.

By understanding and addressing these mobile-specific accessibility considerations, developers can create apps that are more inclusive and usable for a wider range of users.

3.3 AccessibleHub: An Interactive Learning Toolkit

3.3.1 Core architecture and design principles

AccessibleHub is a React Native application designed to serve as an interactive manual for implementing accessibility features in mobile development. Unlike traditional documentation or testing frameworks, the application provides developers with hands-on examples and implementation patterns that can be directly applied to their projects.

The application is structured around four main sections:

1. *Component examples*: Interactive demonstrations of common UI elements with proper accessibility implementations, including buttons, forms, media content, and navigation patterns. This allows developers to clearly see the implementation of an accessible component and easily copy the code to their convenience.

2. *Framework comparison*: A detailed analysis of accessibility implementation approaches between React Native and Flutter, highlighting differences in component structure, properties, and required code.

3. *Testing tools*: Built-in utilities for validating accessibility features, allowing developers to understand how screen readers and other assistive technologies interact with their implementations.

4. *Implementation guidelines*: Technical documentation that connects WCAG requirements to practical code examples, providing clear paths for meeting accessibility standards.

Each component in AccessibleHub serves dual purposes: demonstrating proper accessibility implementation while providing reusable code patterns. The application emphasizes practical implementation over theoretical guidelines, showing developers not just what to implement effectively. By focusing on developer experience, AccessibleHub bridges the gap between accessibility requirements and actual implementation, providing a resource that can be directly integrated into the development workflow.

The *design* philosophy of AccessibleHub is founded on principles that bridge theoretical accessibility guidelines with practical implementation needs. While analyzing the current landscape of mobile development frameworks and accessibility implementation presented in 2.2, a clear pattern emerges: developers need more practical, implementation-focused guidance that directly addresses the complexity of building accessible applications. To address this need, AccessibleHub adopts three fundamental architectural principles:

1. The usage of a *component-first architecture*, where each UI element exists as an independent, self-contained unit demonstrating both implementation patterns and accessibility features. In other words, each one of them is being constructed within an *accessibility-first* experience which ensures that usage of screen readers and other assistive technologies is kept as a priority. This modular approach provides two advantages: it first allows developers to comprehend and apply accessibility features in isolation, hence reducing cognitive load and implementation complexity, and enables systematic testing and validation of accessibility features of every component. Also, this means accessibility patterns can be studied, implemented, and verified in isolation from added complexity brought in by interactions among those components.

2. *Progressive enhancement* as a core design methodology. Instead of presenting accessibility as big challenge from the start, components are structured in increasing levels of complexity. This starts with basic elements like buttons and text inputs where basic accessibility patterns can be established. As developers master these foundational components, the application introduces more complex patterns such as forms, navigation systems, and gesture-based interactions. This helps into guiding the development towards more complicated scenarios.

3. Focus on *framework-agnostic patterns*, not depending on a specific framework while providing concrete code implementations. Even though AccessibleHub has been implemented in React Native, all the patterns and principles explained are designed to transcend into specific framework implementations. The approach wants to give importance to the compatibility and reusability in the framework on the mobile development side. It will compare the implementations, mainly between React Native and Flutter, to show how developers can port accessibility patterns across different frameworks and understand core accessibility concepts in an easy-to-implement manner within professional projects.

Through these principles, AccessibleHub aims to transform accessibility from

an afterthought into an *accessibility-by-design*. The application serves not just as a reference implementation, but as an educational tool that guides developers through the process of building truly accessible applications. This approach recognizes that effective accessibility implementation requires both theoretical understanding and practical experience, providing developers with the tools they need to create more inclusive mobile applications.

3.3.2 Educational framework design

AccessibleHub's educational framework is designed to provide a structured, incremental learning experience that progressively builds accessibility knowledge and skills. The content is organized into different *learning modules*, each focusing on a key aspect of mobile accessibility. This is structured incrementally, so to help a developer gather a general idea on what needs to be implemented following a practical roadmap of steps: this allows to focus on different aspects of mobile accessibility, selecting each time the most relevant ones.

The core of the application is divided into different main sections, following:

1. The *Home* screen - The entry point for the AccessibleHub application. It provides an overview of the main sections and guides users on where to start their accessibility learning journey. The Home screen is designed to be intuitive and user-friendly, with clear call-to-action towards the accessible components section, allowing a developer or a user navigate to the desired section from the Home screen, comprehensive of comparison between the main mobile frameworks, learn about best practices in mobile accessibility and access testing tools documentation. There is also present a compliance dashboard provides an overview of an app's accessibility compliance status, based on the WCAG and MCAG guidelines. Developers can use this information to prioritize their accessibility efforts and focus on the areas that need the most attention.

2. The *Accessible Components* screen - Developers can learn how to implement accessible UI components in their mobile applications. This section

is divided into four subscreens, each focusing on a specific category of components:

- *Buttons and Touchables*: This subscreen covers the implementation of accessible buttons and touchable elements. It provides code examples and best practices for ensuring that these interactive elements are perceivable, operable, and understandable by all users, including those with disabilities;
- *Forms*: The Forms subscreen focuses on creating accessible input forms, including text fields, checkboxes, radio buttons, and more. It demonstrates how to properly label form elements, provide instructions and feedback, and ensure that forms can be navigated and completed using various input methods, such as keyboards and screen readers;
- *Media*: In the Media subscreen, developers learn how to make media content, such as images, videos, and audio, accessible to users with visual or auditory impairments. This includes providing alternative text for images, captions for videos, and transcripts for audio content;
- *Dialogs*: The Dialogs subscreen covers the creation of accessible modal dialogs, popups, and alerts. It provides guidance on how to ensure that these elements are properly announced by screen readers, can be easily dismissed, and do not interfere with the user's ability to navigate the application.

Throughout the Accessible Components section, code implementations are provided, which developers can easily copy to their clipboard and integrate into their own projects. This hands-on approach allows developers to quickly apply the accessibility principles they learn and see the results in action.

3. The *Best Practices* screen - Designed to give developers a general understanding of the overarching principles and guidelines for creating accessible

mobile applications. It is divided into five subscreens, each addressing a key aspect of mobile accessibility:

- *Gestures Tutorial*: This subscreen provides an overview of the various gesture interactions used in mobile applications and how to make them accessible to users with motor impairments or those relying on assistive technologies. It covers best practices for implementing alternative input methods and providing clear instructions and feedback.
- *Semantics Structure*: In the Semantics Structure subscreen, developers learn about the importance of using semantic HTML and WAI-ARIA roles to convey the structure and meaning of the application's content. This helps screen readers and other assistive technologies better understand and navigate the application.
- *Navigation*: The Navigation subscreen focuses on creating accessible navigation patterns, such as menus, tabs, and breadcrumbs. It provides guidance on how to ensure that navigation elements are properly labeled, can be operated using various input methods, and provide clear feedback to the user.
- *Screen Reader Support*: This subscreen covers the specific considerations for making mobile applications compatible with screen readers, such as VoiceOver on iOS and TalkBack on Android. It includes best practices for labeling elements, providing alternative text, and ensuring that the application's content and functionality can be fully accessed and understood using a screen reader.
- *Accessibility Guidelines*: The Accessibility Guidelines subscreen provides an overview of the key accessibility standards and guidelines, such as the Web Content Accessibility Guidelines (WCAG), and how they apply to mobile application development. It helps developers understand the different levels of conformance and how to assess their application's accessibility against these guidelines.

4. The *Framework Comparison* screen - It provides a side-by-side comparison

of the accessibility features and implementation differences between popular mobile development frameworks, such as React Native and Flutter. This section helps developers understand how accessibility is handled in each framework and provides guidance on leveraging the specific accessibility APIs and tools available in each one. By highlighting the similarities and differences between frameworks, developers can make informed decisions about which framework to use for their accessibility needs and how to optimize their implementations for each platform.

5. The *Tools* screen - It serves as a central hub for accessing various accessibility-related tools and resources. This includes links to official documentation, such as the React Native Accessibility API reference and the Flutter Accessibility package documentation. It also provides quick access to popular accessibility testing tools, such as Accessibility Scanner for Android and Accessibility Inspector for iOS. By consolidating these resources in one place, the Tools screen makes it easy for developers to find the information and tools they need to ensure their applications are fully accessible.

6. The *Settings* screen - It allows users to customize various aspects of the AccessibleHub application to suit their individual learning needs and preferences. This includes options for adjusting the font size, color contrast, and language settings to ensure that the application itself is accessible to a wide range of users. It also provides information on how to configure the accessibility settings on the user's device, such as enabling screen readers or adjusting the display settings. By offering these customization options and guidance, the Settings page reinforces the importance of accessibility as an everyday tool, meant to accompany practical user needs in an easy and quick way.

TODO COMMENT What I would do here personally:

1. Providing screenshots or wireframes of key app screens and user flows.

2. Explaining the rationale behind the app structure and content organization.

3. Discussing the pedagogical principles and learning theories guiding your approach.

3.4 Accessibility Implementation Guidelines

TODO COMMENT This section should apply into practice, going into formal details of each screen, how to apply guidelines which should be present formally, but in a more readable way, not like Matteo's. I would proceed like this. For each screen/section presented:

1. Foundational accessibility principles

2. WCAG/MCAG guidelines

3. Practical React Native implementation discussed formally

3.4.1 Perceivable

3.4.1.1 Text alternatives

3.4.1.2 Captions and audio descriptions

3.4.1.3 Color contrast

3.4.2 Operable

3.4.2.1 Keyboard navigation

3.4.2.2 Touchscreen gestures

3.4.2.3 Screen reader compatibility

3.4.3 Understandable

3.4.3.1 Predictable interactions

3.4.3.2 Input assistance

3.4.3.3 Error handling and feedback

3.4.4 Robustness

3.4.4.1 Compatibility with assistive technologies

3.5 Fostering an accessibility-focused developer community

3.5.1 Knowledge sharing and collaboration

3.5.2 Contributor recognition and rewards

3.5.3 Continuous learning and improvement

Chapter 4

Accessibility analysis: framework comparison and implementation patterns

This chapter presents a systematic comparative analysis between React Native and Flutter frameworks regarding their accessibility implementations, examining how each framework addresses accessibility requirements in practice. Through empirical evaluation of equivalent components, we investigate three key research questions: whether components are accessible by default, if non-accessible components can be made accessible, and the development overhead required for accessibility implementation. Building upon the educational toolkit introduced previously, this analysis combines quantitative metrics with qualitative assessment of developer experience, providing insights into how each framework supports the creation of accessible mobile applications. The comparison leverages practical implementations to demonstrate how theoretical guidelines translate into functional code, offering a comprehensive evaluation of accessibility approaches via tables and visual representations.

4.1 Research methodology

4.1.1 Research questions and objectives

4.1.2 Testing approach and criteria

4.1.3 Evaluation metrics

4.2 React Native accessibility implementation

4.2.1 Component accessibility analysis

4.2.2 Native platform integration

4.2.3 Implementation patterns and solutions

4.3 Flutter accessibility implementation

4.3.1 Component accessibility patterns

4.3.2 Platform-specific considerations

4.3.3 Common implementation challenges

4.4 Comparative analysis

4.4.1 Default accessibility support

4.4.2 Implementation complexity

4.4.3 Developer experience

4.4.4 Performance implications

4.5 Framework-specific best practices

4.5.1 React Native optimization patterns

4.5.2 Flutter accessibility patterns

4.5.3 Cross-platform considerations

Chapter 5

Conclusions and future research

Lorem ipsum

5.1 Section

Bibliography

Articles and papers

[1] Jaramillo-Alcázar Angel, Luján-Mora - Sergio, and Salvador-Ullauri Luis. "Accessibility Assessment of Mobile Serious Games for People with Cognitive Impairments". In: *2017 International Conference on Information Systems and Computer Science (INCISCOS)* (2017), pp. 323–328. DOI: 10.1109/INCISCOS.2017.12 (cit. on p. 14).

[2] Matteo Budai. "Mobile content accessibility guidelines on the Flutter framework". In: (2024). URL: https://hdl.handle.net/20.500.12608/68870 (cit. on p. 16).

[3] Wei Chen, Ravi Kumar, and Li Zhang. "AccuBot: Automated Accessibility Testing for Mobile Applications". In: *ACM Transactions on Accessible Computing* 16.2 (2023), pp. 1–25. DOI: 10.1145/3584701 (cit. on p. 17).

[4] Silva Claudia, Eler - Marcelo M., and Fraser Gordon. "A survey on the tool support for the automatic evaluation of mobile accessibility". In: *Proceedings of the 8th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-exclusion* (2018), pp. 286–293 (cit. on p. 14).

[6] Oliveira Camila Dias de et al. "Accessibility in Mobile Applications for Elderly Users: A Systematic Mapping". In: *2018 IEEE Frontiers in Education Conference (FIE)* (2018), pp. 1–9 (cit. on p. 13).

[8] Vendome Christopher - Solano Diana and Liñán Santiago - Linares-Vásquez Mario. "Can everyone use my app? An Empirical Study on Accessibility in Android Apps". In: *IEEE* (2019), pp. 41–52. DOI: 10.1109/ICSME.2019.00014 (cit. on p. 12).

[9] Abu Zahra - Husam and Zein - Samer. “A Systematic Comparison Between Flutter and React Native from Automation Testing Perspective”. In: (2022), pp. 6–12. DOI: 10.1109/ISMSIT56059.2022.9932749 (cit. on p. 16).

[10] Alshayban Abdulaziz - Ahmed Iftekhar and Malek Sam. “Accessibility Issues in Android Apps: State of Affairs, Sentiments, and Ways Forward”. In: *International Conference on Software Engineering (ICSE)* (2020), pp. 1323–1334. DOI: 10.1145/3377811.3380392 (cit. on p. 15).

[12] An Nguyen and John Smith. “AccessiFlutter: Enhancing Accessibility in Flutter Applications through Automated Widget Analysis”. In: *Journal of Mobile Engineering* 8 (2022), pp. 45–60. DOI: 10.1016/j.jme.2022.03.004 (cit. on p. 17).

[13] Lorenzo Perinello and Ombretta Gaggi. “Accessibility of Mobile User Interfaces using Flutter and React Native”. In: *IEEE* (2024), pp. 1–8. DOI: 10.1109/CCNC51664.2024.10454681 (cit. on pp. 16, 23).

[14] Zaina Luciana AM Fortes - Renata PM, Casadei Vitor - Nozaki - Leonardo Seiji, and Débora Maria Barroso Paiva. “Preventing accessibility barriers: Guidelines for using user interface design patterns in mobile applications”. In: *Journal of Systems and Software* 186 (2022), p. 111213 (cit. on p. 11).

[15] Pandey Maulishree - Bondre Sharvari - O’Modhrain Sile and Steve Oney. “Accessibility of UI Frameworks and Libraries for Programmers with Visual Impairments”. In: *IEEE* (2022), pp. 1–10. DOI: 10.1109/VL/HCC53370.2022.9833098 (cit. on p. 12).

[16] Priya Singh and Emily Thompson. “A11yReact: A React Native Library for Streamlined Accessibility Compliance”. In: *IEEE Software* 40.3 (2023), pp. 78–85. DOI: 10.1109/MS.2023.3245678 (cit. on p. 18).

Sites

[5] Parliament Assembly of the Council of Europe. *The right to Internet access*. Europian Union. 2014. URL: https://assembly.coe.int/nw/xml/XRef/Xref-XML2HTML-en.asp?fileid=20870 (visited on 11/04/2024).

[7] Statista Research Department. *Number of smartphone users worldwide from 2014 to 2029*. Statista. 2024. URL: https://www.statista.com/statistics/203734/global-smartphone-penetration-per-capita-since-2005/ (visited on 10/20/2024) (cit. on p. 1).

[17] Ronald L.Mace - North Carolina State University. *Universal Design Principles*. UC Berkeley. 1997. URL: https://dac.berkeley.edu/services/campus-building-accessibility/universal-design-principles (visited on 11/04/2024) (cit. on p. 8).

[18] World Health Organization. *WHO - Disability*. World Health Organization. 2023. URL: https://www.who.int/news-room/fact-sheets/detail/disability-and-health (visited on 10/17/2024) (cit. on p. 2).