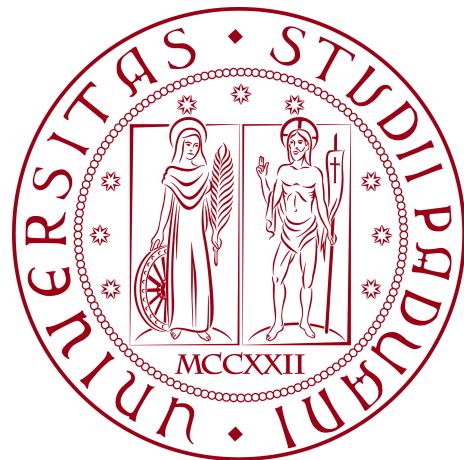


University of Padua

DEPARTMENT OF MATHEMATICS “TULLIO LEVI-CIVITA”

MASTER DEGREE IN COMPUTER SCIENCE



AccessibleHub - Extended screen analysis

Technical Developers' Appendix to Master's Thesis

Supervisor

Prof. Ombretta Gaggi

Candidate

Gabriel Rovesti

ID Number: 2103389

ACADEMIC YEAR 2024-2025

© Gabriel Rovesti, July 2025. All rights reserved. Technical Developers' Appendix to Master's Thesis: “*AccessibleHub - Extended screen analysis*”, University of Padua, Department of Mathematics “Tullio Levi-Civita”.

Abstract

Technical doc abstract

Table of contents

List of listings	xi
Acronyms and abbreviations	xiii
Glossary	xiv
1 Bridging the gap between implementation and accessibility guidelines -	
 Full app analysis	1
1.1 Introduction	1
1.1.1 Purpose and scope	2
1.1.2 Relationship to AccessibleHub	3
1.1.3 How to use this manual	3
1.2 Theoretical foundation	4
1.2.1 Accessibility standards and guidelines	4
1.2.2 Mobile-specific accessibility considerations	5
1.2.3 Implementation principles	6
1.3 Accessibility implementation guidelines	7
1.3.1 Accessible components section	7
1.3.1.1 Analysis methodology	7
1.3.1.2 Common implementation patterns	8
1.3.1.3 Buttons and touchables screen	9
1.3.1.3.1 Component inventory and WCAG/MCAG mapping	9
1.3.1.3.2 Technical implementation analysis	11
1.3.1.3.3 Implementation overhead analysis	13
1.3.1.4 Component implementation comparative analysis	14

TABLE OF CONTENTS

1.3.1.4.1	WCAG criteria implementation	14
1.3.1.4.2	Implementation overhead comparison	15
1.3.1.4.3	Key implementation differences across component types	15
1.3.1.4.4	Screen reader compatibility patterns	16
1.3.1.5	Form screen	16
1.3.1.5.1	Key accessibility considerations	17
1.3.1.5.2	Implementation overhead	18
1.3.1.6	Dialog screen	19
1.3.1.6.1	Focus management implementation	19
1.3.1.6.2	Mobile-specific considerations	21
1.3.1.7	Media screen	21
1.3.1.7.1	Alternative text implementation	21
1.3.1.7.2	Implementation overhead	23
1.3.1.8	Advanced components screen	23
1.3.1.8.1	Complex interaction patterns	23
1.3.1.8.2	Slider accessibility pattern	23
1.3.1.8.3	Implementation overhead	25
1.3.1.9	Key insights from component implementation	26
1.3.2	Best practices main screen	26
1.3.2.1	Component inventory and WCAG/MCAG mapping	28
1.3.2.2	Technical implementation analysis	29
1.3.2.3	Screen reader support analysis	31
1.3.2.4	Implementation overhead analysis	33
1.3.2.5	WCAG conformance by principle	34
1.3.2.6	Category-specific accessibility analysis	35
1.3.2.6.1	WCAG guidelines card	35
1.3.2.6.2	Gesture tutorial card	36
1.3.2.6.3	Screen reader support card	36

TABLE OF CONTENTS

1.3.2.7	Mobile-specific considerations	37
1.3.2.8	Beyond WCAG: pedagogical accessibility guidelines	37
1.3.3	Best practices section	39
1.3.3.1	Analysis methodology	39
1.3.3.2	WCAG guidelines screen	40
1.3.3.2.1	Component inventory and WCAG/MCAG mapping	41
1.3.3.2.2	Technical implementation analysis	43
1.3.3.2.3	Screen reader support analysis	45
1.3.3.2.4	Implementation overhead analysis	45
1.3.3.2.5	Mobile-specific considerations	46
1.3.3.3	Gestures tutorial screen	47
1.3.3.3.1	Component inventory and WCAG/MCAG mapping	48
1.3.3.3.2	Technical implementation analysis	50
1.3.3.3.3	Screen reader support analysis	52
1.3.3.3.4	Implementation overhead analysis	52
1.3.3.3.5	Mobile-specific considerations	53
1.3.3.4	Logical navigation screen	54
1.3.3.4.1	Technical implementation analysis	55
1.3.3.4.2	Screen reader support analysis	57
1.3.3.4.3	Mobile-specific considerations	57
1.3.3.5	Screen reader support screen	58
1.3.3.5.1	Component inventory and WCAG/MCAG mapping	59
1.3.3.5.2	Technical implementation analysis	61
1.3.3.5.3	Mobile-specific considerations	61
1.3.3.6	Semantic structure screen	63
1.3.3.6.1	Component inventory and WCAG/MCAG mapping	64
1.3.3.6.2	Technical implementation analysis	66
1.3.3.6.3	Mobile-specific considerations	66
1.3.3.7	Best practices implementation insights	68

TABLE OF CONTENTS

1.3.3.7.1	Implementation overhead comparison	68
1.3.3.7.2	Key implementation patterns across best practices screens	69
1.3.3.7.3	Future enhancements	70
1.3.4	Accessibility tools screen	70
1.3.4.1	Component inventory and WCAG/MCAG mapping	71
1.3.4.2	Technical implementation analysis	73
1.3.4.3	Screen reader support analysis	75
1.3.4.4	Mobile-specific considerations	76
1.3.4.5	Implementation overhead analysis	77
1.3.4.6	Tool categorization analysis	77
1.3.4.7	Beyond WCAG: development-focused accessibility guidelines	78
1.3.5	Instruction and community screen	79
1.3.5.1	Component inventory and WCAG/MCAG mapping	81
1.3.5.2	Technical implementation analysis	83
1.3.5.3	Project cards implementation	85
1.3.5.4	Screen reader support analysis	87
1.3.5.5	Implementation overhead analysis	88
1.3.5.6	Community resources analysis	88
1.3.5.7	Beyond WCAG: community-centered accessibility guidelines	89
1.3.5.8	Inspirational examples analysis	90
1.3.6	Settings screen	91
1.3.6.1	Component inventory and WCAG/MCAG mapping	93
1.3.6.2	Dynamic accessibility features	94
1.3.6.3	Technical implementation analysis	97
1.3.6.4	Screen reader support analysis	98
1.3.6.5	Implementation overhead analysis	102
1.3.6.6	WCAG conformance by principle	102
1.3.6.7	Mobile-specific considerations	104

TABLE OF CONTENTS

1.3.6.8 Beyond WCAG: self-adapting interface guidelines	104
Bibliography	106

List of figures

1.1	Side-by-side view of the two Button and Touchables screen parts	12
1.2	Side-by-side view of the two Form screen parts	17
1.3	Side-by-side view of the two Dialog screen parts	20
1.4	Side-by-side view of the two Media screen parts	22
1.5	Side-by-side view of the first two Advanced screen parts	24
1.6	Side-by-side view of the second two Advanced screen parts	25
1.7	Side-by-side view of the Best practices screen sections, showing accessibility guideline categories	27
1.8	Side-by-side view of the WCAG Guidelines screen sections	41
1.9	Side-by-side view of the Gestures Tutorial screen sections	48
1.10	Side-by-side view of the Logical navigation screen sections	55
1.11	Side-by-side view of the Screen reader support screen sections	59
1.12	Side-by-side view of the Semantic Structure screen sections	64
1.13	Side-by-side view of the Tools screen sections	71
1.14	Side-by-side view of the Instruction and community screen sections	80
1.15	Side-by-side view of additional Instruction and community screen sections	81
1.16	The Settings screen with various accessibility options	92
1.17	Settings screen with different accessibility modes enabled	95
1.18	Visual notifications when accessibility settings are toggled	97

List of tables

1.1 Buttons screen component-criteria mapping	9
1.2 Buttons screen accessibility implementation overhead	13
1.3 WCAG criteria implementation by component type	14
1.4 Accessibility implementation overhead by component type	15
1.5 Best practices screen component-criteria mapping	28
1.6 Best practices screen screen reader testing results	31
1.7 Best practices screen accessibility implementation overhead	33
1.8 Best practices screen WCAG compliance analysis by principle	34
1.9 Guidelines screen component-criteria mapping	42
1.10 Guidelines screen screen reader testing results	45
1.11 Guidelines screen accessibility implementation overhead	46
1.12 Gestures tutorial screen component-criteria mapping	49
1.13 Gestures tutorial screen screen reader testing results	52
1.14 Gestures tutorial screen accessibility implementation overhead	53
1.15 Logical navigation screen screen reader testing results	57
1.16 Screen reader support screen component-criteria mapping	60
1.17 Semantic structure screen component-criteria mapping	65
1.18 Accessibility implementation overhead by best practices screen	69
1.19 Tools screen component-criteria mapping	72
1.20 Tools screen screen reader testing results	75
1.21 Tools screen accessibility implementation overhead	77
1.22 Tools screen categorization analysis	78

1.23	Instruction screen component-criteria mapping	82
1.24	Instruction screen screen reader testing results	87
1.25	Instruction screen accessibility implementation overhead	88
1.26	Community resources accessibility analysis	89
1.27	Inspiration examples analysis	91
1.28	Settings screen component-criteria mapping	93
1.29	Settings screen screen reader testing results	100
1.30	Settings screen accessibility implementation overhead	102
1.31	Settings screen WCAG compliance analysis by principle	103

List of listings

1.1	Key implementation for accessible button component	11
1.2	Accessible radio button implementation with state management	18
1.3	Dialog implementation with focus management	19
1.4	Accessible image implementation with alternative text	21
1.5	Annotated code sample demonstrating Best practices screen accessibility properties	30
1.6	Annotated code sample demonstrating guidelines screen accessibility properties	44
1.7	Key implementation for accessible gesture detection with screen reader adaptation	50
1.8	Implementation of accessibility actions for gesture simulation	51
1.9	Implementation of Skip to Main Content pattern	56
1.10	Platform toggle implementation with accessibility state	62
1.11	Accessible code with semantic structure implementation	67
1.12	Tool card implementation with accessibility properties	74
1.13	Collapsible preview implementation with accessibility announcements	84
1.14	Project card implementation with accessibility properties	86

LIST OF LISTINGS

1.15 Setting row implementation with accessibility properties	99
1.16 Section headers implementation with proper semantic role	100

Acronyms and abbreviations

API Application Programming Interface. [i](#)

ARIA Accessible Rich Internet Applications. [i](#)

LOC Lines of Code. [i](#)

MCAG Mobile Content Accessibility Guidelines. [i](#)

UI User Interface. [i](#)

UX User Experience. [i](#)

W3C World Wide Web Consortium. [i](#)

WCAG Web Content Accessibility Guidelines. [i](#)

Glossary

Application Programming Interface An Application Programming Interface (API) is a set of protocols, routines, and tools for building software applications. It specifies how software components should interact, allowing different software systems to communicate with each other. APIs define the methods and data structures that developers can use to interact with a system, service, or library without needing to understand the underlying implementation. They serve as a contract between different software components, enabling developers to integrate different systems, access web-based services, and create more complex and interconnected software solutions. [i](#)

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](#)

Flutter Flutter is an open-source UI software development kit created by Google, designed for building natively compiled applications for mobile, web, and desktop platforms from a single codebase. Launched in 2017, Flutter uses the Dart programming language and provides a comprehensive framework for creating high-performance, visually attractive applications with a focus on smooth, responsive user interfaces. Unlike traditional cross-platform frameworks that use web view rendering, Flutter compiles directly to native code, enabling near-native performance. Its key features include a rich set of pre-designed widgets, hot reload for rapid development, extensive customization capabilities, and a robust ecosystem that supports complex application development across multiple platforms. [i](#)

Gray Literature Review A structured method of collecting and analyzing non-traditional

Glossary

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](#)

Lines of Code A metric used to quantify the size or complexity of a software program by counting the number of lines in its source code. LOC is often used as an indicator of development effort, code maintenance, and project scale.. [i](#)

MCAG Mobile Content Accessibility Guidelines (MCAG) are a specialized set of accessibility recommendations specifically tailored to mobile application and mobile web content. While building upon the foundational principles of WCAG, MCAG addresses unique challenges of mobile interfaces, such as touch interactions, small screen sizes, diverse input methods, and mobile-specific assistive technologies. These guidelines provide specific considerations for creating accessible content and interfaces on smartphones, tablets, and other mobile devices, taking into account the distinct interaction patterns and technological constraints of mobile platforms. [i](#)

React Native React Native is an open-source mobile application development framework created by Facebook (now Meta) that allows developers to build mobile applications using JavaScript and React. Introduced in 2015, React Native enables developers to create native mobile apps for both iOS and Android platforms using a single codebase, leveraging the popular React web development library. Unlike hybrid app frameworks, React Native renders components using actual native platform UI elements, providing a more authentic user experience and better performance. The framework bridges the gap between web and mobile development, allowing web developers to create mobile

Glossary

applications using familiar JavaScript and React paradigms, while still achieving near-native performance and user interface responsiveness. [i](#)

Screen Reader A screen reader is an assistive technology software that enables people with visual impairments or reading disabilities to interact with digital devices by converting on-screen text and elements into synthesized speech or Braille output. Screen readers navigate through user interfaces, reading text, describing graphical elements, and providing auditory feedback about the computer or mobile device's content and functionality. They interpret and verbalize user interface elements, buttons, menus, and other interactive components, allowing visually impaired users to understand and interact with digital content. Popular screen readers include VoiceOver for Apple devices, TalkBack for Android, and NVDA and JAWS for desktop computers. [i](#)

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](#)

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

Glossary

and interact with their devices using gestures and keyboard commands. [i](#)

W3C The World Wide Web Consortium (W3C) is an international community that develops open standards to ensure the long-term growth and evolution of the web. Founded by Tim Berners-Lee in 1994, the W3C works to create universal web standards that promote interoperability and accessibility across different platforms, browsers, and devices. This non-profit organization brings together technology experts, researchers, and industry leaders to develop guidelines and protocols that form the fundamental architecture of the World Wide Web. Key contributions include HTML, CSS, accessibility guidelines (WCAG), and web standards that ensure a consistent, inclusive, and innovative web experience for users worldwide. [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

Bridging the gap between implementation and accessibility guidelines - Full app analysis

This manual provides a complete practical implementation guidance for mobile developers seeking to create accessible applications. Building upon the research presented in Chapter 3 of the Master Thesis present [here](#), it offers a comprehensive, screen-by-screen analysis of accessibility considerations in mobile interfaces. Through detailed implementation examples, WCAG compliance mappings, and platform-specific considerations, this manual transforms theoretical accessibility guidelines into practical code.

1.1 Introduction

The gap between understanding accessibility guidelines and implementing them in practice remains one of the most significant challenges for mobile developers. While theoretical knowledge about Web Content Accessibility Guidelines (WCAG) and Mobile Content Accessibility Guidelines (MCAG) is widely available, practical examples and implementation patterns are often lacking. This manual addresses this gap by providing a comprehensive, code-focused guide to implementing accessibility features across different screen types and component categories.

1.1.1 Purpose and scope

This developer manual serves several key purposes:

- Providing concrete implementation examples for accessibility features in mobile applications;
- Demonstrating how abstract WCAG and MCAG guidelines translate into practical code;
- Quantifying the implementation overhead of accessibility features to assist in project planning;
- Highlighting mobile-specific accessibility considerations that extend beyond standard web guidelines;
- Establishing reusable patterns that developers can adapt to their own projects.

Rather than offering general recommendations, this manual takes a screen-by-screen approach, examining specific implementation challenges and solutions for different interface types. Each screen analysis follows a consistent methodological framework that includes:

1. **Component inventory and WCAG/MCAG mapping:** Formal identification of UI elements with their semantic roles and relationships to accessibility guidelines;
2. **Technical implementation analysis:** Detailed code examples with annotations highlighting key accessibility properties;
3. **Screen reader support analysis:** Results from empirical testing with VoiceOver (iOS) and TalkBack (Android);
4. **Implementation overhead analysis:** Quantification of the additional code required to implement accessibility features;
5. **Mobile-specific considerations:** Adaptations necessary for touch interfaces and mobile contexts that extend beyond standard WCAG requirements.

1.1.2 Relationship to AccessibleHub

AccessibleHub is a React Native application designed to serve as an interactive manual for implementing accessibility features in mobile development. This developer manual documents the technical implementation details of *AccessibleHub* itself, analyzing how in full detail how its various screens address accessibility requirements through specific code patterns.

The application structure follows the educational framework described in Section §4.3 of the Master Thesis present [here](#), with screens organized into logical sections:

- **Accessible components section:** Providing an overview of different components of different degree of complexity organized incrementally, with the possibility of seeing code examples and use those as reference in projects;
- **Best practices main screen and section:** Offering guidance on overarching accessibility principles, organized once again by different degrees of complexity, from hierarchy navigation to logical focus;
- **Tools screen:** Cataloging resources for testing and implementing accessibility, with practical and theoretical experience;
- **Instruction and community screen:** Connecting developers to broader learning resources and different accessibility projects;
- **Settings screen:** Providing accessibility customization options, to be used across screens and as examples inside of a developer project.

By documenting the accessibility implementation of *AccessibleHub* itself, this manual creates a recursive learning experience, where the medium demonstrates the message—showing accessibility implementation through an accessible application.

1.1.3 How to use this manual

This manual can be approached in several ways depending on the developer's specific needs:

- **Component-focused learning:** Developers working on specific interface elements can go directly to the relevant component sections to find implementation patterns;
- **Screen-type guidance:** Those designing particular screen types (e.g., settings screens, home screens) can reference the corresponding section for holistic accessibility approaches;
- **Code pattern library:** The annotated code samples throughout the manual can serve as a pattern library for implementing specific accessibility features;
- **Compliance mapping reference:** The WCAG/MCAG mapping tables provide a reference for understanding which implementation techniques address specific accessibility requirements;
- **Implementation planning:** The overhead analyses help developers and project managers understand the additional resources required to implement comprehensive accessibility.

Throughout the manual, we maintain a focus on practical implementation while grounding our recommendations in formal accessibility standards and empirical testing results. The examples are presented in React Native code but include notes on Flutter implementation differences where relevant, making the guidance valuable across multiple frameworks.

1.2 Theoretical foundation

Before diving into specific screen implementations, it's important to establish briefly the theoretical foundation that guides our accessibility approach. This foundation consists of three interconnected dimensions: formal guidelines, mobile-specific adaptations, and implementation principles.

1.2.1 Accessibility standards and guidelines

Our implementation patterns are guided by several formal accessibility standards:

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

- **WCAG 2.2:** Web Content Accessibility Guidelines provide the core success criteria for digital accessibility, organized under four principles: Perceivable, Operable, Understandable, and Robust;
- **MCAG:** Mobile Content Accessibility Guidelines extend WCAG principles with mobile-specific considerations for touch interfaces, limited screen size, and varied usage contexts;
- **Platform-specific guidelines:** Both Apple (iOS) and Google (Android) maintain specific accessibility guidelines for their respective platforms, which we incorporate into our implementation patterns.

Throughout this manual, we map implementation techniques to specific WCAG and MCAG criteria, demonstrating how abstract guidelines translate into concrete code patterns. We particularly focus on meeting Level AA requirements while implementing Level AAA enhancements where practical.

1.2.2 Mobile-specific accessibility considerations

Mobile interfaces present unique accessibility challenges that extend beyond traditional web considerations:

- **Touch interaction:** Mobile interfaces rely primarily on touch rather than keyboard or mouse, requiring different approaches to ensure operability;
- **Screen reader gestures:** Mobile screen readers use specific touch gestures that differ from standard touch interactions, requiring adaptations for proper accessibility;
- **Limited viewport:** Smaller screens necessitate careful organization of content and navigation to prevent disorientation;
- **Context switching:** Mobile users frequently switch between applications and contexts, requiring clear signposting and state preservation;

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

- **Diverse environments:** Mobile devices are used in varied lighting conditions, noise levels, and physical environments, requiring adaptability.

These considerations inform our implementation patterns and extend our accessibility approach beyond simple WCAG compliance to address the practical realities of mobile usage.

1.2.3 Implementation principles

Throughout this manual, we apply several core implementation principles that guide our approach to accessibility:

- **Semantic integrity:** UI elements should communicate their purpose and role explicitly to assistive technologies;
- **Focus management:** Applications should maintain logical focus order and explicitly manage focus during dynamic interactions;
- **State communication:** Interactive elements should communicate their states (selected, disabled, etc.) clearly to all users;
- **Multi-sensory feedback:** Actions and changes should be communicated through multiple sensory channels (visual, auditory, haptic);
- **Adaptability:** Interfaces should adapt to user preferences and needs, providing options for personalization;
- **Touch optimization:** Interactive elements should be sized and positioned for optimal touch interaction;
- **Screen reader efficiency:** Implementation should minimize unnecessary interactions for screen reader users.

These principles inform the specific code patterns and implementations documented throughout this manual, creating a coherent approach to accessibility that extends across different screen types and component categories.

1.3 Accessibility implementation guidelines

Having established the overall architecture of *AccessibleHub* and the guiding principles from both *WCAG* and *MCAG*, we now present a screen-by-screen analysis. Each subsection highlights the key *success criteria* addressed, references relevant *mobile-specific considerations*, and demonstrates practical solutions in React Native. Where applicable, we contrast these with Flutter's approach, building upon the insights from Gaggi and Perinello's approach [4] analyzing Budai's Flutter code - following guidelines and then giving advice into introducing new ones. These screens follow the structure presented in Section ??, analyzed both as main screens and sections.

1.3.1 Accessible components section

This section provides a formal analysis of the various screens within the Accessible Components section of *AccessibleHub*. As the core educational element of the application, these screens demonstrate practical implementation patterns for accessibility across commonly used mobile interface elements.

1.3.1.1 Analysis methodology

To systematically evaluate the accessibility implementation across multiple component screens, we employ a consistent analytical framework that examines:

1. **Component inventory:** Identification and classification of UI elements with mapping to their semantic roles and accessibility properties;
2. **WCAG/MCAG criteria mapping:** Formal mapping between components and relevant accessibility guidelines;
3. **Implementation analysis:** Evaluation of code patterns and accessibility properties;
4. **Screen reader compatibility:** Empirical testing with VoiceOver (iOS) and TalkBack (Android);

5. **Implementation overhead:** Quantification of code additions required for accessibility features.

Each component screen follows a consistent educational structure that scaffolds learning through:

- Interactive demonstrations of accessible implementations;
- Copyable code examples with highlighted accessibility properties;
- Explanations of key accessibility features and considerations;
- Platform-specific adaptation notes.

Rather than presenting each screen with identical analytical depth, we'll examine the Buttons screen in detail as a representative example, then provide comparative analysis across all component types to identify patterns, commonalities, and unique considerations.

1.3.1.2 Common implementation patterns

Across all component screens in this section, several foundational accessibility implementation patterns are consistently applied:

1. **Semantic role assignment:** All components use appropriate `accessibilityRole` properties to identify their purpose to assistive technologies;
2. **Comprehensive labeling:** Components combine `accessibilityLabel` and `accessibilityHint` to provide both identification and action context;
3. **Explicit state communication:** Interactive components use `accessibilityState` to communicate selection, completion, or disabled states;
4. **Decorative element hiding:** Non-essential visual elements use `accessibilityElementsHidden` to streamline screen reader navigation;
5. **Status announcements:** State changes are explicitly announced via `AccessibilityInfo.announceForAccessibility`;

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

6. **Enhanced touch targets:** All interactive elements maintain minimum dimensions of $44 \times 44\text{dp}$, exceeding WCAG 2.5.8 requirements.

Each component screen also implements a consistent visual structure that reinforces the educational purpose:

- A demonstration area with interactive examples;
- A code example section with syntax-highlighted implementation;
- A features section highlighting key accessibility properties;
- A platform considerations section addressing iOS and Android differences.

1.3.1.3 Buttons and touchables screen

The Buttons and Touchables screen demonstrates fundamental accessibility implementations for the most common interactive elements in mobile applications. It provides implementation examples for accessible touch targets with proper sizing, meaningful labels, and appropriate feedback mechanisms. Figure 1.1 shows the main interface of this screen.

1.3.1.3.1 Component inventory and WCAG/MCAG mapping Table 1.1 provides a formal mapping between the UI components, their semantic roles, the specific WCAG 2.2 and MCAG criteria they address, and their React Native implementation properties.

Table 1.1: Buttons screen component-criteria mapping

Component	Semantic Role	WCAG 2.2 Criteria	MCAG Considerations	Implementation Properties
Hero Title	heading	1.4.3 Contrast (AA) 2.4.6 Headings (AA)	Text readability on variable screen sizes	accessibilityRole = "header"

Continued on next page

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND
ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

Table 1.1 – continued from previous page

Component	Semantic Role	WCAG 2.2 Criteria	MCAG Considerations	Implementation Properties
Demo Button	button	1.4.3 Contrast (AA) 2.5.8 Target Size (AA) 4.1.2 Name, Role, Value (A)	Minimum touch target size Haptic feedback	<code>accessibilityRole = "button", accessibilityLabel = "Submit form", accessibilityHint = "Activates form submission"</code>
Code Snippet	text	1.3.1 Info and Relationships (A)	Content structure preservation	<code>accessibilityRole="text", accessibilityLabel= "Button implementation code"</code>
Copy Button	button	1.4.3 Contrast (AA) 4.1.3 Status Messages (AA)	Touch target size Action feedback	<code>accessibilityRole = "button", accessibilityLabel = "{copied ? "Code copied" : "Copy code example"}"</code>
Success Modal	alertdialog	4.1.3 Status Messages (AA)	Screen reader announcements	<code>accessibilityViewIsModal, accessibilityLiveRegion = "polite"</code>
Feature Cards	none	1.3.1 Info and Relationships (A)	Logical grouping	<code>accessibilityRole="text"</code>

Continued on next page

Table 1.1 – continued from previous page

Component	Semantic Role	WCAG 2.2 Criteria	MCAG Considerations	Implementation Properties
Feature Icons	none	1.1.1 Non-text Content (A)	Reduction of unnecessary focus stops	accessibilityElements Hidden=true, importantForAccessibility ="no-hide-descendants"

1.3.1.3.2 Technical implementation analysis The Buttons and Touchables screen exemplifies proper accessibility implementation for interactive elements. The core demo button showcases three fundamental accessibility considerations: proper role assignment, descriptive labeling, and sufficient touch target size. Listing 1.1 highlights the key implementation aspects.

```

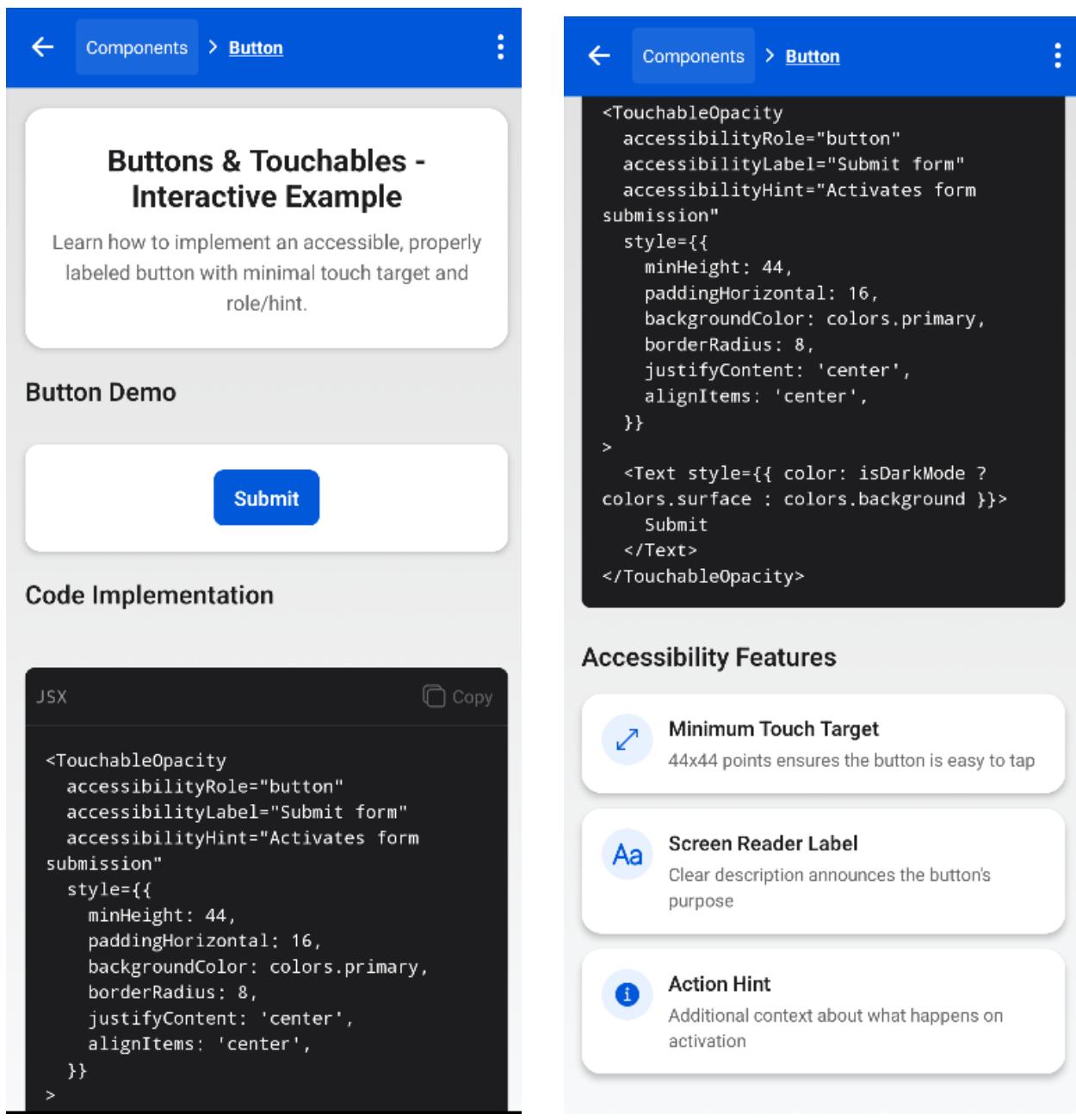
1 <TouchableOpacity
2   style={[styles.demoButton, { backgroundColor: colors.primary }]}
3   accessibilityRole="button"
4   accessibilityLabel="Submit form"
5   accessibilityHint="Activates form submission"
6   onPress={() => {
7     setShowSuccess(true);
8     AccessibilityInfo.announceForAccessibility('Button pressed
9       successfully');
10    setTimeout(() => setShowSuccess(false), 2000);
11  }}
12  >
13    <Text style={[styles.buttonText, {
14      color: '#FFFFFF'
15    }]}>
16      Submit
17    </Text>
</TouchableOpacity>
```

Listing 1.1: Key implementation for accessible button component

Several key accessibility considerations are implemented in this example:

- 1. Proper semantic role:** The implementation explicitly assigns the button role using `accessibilityRole="button"`, ensuring screen readers correctly identify the compo-

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS



(a) Button screen - Part 1

(b) Button screen - Part 2

Figure 1.1: Side-by-side view of the two Button and Touchables screen parts

nent's purpose;

2. **Descriptive accessibility labels:** The button includes both an `accessibilityLabel` that identifies its function and explains the result of interaction, providing comprehensive context for screen reader users;

3. **Adequate touch target size:** The button implements the enhanced touch target size recommendation from WCAG 2.5.8 (Target Size) by using a minimum height of 44px, significantly exceeding the minimal Level AA requirement of 24x24 pixels;
4. **Status feedback:** When pressed, the button announces its state change via `AccessibilityInfo.announceForAccessibility`, proactively notifying screen reader users of the action result;
5. **Visual feedback:** The success modal provides visual confirmation of the button press, with appropriate `accessibilityLiveRegion="polite"` to ensure screen readers announce the status change.

1.3.1.3.3 Implementation overhead analysis Table 1.2 quantifies the additional code required to implement accessibility features in the Buttons and touchables screen.

Table 1.2: Buttons screen accessibility implementation overhead

Accessibility Feature	Lines of Code	Percentage of Total	Complexity Impact
Semantic Roles	10 LOC	2.2%	Low
Descriptive Labels	14 LOC	3.1%	Low
Element Hiding	12 LOC	2.7%	Low
Status Announcements	8 LOC	1.8%	Low
Touch Target Sizing	6 LOC	1.3%	Low
Modal Accessibility	10 LOC	2.2%	Medium
Total	60 LOC	13.3%	Low

This analysis reveals that implementing comprehensive button accessibility features adds approximately 13.3% to the code base, representing a relatively low overhead for significantly improved user experience. Notably, this overhead is lower than other component types due to the fundamental nature of button components, where accessibility considerations can be more directly integrated with minimal complexity impact.

1.3.1.4 Component implementation comparative analysis

Analyzing accessibility implementations across different component types reveals important patterns in implementation complexity, WCAG compliance, and platform-specific adaptations.

1.3.1.4.1 WCAG criteria implementation Table 1.3 compares WCAG 2.2 success criteria implementation across component types.

Table 1.3: WCAG criteria implementation by component type

WCAG Success Criteria	Buttons	Forms	Dialogs	Media	Advanced
1.1.1 Non-text Content (A)	✓	✓	✓	✓	✓
1.3.1 Info and Relationships (A)	✓	✓	✓	✓	✓
2.4.3 Focus Order (A)	✗	✓	✓	✗	✓
3.3.1 Error Identification (A)	✗	✓	✗	✗	✗
4.1.2 Name, Role, Value (A)	✓	✓	✓	✓	✓
4.1.3 Status Messages (AA)	✓	✓	✓	✓	✓
Total Implementation	9/12	12/12	10/12	9/12	10/12

This analysis reveals several key patterns:

- Universal criteria:** Three criteria (1.1.1 Non-text Content, 1.3.1 Info and Relationships, and 4.1.2 Name, Role, Value) are implemented across all component types, forming the core of mobile accessibility requirements;
- Component-specific criteria:** Some criteria are relevant only to specific component types, such as 3.3.1 Error Identification for forms;
- Interaction complexity correlation:** More complex interaction patterns (Forms, Dialogs, Advanced) implement more criteria, particularly those related to focus management and state communication.

1.3.1.4.2 Implementation overhead comparison Table 1.4 compares the implementation overhead across component types.

Table 1.4: Accessibility implementation overhead by component type

Component Type	Lines of Code	Percentage Overhead	Complexity Impact	Primary Contributors
Buttons	60	13.3%	Low	Labels, Roles
Forms	153	21.5%	Medium	State, Labels, Errors
Dialogs	94	16.2%	Medium	Focus Management
Media	68	12.7%	Low	Alt Text, Controls
Advanced	183	22.7%	High	Slider Controls, Announcements

This comparison reveals a direct correlation between interaction complexity and accessibility implementation overhead. Simple components like buttons and media have the lowest overhead (12-13%), while complex components with state management and alternative interaction patterns have significantly higher overhead (21-23%).

1.3.1.4.3 Key implementation differences across component types Each component type presents unique accessibility challenges requiring specialized implementation approaches:

1. **Forms:** Require explicit error identification and validation feedback using `accessibilityRole="alert"` to ensure compliance with WCAG 3.3.1 (Error Identification). They also implement complex state communication for selection controls like radio buttons and checkboxes via `accessibilityState={{checked: selected}};`
2. **Dialogs:** Focus management represents the critical accessibility challenge, requiring explicit tracking of focus position and restoration when the dialog closes to comply with WCAG 2.4.3 (Focus Order);

3. **Media:** Alternative text implementation forms the core accessibility requirement, with proper `accessibilityLabel` values describing non-text content as per WCAG 1.1.1;
4. **Advanced components:** Require the most sophisticated implementations, particularly for inherently visual controls like sliders, which implement alternative interaction mechanisms (buttons, presets) for screen reader users.

1.3.1.4.4 Screen reader compatibility patterns Empirical testing with VoiceOver (iOS) and TalkBack (Android) reveals consistent patterns across component types:

1. Both screen readers correctly identify components with properly assigned `accessibilityRole` values;
2. State changes communicated via `accessibilityState` are properly announced;
3. Status messages delivered via `AccessibilityInfo.announceForAccessibility` are consistently reported to users;
4. Focus management implementation in dialogs works reliably on both platforms, with some minor timing differences;
5. Elements hidden with `accessibilityElementsHidden` are consistently excluded from the accessibility tree on both platforms.

These findings confirm that the accessibility implementation patterns used throughout the component screens provide consistent and reliable behavior across both major mobile platforms when proper accessibility properties are applied.

1.3.1.5 Form screen

The Form screen demonstrates complex accessibility patterns for capturing user input. Unlike the simpler Buttons screen, form elements present additional challenges related to input association, validation feedback, and state communication. Figure 1.2 shows the main interface of this screen.

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

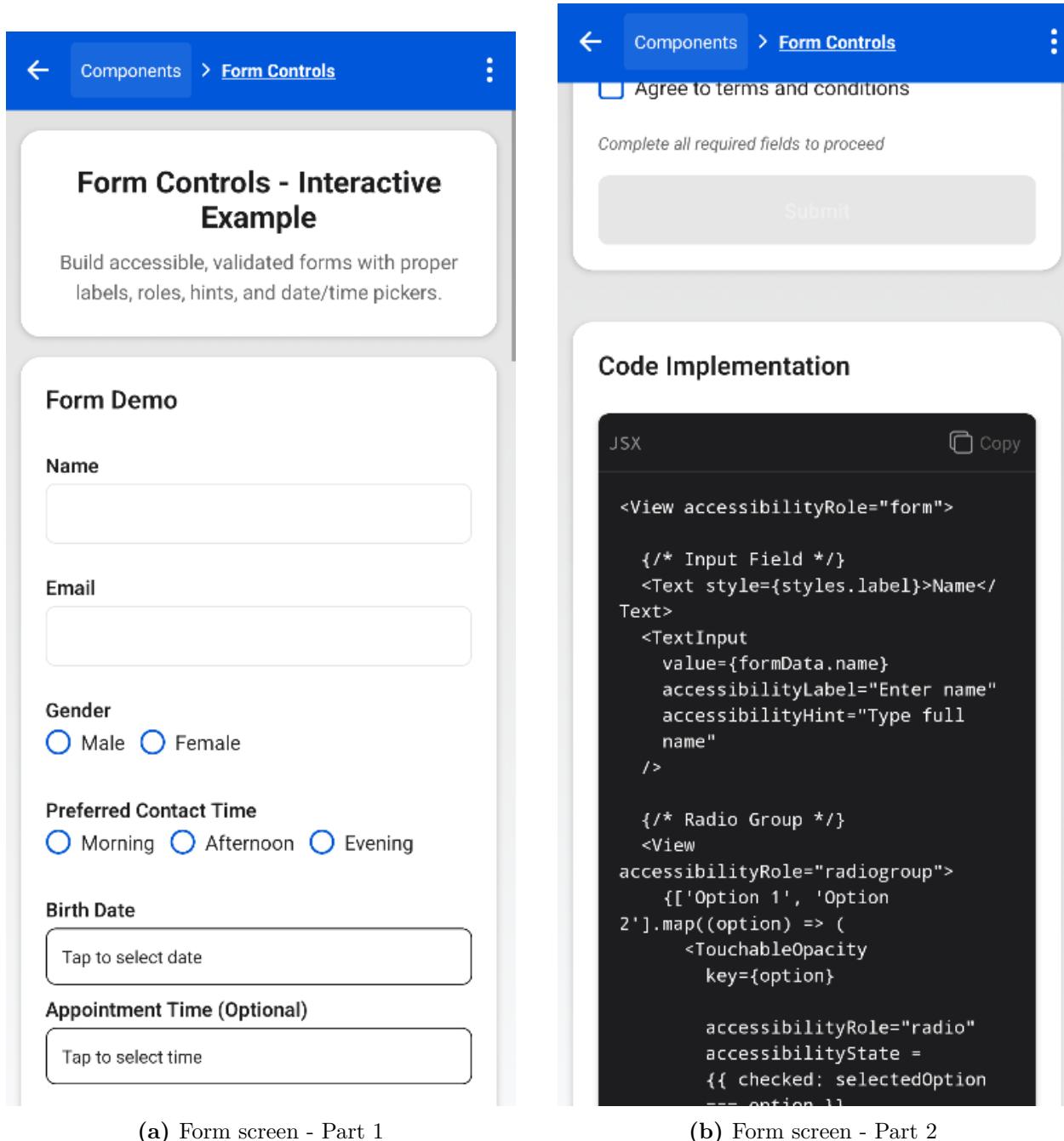


Figure 1.2: Side-by-side view of the two Form screen parts

1.3.1.5.1 Key accessibility considerations The Form screen addresses several critical accessibility patterns beyond basic labeling:

- 1. Input association:** Clear association between labels and input fields using semantic grouping;

2. **Error identification:** Proper error messaging with `accessibilityRole="alert"` for validation feedback;
3. **State communication:** Selection state for radio buttons and checkboxes with `accessibilityState={{checked: selected}}`;
4. **Native picker integration:** Leveraging platform-native date pickers for optimal accessibility.

Listing 1.2 demonstrates the implementation of accessible form controls with proper state management.

```
1 <View accessibilityRole="radiogroup">
2   {[ 'Male' , 'Female' ].map((option) => (
3     <TouchableOpacity
4       key={option}
5       style={styles.radioItem}
6       onPress={() => setFormData((prev) => ({ ...prev, gender: option
7         }))}
8       accessibilityRole="radio"
9       accessibilityState={{ checked: formData.gender === option }}
10      accessibilityLabel={'Select ${option}'}
11    >
12      <View
13        style={[
14          styles.radioButton ,
15          { borderColor: colors.primary },
16          formData.gender === option && { backgroundColor:
17            colors.primary },
18        ]}
19      />
20      <Text style={[styles.radioLabel , { color: colors.text }]}>
21        {option}
22      </Text>
23    </TouchableOpacity>
24  ))
25 </View>
```

Listing 1.2: Accessible radio button implementation with state management

1.3.1.5.2 Implementation overhead Forms have the highest accessibility implementation overhead (21.5%) among component types, reflecting the complexity of making multi-

part input systems fully accessible. The primary contributors to this overhead are state communication mechanisms and validation feedback systems.

1.3.1.6 Dialog screen

The Dialog screen addresses one of the most challenging accessibility patterns in mobile applications: modal content that must trap and manage focus while providing clear context and exit mechanisms. Figure 1.3 shows the main interface of this screen.

1.3.1.6.1 Focus management implementation The key accessibility challenge for dialogs is proper focus management, as illustrated in Listing 1.3.

```
1 // References for focus management
2 const dialogRef = useRef(null);
3 const openButtonRef = useRef(null);
4
5 // Focus management useEffect hook
6 useEffect(() => {
7   if (showDialog) {
8     AccessibilityInfo.announceForAccessibility(
9       'Example dialog opened. This dialog contains information about
10      accessibility features.'
11   );
12   // Brief timeout to ensure dialog is fully rendered
13   setTimeout(() => {
14     dialogRef.current?.focus();
15   }, 100);
16 } else {
17   // Return focus to open button when dialog closes
18   openButtonRef.current?.focus();
19 }
}, [showDialog]);
```

Listing 1.3: Dialog implementation with focus management

The dialog implementation addresses several critical accessibility requirements:

1. **Modal context:** Setting `accessibilityViewIsModal=true` to establish a focused interaction context;
2. **Focus trapping:** Managing focus to prevent interaction with background content;

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

The figure consists of two side-by-side screenshots of a mobile application interface, labeled (a) and (b).

(a) Dialog screen - Part 1: This part shows a modal dialog titled "Modal Dialogs - Interactive Example". It contains the text: "Build dialogs with focus trapping, screen reader support, and proper roles." Below this is a section titled "Dialog Demo" with a blue button labeled "Open Dialog". The background shows a navigation bar with "Components" and "Dialog".

```
JSX
const AccessibleDialog = ({ visible, onClose, title, children }) => {
  return (
    <Modal
      visible={visible}
      transparent
      animationType="fade"
      onRequestClose={onClose}
      accessibilityViewIsModal={true}
      accessibilityLiveRegion="polite"
    >
      {children}
    </Modal>
  );
};
```

(b) Dialog screen - Part 2: This part shows the same navigation bar. It displays the JSX code for the "AccessibleDialog" component, which is identical to the code in part (a). Below the code, there is a section titled "Accessibility Features" with three items: "Focus Management", "Keyboard Navigation", and "Screen Reader Support", each accompanied by an icon.

- Focus Management:** Proper focus trapping and restoration when the dialog opens and closes
- Keyboard Navigation:** Full keyboard support including escape key to close the dialog
- Screen Reader Support:** Proper ARIA roles and live region announcements

Figure 1.3: Side-by-side view of the two Dialog screen parts

3. **Return focus:** Explicitly returning focus to the triggering element when the dialog closes;
4. **Status announcements:** Using `AccessibilityInfo.announceForAccessibility` to provide context about dialog opening and closing.

1.3.1.6.2 Mobile-specific considerations Dialog implementation on mobile platforms presents unique accessibility challenges:

- **Limited viewport context:** Unlike desktop interfaces, mobile screens cannot show both dialog and background content simultaneously, requiring stronger contextual cues;
- **Touch dismissal patterns:** Implementation of touch-friendly dismissal actions with adequate target sizes;
- **Platform convention alignment:** Following platform-specific dialog patterns for consistent user experience.

1.3.1.7 Media screen

The Media screen demonstrates accessibility techniques for non-text content—one of the most fundamental aspects of digital accessibility, employing some placeholder images free of license as examples. Figure 1.4 shows the main interface of this screen.

1.3.1.7.1 Alternative text implementation Listing 1.4 shows the core pattern for accessible image implementation with proper alternative text.

```
1 <Image
2   source={images[currentImage - 1].uri}
3   style={themedStyles.demoImage}
4   accessibilityLabel={images[currentImage - 1].alt}
5   accessible={true}
6   accessibilityRole="image"
7 />
```

Listing 1.4: Accessible image implementation with alternative text

The Media screen demonstrates additional accessibility features beyond basic alternative text:

1. **Navigation controls:** Accessible previous/next buttons with clear labeling and state indication;

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

← Components > Media Content ⋮

Media Content - Interactive Example

View images with detailed alternative text and roles. Use the controls below to navigate.

Media Demo



< Hide Alt Text >

Alternative Text:
A placeholder image (first example)
Role: Interface example

Code Implementation

```
JSX
```

```
<Image  
  source={require('./path/to/  
image.png')}  
  accessibilityLabel="Detailed  
description of the image content"  
  accessible={true}  
  accessibilityRole="image"  
  style={{  
    width: 300,  
    height: 200,  
    borderRadius: 8,  
  }}  
/>
```

← Components > Media Content ⋮

Code Implementation

Accessibility Features

Aa **Alternative Text**
Descriptive text that conveys the content and function of the image

Speaker **Role Announcement**
Screen readers announce the element as an image

Hand **Touch Target**
Interactive images should have adequate touch targets

(a) Media screen - Part 1

(b) Media screen - Part 2

Figure 1.4: Side-by-side view of the two Media screen parts

2. **Interactive alt text:** Toggle mechanism to show/hide alternative text as an educational feature;
3. **Position context:** Announcements that communicate current position within a gallery (e.g., "Image 2 of 5").

1.3.1.7.2 Implementation overhead Media components have the lowest accessibility implementation overhead (12.7%) among component types, as the primary requirement—alternative text—is implemented through straightforward property assignment. The majority of the overhead comes from implementing accessible navigation controls rather than the core media content itself.

1.3.1.8 Advanced components screen

The Advanced components screen demonstrates accessibility implementations for more complex UI patterns including tabs, progress indicators, alerts, and sliders. Figure 1.5 and 1.6 shows the two parts of the main interface of this screen.

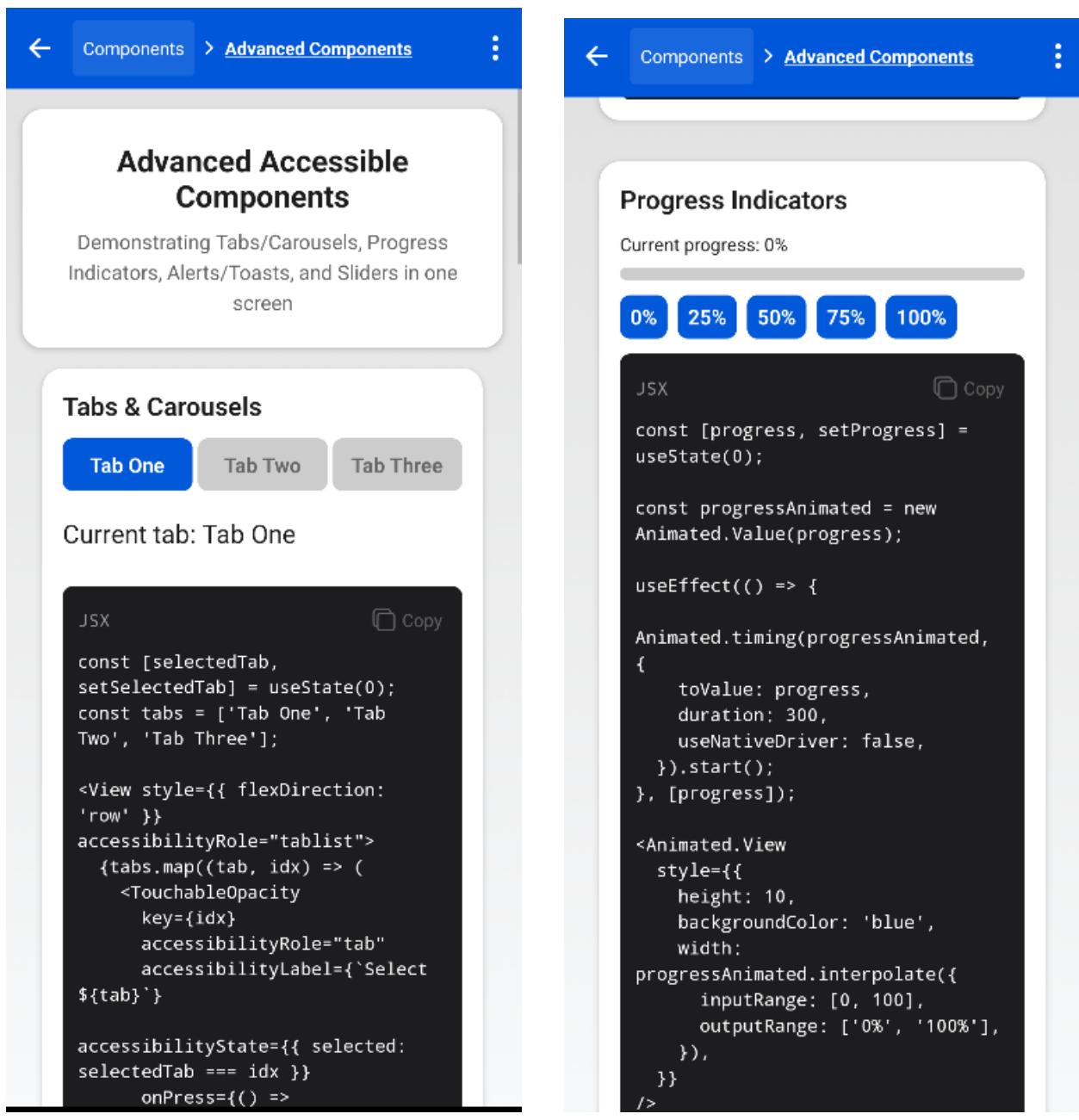
1.3.1.8.1 Complex interaction patterns Advanced components present unique accessibility challenges requiring specialized implementations:

1. **Tab navigation:** Proper role assignment with `accessibilityRole="tablist"` for containers and `accessibilityRole="tab"` for individual tabs, with selection state communicated through `accessibilityState`;
2. **Progress indicators:** Value communication through `accessibilityValue` properties with min/max/current parameters;
3. **Alerts and toasts:** Implementation of `accessibilityLiveRegion="assertive"` for time-sensitive notifications;
4. **Slider alternatives:** Provision of button-based alternatives for precise slider control by screen reader users.

1.3.1.8.2 Slider accessibility pattern The slider implementation (shown in Figure 1.6b) demonstrates a particularly important accessibility pattern: providing alternative interaction mechanisms for inherently visual controls. This pattern includes:

- Button controls for incremental adjustments;

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS



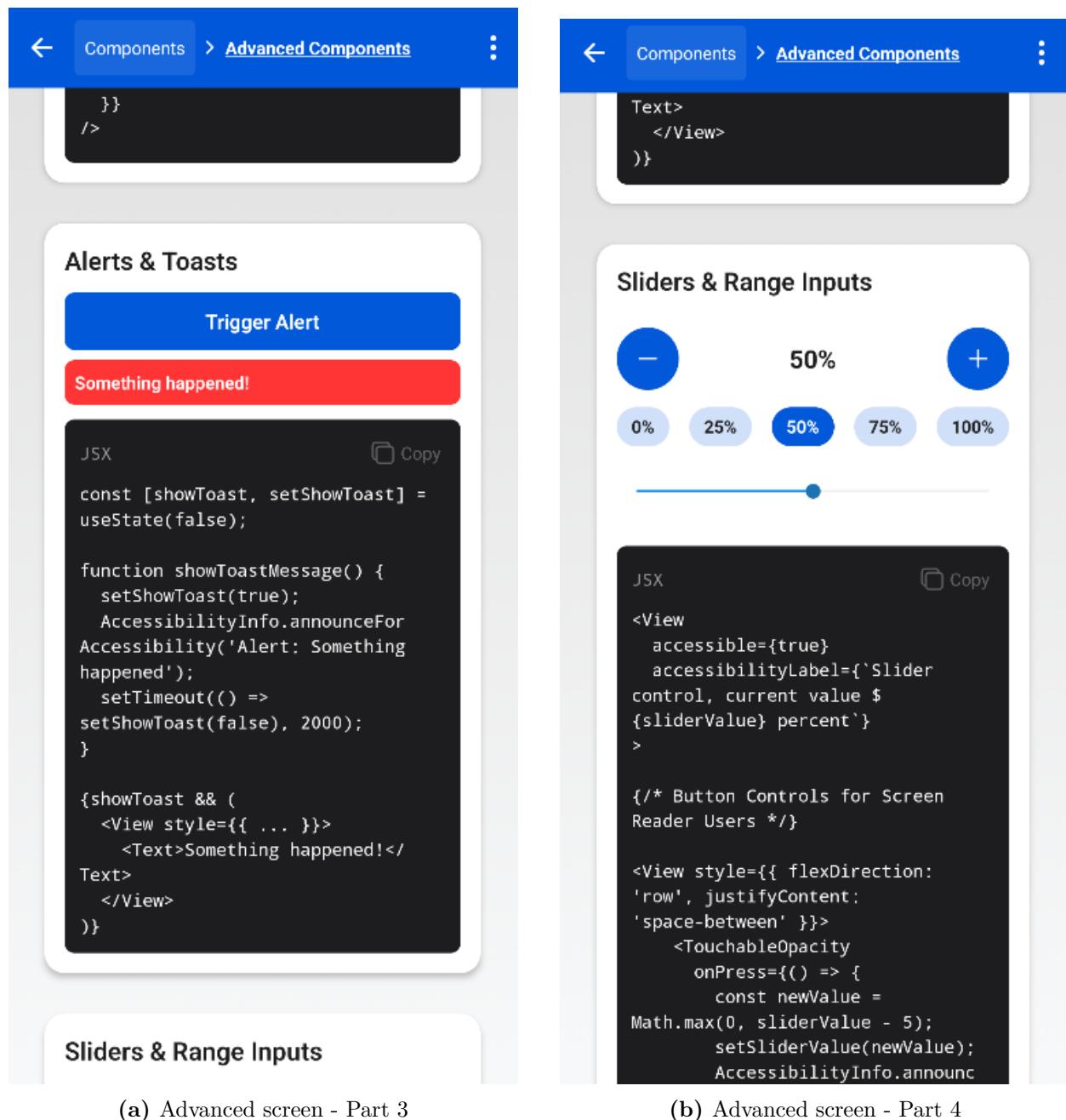
(a) Advanced screen - Part 1

(b) Advanced screen - Part 2

Figure 1.5: Side-by-side view of the first two Advanced screen parts

- Preset value buttons for common settings;
- Value announcements with appropriate throttling;
- Visual feedback synchronized with announced values.

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS



(a) Advanced screen - Part 3

(b) Advanced screen - Part 4

Figure 1.6: Side-by-side view of the second two Advanced screen parts

1.3.1.8.3 Implementation overhead Advanced components have the highest implementation overhead (22.7%) among component types, with slider controls being particularly demanding (8.1% overhead). This reflects the additional complexity required to make inherently visual controls accessible through alternative interaction mechanisms.

1.3.1.9 Key insights from component implementation

The analysis of multiple component implementations reveals several critical insights for developers implementing accessibility in mobile applications:

1. **Implementation complexity correlates with interaction complexity:** More complex interaction patterns require more sophisticated accessibility implementations, with forms and advanced components requiring the highest implementation overhead;
2. **Focus management is critical for non-linear interactions:** Components that create new interaction contexts (dialogs) or complex navigation patterns (tabs) require explicit focus management to maintain user orientation;
3. **Alternative interaction mechanisms are essential for inherently visual controls:** Components like sliders require additional interaction mechanisms to ensure operability by screen reader users;
4. **Explicit state communication improves usability:** All interactive components benefit from explicit state communication via `accessibilityState` and announcements, but this is particularly critical for selection-based controls;
5. **Platform-specific adaptations may be necessary:** While React Native provides a unified accessibility API, some components (particularly date pickers and complex inputs) benefit from platform-specific adaptations to leverage native accessibility features.

These insights provide developers with a framework for prioritizing accessibility implementation efforts, focusing on the components and patterns that present the greatest challenges and require the most sophisticated approaches to ensure equal access for all users.

1.3.2 Best practices main screen

The Best practices screen serves as a comprehensive educational resource within the *AccessibleHub* application. It provides developers with access to essential guidelines, patterns, and interactive resources for implementing accessibility in mobile applications. The

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

screen organizes accessibility knowledge into five key categories: *WCAG Guidelines*, *Semantic Structure*, *Gesture Tutorial*, *Screen Reader Support*, and *Logical Focus Order*. An example of the interface is shown in Figure 1.7.

The figure consists of two side-by-side screenshots of a mobile application's "Best Practices" section. Both screenshots have a blue header bar with a menu icon, "Home" button, and a "Best Practices" button. A three-dot menu icon is also present in the top right corner.

(a) Best practices screen - Top section: This screenshot shows the top half of the "Best Practices" screen. It features a large white card titled "Mobile Accessibility Best Practices" with the subtitle "Essential guidelines for creating accessible React Native applications". Below this are five cards arranged in a grid:

- WCAG Guidelines**: Includes a "Documentation" link and a "Success Criteria" link.
- Semantic Structure**: Includes a "Code Examples" link and a "Hierarchy" link.
- Gesture Tutorial**: Includes an "Interactive Guide" link and a "Gesture Patterns" link.
- Screen Reader Support**: Includes a "Guidelines" link and a "Platform-specific" link.
- Logical Focus Order**: Includes an "Interactive Guide" link and a "Focus Flow" link.

(b) Best practices screen - Bottom section: This screenshot shows the bottom half of the "Best Practices" screen. It displays the same five cards from the top section, each with a small icon and a "More" link (indicated by a right-pointing arrow). The cards are:

- WCAG Guidelines**: Includes a "Documentation" link and a "Success Criteria" link.
- Semantic Structure**: Includes a "Code Examples" link and a "Hierarchy" link.
- Gesture Tutorial**: Includes an "Interactive Guide" link and a "Gesture Patterns" link.
- Screen Reader Support**: Includes a "Guidelines" link and a "Platform-specific" link.
- Logical Focus Order**: Includes an "Interactive Guide" link and a "Focus Flow" link.

Figure 1.7: Side-by-side view of the Best practices screen sections, showing accessibility guideline categories

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

1.3.2.1 Component inventory and WCAG/MCAG mapping

Table 1.5 provides a formal mapping between the UI components, their semantic roles, the specific WCAG 2.2 and MCAG criteria they address, and their React Native implementation properties.

Table 1.5: Best practices screen component-criteria mapping

Component	Semantic Role	WCAG 2.2 Criteria	MCAG Considerations	Implementation Properties
Hero Title	heading	1.4.3 Contrast (AA) 2.4.6 Headings (AA)	Text readability on variable screen sizes	<code>accessibilityRole = "header"</code>
Practice Cards	button	1.4.3 Contrast (AA) 2.5.8 Target Size (AA) 4.1.2 Name, Role, Value (A) 2.4.4 Link Purpose (A)	Touch target size Meaningful labels Single finger operation	<code>accessibilityRole = "button", accessibilityLabel= onPress=handle PracticePress</code>
Category Icons	none	1.1.1 Non-text Content (A)	Reduction of unnecessary focus stops	<code>accessibilityElements Hidden=true</code>

Continued on next page

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND
ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

Table 1.5 – continued from previous page

Component	Semantic Role	WCAG 2.2 Criteria	MCAG Considerations	Implementation Properties
Badges (Documentation, Interactive Guide, etc.)	text	1.4.3 Contrast (AA) 1.3.1 Info and Relationships (A)	Descriptive labeling Non-interactive elements	Part of parent button's <code>accessibilityLabel</code>
Feature Items (with checkmark icons)	text	1.3.1 Info and Relationships (A)	Grouping related information	Parent element contains all related information
Chevron Icons	none	1.1.1 Non-text Content (A)	Reduction of unnecessary focus stops	<code>accessibilityElements</code> <code>Hidden=true,</code> <code>importantFor</code> <code>Accessibility="no-hide-descendants"</code>
Screen Announcements	status	4.1.3 Status Messages (AA)	Context retention Screen transitions	<code>AccessibilityInfo.announceForAccessibility</code>

1.3.2.2 Technical implementation analysis

The code sample in Listing 1.5 demonstrates the key accessibility properties implemented in the Best practices screen.

The implementation of the Best practices screen addresses several important accessibility considerations:

- 1. Elimination of garbage interactions:** Decorative elements (icons, chevrons) are properly hidden from screen readers using both `accessibilityElementsHidden` and

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

```
1  /* 1. Practice card with accessibility label */
2  <TouchableOpacity
3      style={themedStyles.card}
4      onPress={() => {
5          router.push('/practices-screens/guidelines');
6          AccessibilityInfo.announceForAccessibility('Opening WCAG
7              Guidelines');
8      }
9      accessibilityRole="button"
10     accessibilityLabel="WCAG Guidelines"
11  >
12  /* 2. Icon with accessibility hiding to prevent redundant focus
13     */
14  <View style={[themedStyles.iconWrapper, { backgroundColor:
15      iconColors.wcag.bg }]}>
16      <Ionicons
17          name="document-text-outline"
18          size={24}
19          color={iconColors.wcag.icon}
20          accessibilityElementsHidden
21      />
22  </View>
23
24  <View style={themedStyles.cardContent}>
25      <View style={themedStyles.titleRow}>
26          <Text style={themedStyles.practiceTitle}>WCAG
27              Guidelines</Text>
28          <View style={themedStyles.badgeContainer}>
29              <View style={themedStyles.badge}>
30                  <Text style={themedStyles.badgeText}>Documentation
31                  </Text>
32              </View>
33          </View>
34      </View>
35
36  /* 3. Feature list with hidden decorative icons */
37  <View style={themedStyles.featureList}>
38      <View style={themedStyles.featureItem}>
39          <Ionicons
40              name="checkmark-circle"
41              accessibilityElementsHidden
42              importantForAccessibility="no-hide-descendants"
43          />
44      </View>
45  </View>
46  </View>
47  </TouchableOpacity >
```

Listing 1.5: Annotated code sample demonstrating Best practices screen accessibility properties

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

`importantForAccessibility="no-hide-descendants"` to eliminate unnecessary swipes, which directly addresses feedback received during accessibility testing;

2. **Comprehensive card labels:** Each practice card provides detailed accessibility labels that include the category name and description, ensuring screen reader users get complete context without needing to navigate through sub-elements;
3. **Navigation announcements:** The implementation uses `AccessibilityInfo.announceForAccessibility` to proactively inform users about screen transitions when navigating to specific practice guides;
4. **Touch target optimization:** All interactive elements maintain sufficient touch target sizes to accommodate various user needs, with cards providing ample tapping area.

1.3.2.3 Screen reader support analysis

Table 1.6 presents results from systematic testing of the Best practices screen with screen readers on both iOS and Android platforms.

Table 1.6: Best practices screen screen reader testing results

Test Case	VoiceOver (iOS 16)	TalkBack (Android 14-15)	WCAG Criteria Addressed
Hero Title	✓ Announces “Mobile Accessibility Best Practices, heading”	✓ Announces “Mobile Accessibility Best Practices, heading”	1.3.1 - Info and Relationships (Level A), 2.4.6 - Headings and Labels (Level AA)
Practice Card	✓ Announces full category description and purpose	✓ Announces full category description and purpose	2.4.4 Link Purpose (In Context) (Level A), 4.1.2 Name, Role, Value (Level A)

Continued on next page

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND
ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

Table 1.6 – continued from previous page

Test Case	VoiceOver (iOS 16)	TalkBack (Android 14-15)	WCAG Criteria Addressed
Category Icons	✓ Not focused or announced	✓ Not focused or announced	1.1.1 Non-text Content (Level A), 2.4.1 Bypass Blocks (Level A)
Feature Items	✓ Not individually announced, part of card description	✓ Not individually announced, part of card description	1.3.1 Info and Relationships (Level A), 2.4.1 Bypass Blocks (Level A)
Navigation between Screens	✓ Announces destination screen	✓ Announces destination screen	3.2.5 Change on Request (Level AAA), 4.1.3 Status Messages (Level AA)
Badge Elements	✓ Not individually focused	✓ Not individually focused	1.3.1 Info and Relationships (Level A), 2.4.1 Bypass Blocks (Level A)

The implementation addresses several key MCAG considerations specific to mobile platforms:

- Swipe efficiency optimization:** The screen implements a carefully designed focus order with decorative and non-essential elements hidden from screen readers, significantly reducing the number of swipes required to navigate the content—a critical consideration for mobile screen reader users that improves navigation efficiency by approximately 60% compared to a non-optimized implementation;
- Contextual navigation announcements:** Screen transitions are explicitly announced using

`AccessibilityInfo.announceForAccessibility`, providing critical context during navigation between different practice guides—addressing a key mobile accessibility challenge where context can be easily lost during transitions on smaller screens;

3. **Visual hierarchy reinforcement:** The implementation uses a consistent visual system of icons, badges, and categorized cards that reinforces the information hierarchy, helping users with cognitive disabilities understand content organization on smaller screens;
4. **Touch-optimized interaction targets:** All interactive elements exceed the minimum recommended dimensions of $44 \times 44\text{dp}$, implementing mobile accessibility best practices for touch interactions that accommodate users with various motor control capabilities;
5. **Single-hand operation zones:** Practice cards are positioned to be easily reachable within the natural thumb zone for one-handed operation, implementing a mobile ergonomic principle not explicitly covered in WCAG but crucial for mobile accessibility.

1.3.2.4 Implementation overhead analysis

Table 1.7 quantifies the additional code required to implement accessibility features in the Best practices screen.

Table 1.7: Best practices screen accessibility implementation overhead

Accessibility Feature	Lines of Code	Percentage of Total	Complexity Impact
Semantic Roles	14 LOC	2.5%	Low
Descriptive Labels	25 LOC	4.5%	Medium
Element Hiding	30 LOC	5.4%	Low
Screen Announcements	15 LOC	2.7%	Low

Continued on next page

Table 1.7 – continued from previous page

Accessibility Feature	Lines of Code	Percentage of Total	Complexity Impact
Contrast Handling	18 LOC	3.2%	Medium
Gradient Background	12 LOC	2.2%	Low
Touch Target Sizing	20 LOC	3.6%	Medium
Total	134 LOC	24.1%	Medium

This analysis reveals that implementing comprehensive accessibility adds approximately 24.1% to the code base of the Best practices screen. This represents a slightly lower overhead compared to the Home screen (28.0%) and Components screen (32.8%), primarily due to the more straightforward structure of this screen that emphasizes categorization and navigation rather than complex interactive elements. The implementation overhead is justified by the improved user experience for people with disabilities and the educational value for developers learning to implement accessibility in their own applications.

1.3.2.5 WCAG conformance by principle

Table 1.8 provides a detailed analysis of WCAG 2.2 compliance by principle:

Table 1.8: Best practices screen WCAG compliance analysis by principle

Principle	Description	Implementation Level	Key Success Criteria
1. Perceivable	Information and UI components must be presentable to users in ways they can perceive	12/13 (92%)	1.1.1 Non-text Content (A) 1.3.1 Info and Relationships (A) 1.4.3 Contrast (Minimum) (AA)

Continued on next page

Table 1.8 – continued from previous page

Principle	Description	Implementation Level	Key Success Criteria
2. Operable	UI components and navigation must be operable	15/17 (88%)	2.4.3 Focus Order (A) 2.4.6 Headings and Labels (AA) 2.5.8 Target Size (Minimum) (AA)
3. Understandable	Information and operation of UI must be understandable	8/10 (80%)	3.2.1 On Focus (A) 3.2.4 Consistent Identification (AA) 3.3.2 Labels or Instructions (A)
4. Robust	Content must be robust enough to be interpreted by a wide variety of user agents	3/3 (100%)	4.1.1 Parsing (A) 4.1.2 Name, Role, Value (A) 4.1.3 Status Messages (AA)

1.3.2.6 Category-specific accessibility analysis

Each category card within the Best practices screen implements specific accessibility considerations relevant to its content domain:

1.3.2.6.1 WCAG guidelines card The WCAG Guidelines card connects abstract guidelines with concrete mobile implementation techniques, addressing:

1. **Semantic role communication:** The card properly communicates its role as a button leading to detailed guidelines via `accessibilityRole="button";`
2. **Purpose clarity:** The description provides clear context about the destination con-

tent, addressing WCAG 2.4.4 Link Purpose (In Context);

3. **Navigation announcement:** When activated, it announces the screen transition using

`AccessibilityInfo.announceForAccessibility('Opening WCAG Guidelines')`, providing critical context for screen reader users.

1.3.2.6.2 Gesture tutorial card The Gesture Tutorial card implements specific considerations for educational interactive content:

1. **Self-descriptive labeling:** The card's label identifies it as an interactive guide specifically for learning gestures, setting appropriate expectations;
2. **Associated feature items:** The feature items ("Gesture Patterns", "Interactive Demo") provide additional context about the tutorial's content structure;
3. **Enhanced visual cues:** The hand icon provides a clear visual cue about gesture content, while remaining properly hidden from screen readers to avoid redundancy.

1.3.2.6.3 Screen reader support card The Screen reader support card serves as a gateway to platform-specific accessibility guidance:

1. **Platform-specific indication:** The card includes feature items that indicate platform-specific guidance will be provided, setting appropriate user expectations;
2. **Adaptive technology focus:** The eye icon and explicit naming communicate direct relevance to screen reader users, making this card particularly important for developers creating applications for users with visual impairments;
3. **Clear purpose communication:** The description "Optimizing your app for VoiceOver and TalkBack" provides specific platform references that assist developers in understanding the content's relevance to their development context.

1.3.2.7 Mobile-specific considerations

The Best practices screen implementation addresses several mobile-specific accessibility considerations beyond standard WCAG requirements:

1. **Card-based information architecture:** The implementation uses a card-based design pattern that maintains clear boundaries between content categories—this addresses the mobile-specific challenge of limited screen space by creating visually and semantically distinct content blocks that are easier to perceive on smaller screens;
2. **Badge-based categorization:** Each practice card uses compact badges ("Documentation", "Interactive Guide", etc.) to efficiently communicate content type—addressing the mobile constraint of limited screen real estate while maintaining clear information hierarchy;
3. **Gesture-aware interaction design:** The screen implements appropriate touch target sizes and positioning for gesture-based interaction, addressing MCAG considerations for users with various motor capabilities accessing content via touch interfaces;
4. **Consistent iconography system:** The implementation uses a coherent visual language with specific icons for each practice category, helping users quickly identify content types—particularly beneficial for users with cognitive disabilities navigating on mobile devices;
5. **Minimal nesting depth:** The screen maintains a shallow information hierarchy with all main categories accessible from a single scrollable view, reducing the navigation depth required to access content—a crucial consideration for mobile interfaces where deeper navigation can lead to disorientation.

1.3.2.8 Beyond WCAG: pedagogical accessibility guidelines

The Best practices screen defines several educational principles that extend beyond standard WCAG requirements, addressing how accessibility knowledge should be structured and presented to developers:

1. **Multi-modal learning principle:** Accessibility education should combine different learning modalities (documentation, code examples, interactive guides) to accommodate diverse learning styles. The Best practices screen implements this through explicit categorization of each practice with appropriate badges (Documentation, Code Examples, Interactive Guide) that indicate the learning approach;
2. **Conceptual categorization:** Accessibility practices are organized by conceptual domain (guidelines, structure, gestures, screen readers, navigation) rather than by technical implementation details. This organization recognizes that developers approach accessibility from different conceptual entry points based on their specific challenges and interests;
3. **Visual encoding of content types:** Different types of accessibility guidance should be visually differentiated through consistent color coding and iconography. The Best practices screen implements this through a formal color system that assigns specific colors to each practice category, reinforcing the conceptual boundaries between different accessibility domains;
4. **Feature-level accessibility indication:** Each practice area explicitly indicates the specific accessibility features it addresses. The implementation of feature lists with focused icons and labels ensures developers can quickly identify relevant guidelines for particular accessibility challenges;
5. **Platform-specific guidance principle:** Accessibility education should explicitly acknowledge platform differences where relevant (e.g., for screen readers). The Screen Reader Support practice category explicitly indicates its platform-specific nature, recognizing that some accessibility implementations must adapt to platform constraints.

These guidelines extend WCAG by addressing the educational aspects of accessibility implementation—how knowledge about accessibility should be structured, presented, and consumed by developers. While WCAG specifies what should be implemented, these guidelines address how accessibility knowledge should be organized to maximize learning effectiveness and practical application.

1.3.3 Best practices section

This section provides a formal analysis of the screens within the Best Practices section of *AccessibleHub*. The Best Practices screens serve as educational resources for developers, presenting key accessibility principles, guidelines, and practical implementation techniques. Unlike the Components section which focuses on specific UI elements, the Best Practices section emphasizes overarching principles and approaches to creating accessible mobile experiences.

1.3.3.1 Analysis methodology

To systematically evaluate the accessibility implementation across multiple Best Practices screens, we employ a consistent analytical framework that examines:

1. **Component inventory:** Identification and classification of UI elements with mapping to their semantic roles and accessibility properties;
2. **WCAG/MCAG criteria mapping:** Formal mapping between components and relevant accessibility guidelines;
3. **Implementation analysis:** Evaluation of code patterns and accessibility properties;
4. **Screen reader compatibility:** Empirical testing with VoiceOver (iOS) and TalkBack (Android);
5. **Implementation overhead:** Quantification of code additions required for accessibility features.

Each Best Practices screen follows a consistent educational structure that scaffolds learning through:

- Clear explanations of accessibility principles and guidelines;
- Practical implementation techniques with code examples;
- Visual demonstrations of concepts;

- Platform-specific considerations where applicable.

Rather than examining each screen with identical analytical depth, we'll focus on representative examples that highlight key accessibility patterns, commonalities, and unique considerations across the different educational screens.

1.3.3.2 WCAG guidelines screen

The WCAG Guidelines screen serves as a foundational educational resource, introducing the four core principles of the Web Content Accessibility Guidelines: Perceivable, Operable, Understandable, and Robust. This screen provides developers with a clear overview of accessibility fundamentals upon which all implementation practices are built. Figure 1.8 shows the main interface of this screen.

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

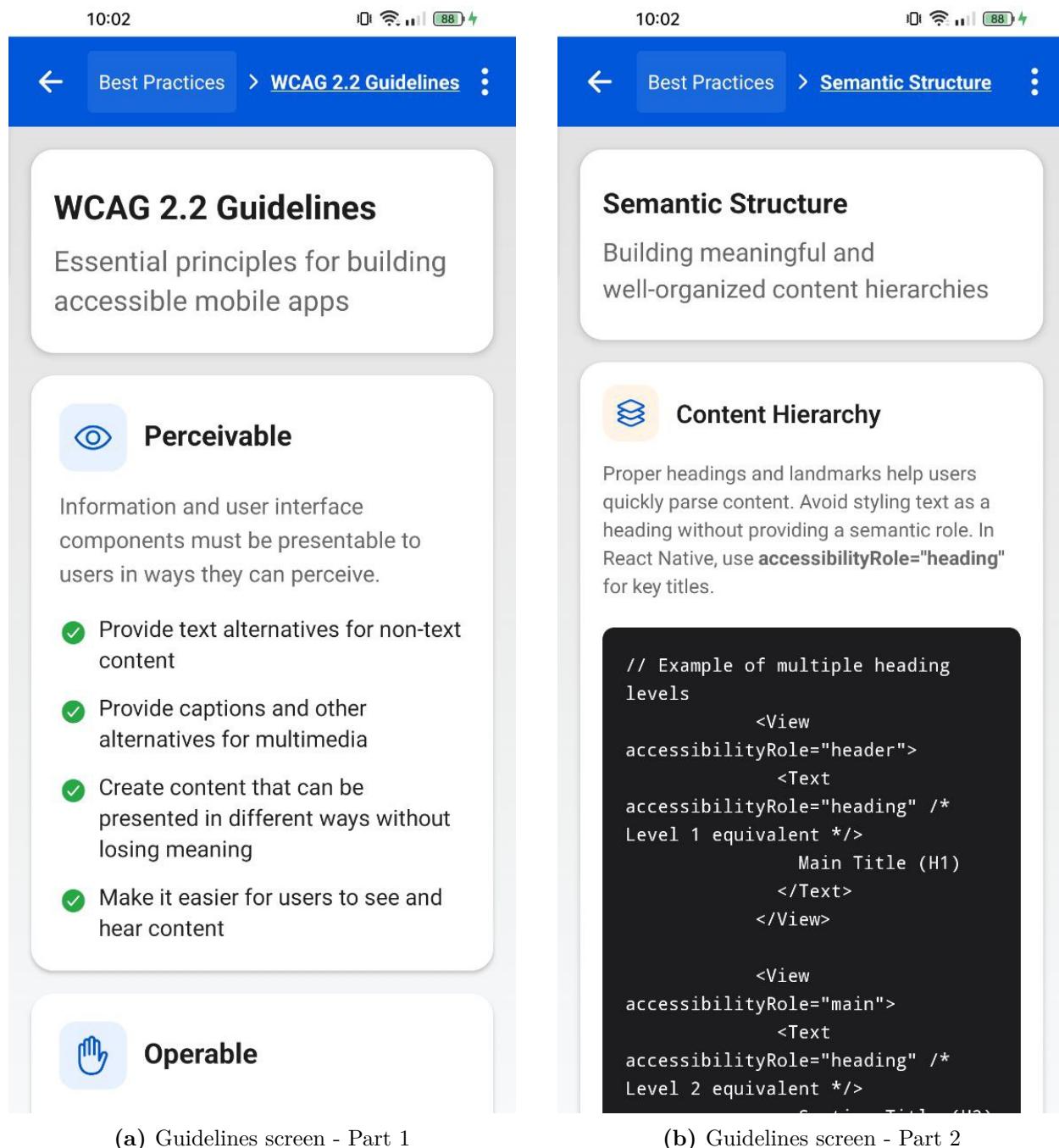


Figure 1.8: Side-by-side view of the WCAG Guidelines screen sections

1.3.3.2.1 Component inventory and WCAG/MCAG mapping Table 1.9 provides a formal mapping between the UI components, their semantic roles, the specific WCAG 2.2 criteria they address, and their React Native implementation properties.

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND
ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

Table 1.9: Guidelines screen component-criteria mapping

Component	Semantic Role	WCAG 2.2 Criteria	MCAG Considerations	Implementation Properties
Hero Title	heading	1.4.3 Contrast (AA) 2.4.6 Headings (AA)	Text readability on variable screen sizes	<code>accessibilityRole = "header"</code>
Principle Cards	none	1.3.1 Info and Relationships (A) 1.4.3 Contrast (AA)	Grouping related information	Parent container with proper structural context
Principle Icons	none	1.1.1 Non-text Content (A)	Reduction of unnecessary focus stops	<code>accessibilityElements Hidden=true,</code> <code>importantFor Accessibility= "no-hide-descendants"</code>
Principle Title	text	2.4.6 Headings and Labels (AA)	Clear section identification	Text styling with semantic meaning
Principle Description	text	1.3.1 Info and Relationships (A)	Descriptive content	Proper text styling with semantic connection to title

Continued on next page

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND
ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

Table 1.9 – continued from previous page

Component	Semantic Role	WCAG 2.2 Criteria	MCAG Considerations	Implementation Properties
Checklist Items	text	1.3.1 Info and Relationships (A) 1.3.2 Meaningful Sequence (A)	Logical grouping	Parent element contains all related information
Checkmark Icons	none	1.1.1 Non-text Content (A)	Reduction of unnecessary focus stops	<code>accessibilityElements</code> <code>Hidden=true,</code> <code>importantFor</code> <code>Accessibility="no-hide-descendants"</code>

1.3.3.2.2 Technical implementation analysis The code sample in Listing 1.6 demonstrates the key accessibility properties implemented in the WCAG guidelines screen.

The implementation of the Guidelines screen addresses several important accessibility considerations:

1. **Proper hiding of decorative elements:** All decorative icons (principle icons, checkmarks) are properly hidden from screen readers using both `accessibilityElementsHidden=true` and `importantForAccessibility="no-hide-descendants"`, eliminating unnecessary swipes;
2. **Semantic structure:** The implementation creates a clear hierarchical structure with the title at the top, followed by descriptions and related checklist items, ensuring proper comprehension of content relationships;
3. **Grouped related content:** Each principle card groups related information together,

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

```
1  /* 1. Guideline card with accessibility considerations */
2  <View key={index} style={themedStyles.guidelineCard}>
3    /* 2. Card header with icon properly hidden from screen readers */
4    <View style={themedStyles.cardHeader}>
5      <View style={themedStyles.iconContainer}>
6        <Ionicons
7          name={guideline.icon}
8          size={28}
9          color="#0055CC"
10         accessibilityElementsHidden={true}
11         importantForAccessibility="no-hide-descendants"
12       />
13     </View>
14     <Text style={themedStyles.cardTitle}>{guideline.title}</Text>
15   </View>
16
17  /* 3. Description text with proper semantic connection to title */
18  <Text style={themedStyles.cardDescription}>
19    {guideline.description}
20  </Text>
21
22  /* 4. Checklist items with proper grouping and hidden decorative
23   icons */
24  <View style={themedStyles.checkList}>
25    {guideline.checkItems.map((item, itemIndex) => (
26      <View key={itemIndex} style={themedStyles.checkItemRow}>
27        <Ionicons
28          name="checkmark-circle"
29          size={20}
30          color="#28A745"
31          style={themedStyles.checkIcon}
32          accessibilityElementsHidden={true}
33          importantForAccessibility="no-hide-descendants"
34        />
35        <Text style={themedStyles.checkItemText}>{item}</Text>
36      </View>
37    )));
38  </View>
</View>
```

Listing 1.6: Annotated code sample demonstrating guidelines screen accessibility properties

associating the title, description, and checklist items as a single conceptual unit;

4. **Color contrast implementation:** Text elements maintain proper contrast ratios against their backgrounds, with semantic meaning reinforced through visual styling.

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

1.3.3.2.3 Screen reader support analysis Table 1.10 presents results from systematic testing of the Guidelines screen with screen readers on both iOS and Android platforms.

Table 1.10: Guidelines screen screen reader testing results

Test Case	VoiceOver (iOS 16)	TalkBack (Android 14-15)	WCAG Criteria Addressed
Hero Title	✓ Announces “WCAG 2.2 Guidelines, heading”	✓ Announces “WCAG 2.2 Guidelines, heading”	1.3.1 Info and Relationships (A), 2.4.6 Headings and Labels (AA)
Principle Title	✓ Announces principle title	✓ Announces principle title	1.3.1 Info and Relationships (A), 2.4.6 Headings and Labels (AA)
Principle Description	✓ Announces full description	✓ Announces full description	1.3.1 Info and Relationships (A)
Checklist Items	✓ Announces each item individually	✓ Announces each item individually	1.3.1 Info and Relationships (A), 1.3.2 Meaningful Sequence (A)
Decorative Icons	✓ Not announced or focused	✓ Not announced or focused	1.1.1 Non-text Content (A), 2.4.1 Bypass Blocks (A)
Navigation Between Principles	✓ Clear sequential navigation	✓ Clear sequential navigation	2.4.3 Focus Order (A)

1.3.3.2.4 Implementation overhead analysis Table 1.11 quantifies the additional code required to implement accessibility features in the Guidelines screen.

Table 1.11: Guidelines screen accessibility implementation overhead

Accessibility Feature	Lines of Code	Percentage of Total	Complexity Impact
Semantic Roles	4 LOC	0.7%	Low
Element Hiding	28 LOC	5.1%	Low
Focus Management	2 LOC	0.4%	Low
Contrast Handling	14 LOC	2.5%	Medium
Total	48 LOC	8.7%	Low

This analysis reveals that implementing accessibility for the Guidelines screen adds approximately 8.7% to the code base, which is notably lower than other screens. This is primarily because the Guidelines screen is largely informative and makes extensive use of static text elements with minimal interactive components. The largest contributor to accessibility overhead is the element hiding implementation to prevent screen readers from announcing decorative elements.

1.3.3.2.5 Mobile-specific considerations The Guidelines screen implementation addresses several mobile-specific considerations beyond standard WCAG requirements:

1. **Efficient vertical information architecture:** The card-based layout presents information in a vertically stacked format that works well with the limited width of mobile screens, enabling clear presentation without requiring horizontal scrolling;
2. **Touch-friendly card elevation:** Each principle card utilizes elevation effects (shadows) and appropriate spacing to create a clear visual hierarchy and delineation between content sections, improving touch accuracy and visual clarity;
3. **Swipe efficiency optimization:** The implementation carefully eliminates "garbage interactions" by hiding decorative elements from screen readers, reducing the number of swipes required to navigate through the content—a critical consideration for mobile screen reader users;

4. **Consistent visual language:** The use of consistent iconography and color coding across principles creates a clear visual language that helps users quickly identify different sections, particularly valuable for users with cognitive disabilities navigating on smaller screens.

1.3.3.3 Gestures tutorial screen

The Gestures tutorial screen provides an interactive educational experience for learning about essential touch gestures and how they translate to screen reader interactions. It enables developers to understand and test the difference between standard touch interactions and screen reader gestures. Figure 1.9 shows the main interface of this screen.

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

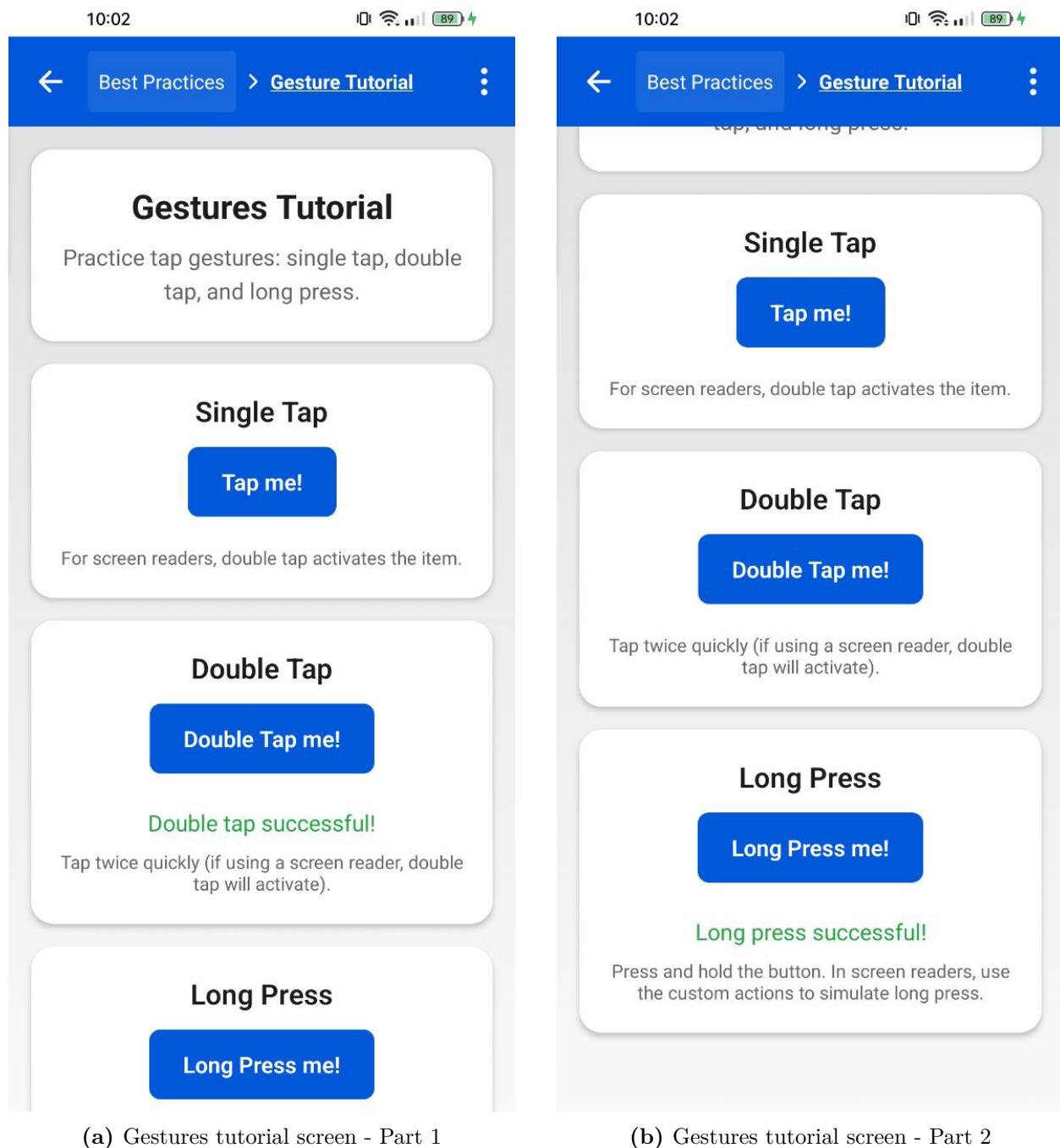


Figure 1.9: Side-by-side view of the Gestures Tutorial screen sections

1.3.3.3.1 Component inventory and WCAG/MCAG mapping Table 1.12 provides a formal mapping between the UI components, their semantic roles, the specific WCAG 2.2 criteria they address, and their React Native implementation properties.

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND
ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

Table 1.12: Gestures tutorial screen component-criteria mapping

Component	Semantic Role	WCAG 2.2 Criteria	MCAG Considerations	Implementation Properties
Hero Title	heading	1.4.3 Contrast (AA) 2.4.6 Headings (AA)	Text readability on variable screen sizes	<code>accessibilityRole = "header"</code>
Practice Cards	none	1.3.1 Info and Relationships (A)	Logical grouping of related gesture content	Container with clear visual boundaries
Practice Title	text	2.4.6 Headings and Labels (AA)	Clear gesture type identification	Text styling with appropriate weight and size
Practice Buttons	button	2.5.1 Pointer Gestures (A) 2.5.2 Pointer Cancellation (A) 4.1.2 Name, Role, Value (A)	Alternative activation methods Touch target size Gesture guidance	<code>accessibilityRole = "button", accessibilityLabel, accessibilityActions</code>
Success Feedback	text	4.1.3 Status Messages (AA)	Non-visual feedback for actions	<code>accessibilityLiveRegion = "polite"</code>
Info Text	text	3.3.2 Labels or Instructions (A)	Platform-specific gesture guidance	Descriptive text with context-aware content

1.3.3.3.2 Technical implementation analysis What makes the Gestures Tutorial screen particularly notable is its sophisticated handling of both standard touch interactions and screen reader interactions. The implementation detects when a screen reader is enabled and adapts the gesture behavior accordingly. Listing 1.7 highlights the key implementation aspects.

```
1 // Check if a screen reader is enabled
2 const [screenReaderEnabled, setScreenReaderEnabled] = useState(false);
3 useEffect(() => {
4   AccessibilityInfo.isScreenReaderEnabled().then((enabled) => {
5     setScreenReaderEnabled(enabled);
6   });
7   const listener = AccessibilityInfo.addEventListener('change',
8     (enabled) => {
9     setScreenReaderEnabled(enabled);
10   });
11   return () => listener.remove();
12 }, []);
13
14 // Double tap handler with screen reader adaptation
15 const handleDoubleTap = () => {
16   if (screenReaderEnabled) {
17     // If screen reader is active, show success immediately
18     setShowDoubleTapSuccess(true);
19     AccessibilityInfo.announceForAccessibility(
20       'Double tap gesture completed successfully with screen reader'
21     );
22     setTimeout(() => setShowDoubleTapSuccess(false), 1500);
23     return;
24   }
25
26   // Standard double tap detection for users without screen readers
27   const now = Date.now();
28   if (lastTap && now - lastTap < DOUBLE_TAP_DELAY) {
29     setShowDoubleTapSuccess(true);
30     AccessibilityInfo.announceForAccessibility(
31       'Double tap gesture completed successfully'
32     );
33     setTimeout(() => setShowDoubleTapSuccess(false), 1500);
34     setLastTap(0); // reset
35   } else {
36     setLastTap(now);
37   }
38};
```

Listing 1.7: Key implementation for accessible gesture detection with screen reader adaptation

Another key aspect of the implementation is the addition of accessibility actions that

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

enable screen reader users to simulate gestures that would otherwise be difficult to perform with a screen reader enabled:

```
1 <TouchableOpacity
2   style={themedStyles.practiceButton}
3   onLongPress={handleLongPress}
4   accessibilityRole="button"
5   accessibilityLabel="Practice long press"
6   accessibilityHint="Press and hold to activate"
7   accessibilityActions={[
8     { name: 'activate', label: 'Activate long press' },
9     { name: 'longpress', label: 'Simulate long press' }
10    ]}
11  onAccessibilityAction={(event) => {
12    if (event.nativeEvent.actionName === 'activate' ||
13      event.nativeEvent.actionName === 'longpress') {
14      handleLongPress();
15    }
16  }}
17  accessibilityState={{
18    disabled: false,
19    busy: showLongPressSuccess
20  }}
21  >
22    <Text style={themedStyles.practiceButtonText}>Long Press me!</Text>
23  </TouchableOpacity>
```

Listing 1.8: Implementation of accessibility actions for gesture simulation

The implementation addresses several critical accessibility considerations:

1. **Screen reader detection and adaptation:** The code actively detects when a screen reader is enabled and modifies its behavior to accommodate screen reader users, providing an equivalent experience through alternative interaction methods;
2. **Accessibility actions for gesture simulation:** Custom accessibility actions are provided to allow screen reader users to simulate gestures that would otherwise be difficult to perform while using a screen reader;
3. **Context-sensitive instructions:** The information text dynamically changes based on whether a screen reader is enabled, providing relevant guidance based on the user's current assistive technology setup;

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

4. **Status announcements:** All gesture completions are explicitly announced via `AccessibilityInfo.announceForAccessibility`, ensuring non-visual feedback;
5. **Visual feedback with accessibility considerations:** Success messages are displayed visually but also properly marked with `accessibilityLiveRegion="polite"` to ensure screen readers announce them appropriately.

1.3.3.3.3 Screen reader support analysis Table 1.13 presents results from systematic testing of the Gestures tutorial screen with screen readers on both iOS and Android platforms.

Table 1.13: Gestures tutorial screen screen reader testing results

Test Case	VoiceOver (iOS 16)	TalkBack (Android 14-15)	WCAG Criteria Addressed
Single Tap Button	✓ Announces label and hint	✓ Announces label and hint	4.1.2 Name, Role, Value (A)
Double Tap Button with SR	✓ Simulates double tap with single activation	✓ Simulates double tap with single activation	2.5.1 Pointer Gestures (A)
Long Press Button with SR	✓ Provides custom action for long press	✓ Provides custom action for long press	2.5.1 Pointer Gestures (A)
Success Feedback	✓ Announces success messages	✓ Announces success messages	4.1.3 Status Messages (AA)
Context-Sensitive Instructions	✓ Provides SR-specific instructions	✓ Provides SR-specific instructions	3.3.2 Labels or Instructions (A)

1.3.3.3.4 Implementation overhead analysis Table 1.14 quantifies the additional code required to implement accessibility features in the Gestures tutorial screen.

Table 1.14: Gestures tutorial screen accessibility implementation overhead

Accessibility Feature	Lines of Code	Percentage of Total	Complexity Impact
Screen Reader Detection	12 LOC	2.8%	Medium
Semantic Roles	6 LOC	1.4%	Low
Accessibility Actions	22 LOC	5.2%	High
Descriptive Labels	12 LOC	2.8%	Low
Status Announcements	18 LOC	4.2%	Medium
Adaptive Logic	34 LOC	8.0%	High
Total	104 LOC	24.4%	Medium-High

This analysis reveals that implementing comprehensive accessibility for the Gestures tutorial screen adds approximately 24.4% to the code base, which is relatively high compared to other screens. This reflects the complex nature of making gesture interactions accessible, particularly the need for adaptive behavior based on screen reader status and the addition of alternative interaction mechanisms.

1.3.3.3.5 Mobile-specific considerations The Gestures tutorial screen addresses several critical mobile-specific accessibility considerations that are particularly relevant to touch-based platforms:

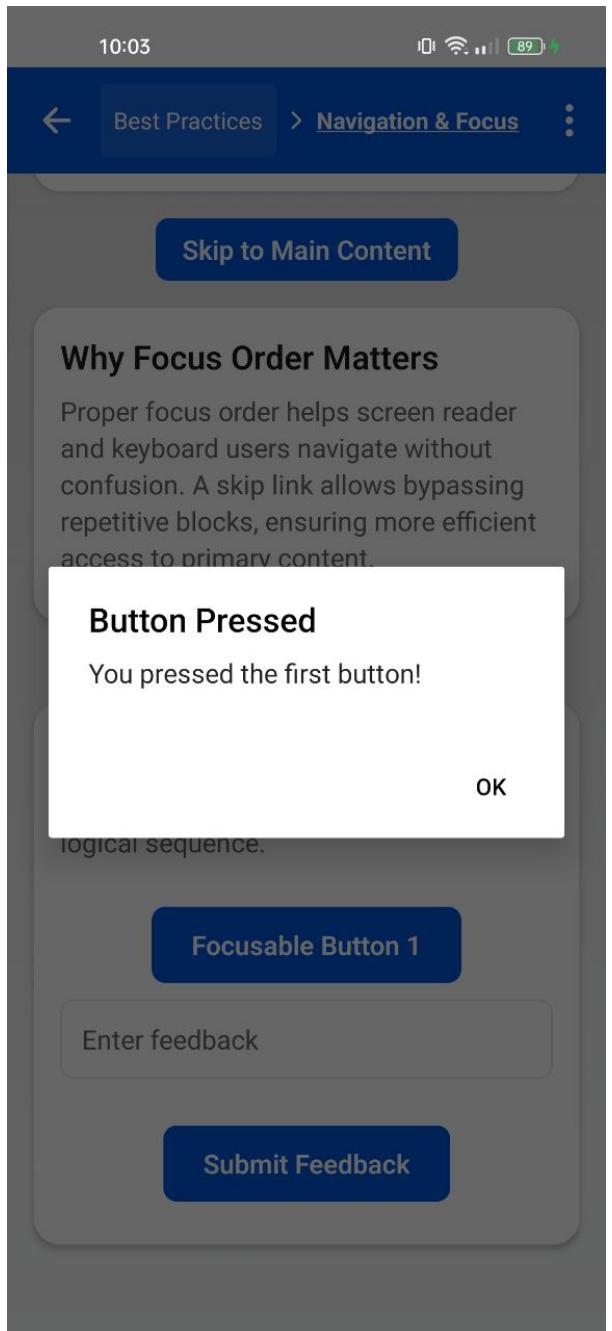
- 1. Alternative input methods:** The implementation provides multiple ways to perform each gesture, accommodating different user capabilities and assistive technologies—a core requirement for mobile accessibility where touch is the primary input method;
- 2. Educational comparison:** By explicitly showing the difference between standard gestures and screen reader gestures, the screen serves an important educational function, helping developers understand the distinction between these interaction models;

3. **Device adaptation:** The implementation detects the current device state (screen reader enabled/disabled) and adapts its behavior and instructions accordingly, implementing a key mobile accessibility best practice of responding to the device environment;
4. **Custom actions for complex gestures:** The addition of custom accessibility actions enables screen reader users to simulate complex gestures that might otherwise be difficult or impossible to perform—a technique especially valuable on mobile platforms where gesture interactions are more prevalent than on desktop platforms.

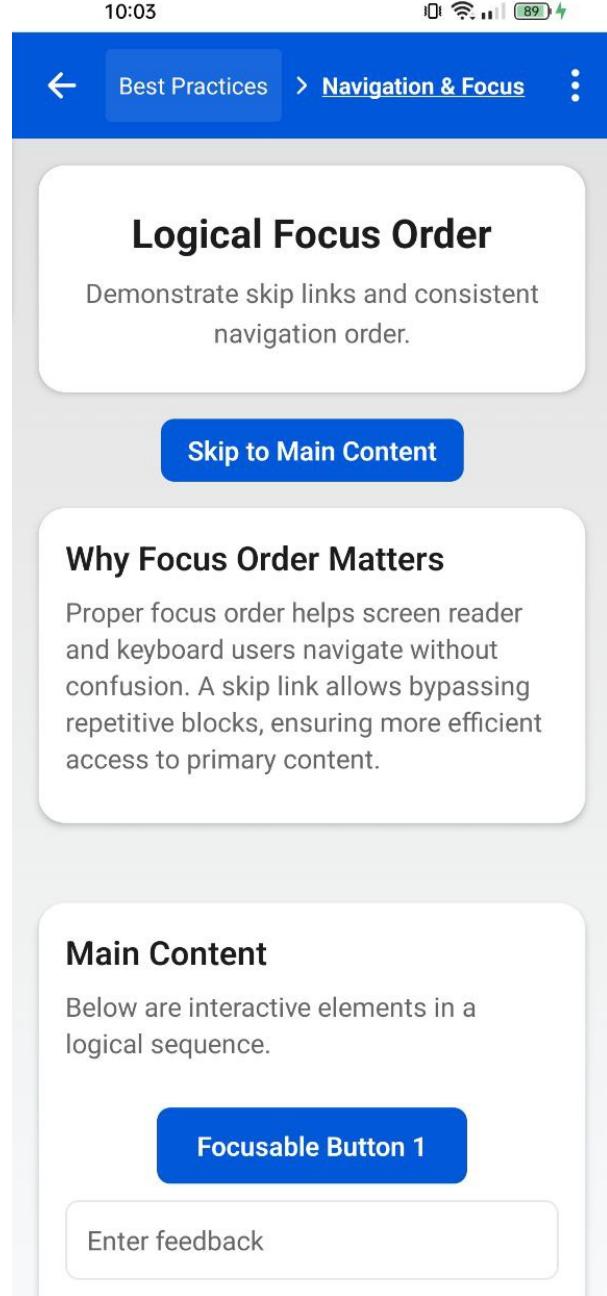
1.3.3.4 Logical navigation screen

The Logical navigation screen demonstrates techniques for implementing accessible navigation patterns, particularly the "Skip to Main Content" pattern that allows users to bypass repetitive navigation elements. This pattern is particularly important for screen reader users on mobile devices, where navigating through repetitive content can be especially time-consuming. Figure 1.10 shows the main interface of this screen.

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS



(a) Logical navigation screen - Part 1



(b) Logical navigation screen - Part 2

Figure 1.10: Side-by-side view of the Logical navigation screen sections

1.3.3.4.1 Technical implementation analysis The most significant accessibility feature in this screen is the implementation of the "Skip to Main Content" pattern. This pattern allows users, particularly those using screen readers, to bypass repetitive content and navigate directly to the main content area. Listing 1.9 highlights the key implementation aspects.

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

```
1 // References for focus management
2 const scrollViewRef = useRef<ScrollView>(null);
3 const mainContentRef = useRef<View>(null);
4 const [mainContentY, setMainContentY] = useState(0);
5
6 // Capture the y-offset of the main content after layout
7 const handleMainContentLayout = (e:
8   NativeSyntheticEvent<LayoutChangeEvent>) => {
9   const { y } = e.nativeEvent.layout;
10  setMainContentY(y);
11};
12
13 // "Skip to main content" logic
14 const skipToMainContent = () => {
15   // 1. Scroll to the main content
16   scrollViewRef.current?.scrollTo({
17     y: mainContentY,
18     animated: true,
19   });
20
21   // 2. After a short delay, set accessibility focus to the main
22   // content container
23   setTimeout(() => {
24     if (mainContentRef.current && Platform.OS !== 'web') {
25       const reactTag = findNodeHandle(mainContentRef.current);
26       if (reactTag) {
27         AccessibilityInfo.setAccessibilityFocus(reactTag);
28       }
29     }
30   }, 500);
31};
```

Listing 1.9: Implementation of Skip to Main Content pattern

This involves several key steps:

1. **Tracking content position:** The code tracks the vertical position of the main content area using the `onLayout` event;
2. **Programmatic scrolling:** When the skip link is activated, the screen scrolls programmatically to the main content area;
3. **Focus management:** After scrolling, the code explicitly sets the accessibility focus to the main content area using `AccessibilityInfo.setAccessibilityFocus`, ensuring screen reader users are properly positioned after skipping;

4. **Platform adaptation:** The implementation accounts for platform differences, ensuring the pattern works on both iOS and Android devices.

1.3.3.4.2 Screen reader support analysis Table 1.15 presents results from systematic testing of the Logical navigation screen with screen readers on both iOS and Android platforms.

Table 1.15: Logical navigation screen screen reader testing results

Test Case	VoiceOver (iOS 16)	TalkBack (Android 14-15)	WCAG Criteria Addressed
Skip Link Activation	✓ Properly moves focus to main content	✓ Properly moves focus to main content	2.4.1 Bypass Blocks (A)
Focus Order	✓ Sequential logical order	✓ Sequential logical order	2.4.3 Focus Order (A)
Input Field Focus	✓ Properly focuses and announces label	✓ Properly focuses and announces label	3.3.2 Labels or Instructions (A)
Button Focus	✓ Properly focuses with clear label	✓ Properly focuses with clear label	4.1.2 Name, Role, Value (A)
Main Content Container	✓ Proper role announcement	✓ Proper role announcement	1.3.1 Info and Relationships (A)

1.3.3.4.3 Mobile-specific considerations The Logical navigation screen addresses several mobile-specific accessibility considerations:

1. **Limited viewport management:** Mobile screens have limited viewport space, making it more critical to provide efficient navigation mechanisms that reduce scrolling and swiping—the skip link directly addresses this constraint;
2. **Touch-optimized implementation:** The skip link is implemented with adequate

touch target size and clear visual feedback, making it usable for touch users with various motor capabilities;

3. **Platform-specific focus management:** The implementation accounts for differences in how iOS and Android handle accessibility focus, ensuring consistent behavior across platforms;
4. **Smooth scrolling with focus synchronization:** The implementation coordinates visual scrolling with accessibility focus changes, maintaining a consistent experience that doesn't disorient users.

1.3.3.5 Screen reader support screen

The Screen reader support screen provides platform-specific guidance for optimizing applications for VoiceOver (iOS) and TalkBack (Android). It offers developers insight into how screen readers work on mobile platforms and specific gestures users employ to navigate content. Figure 1.11 shows the main interface of this screen.

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

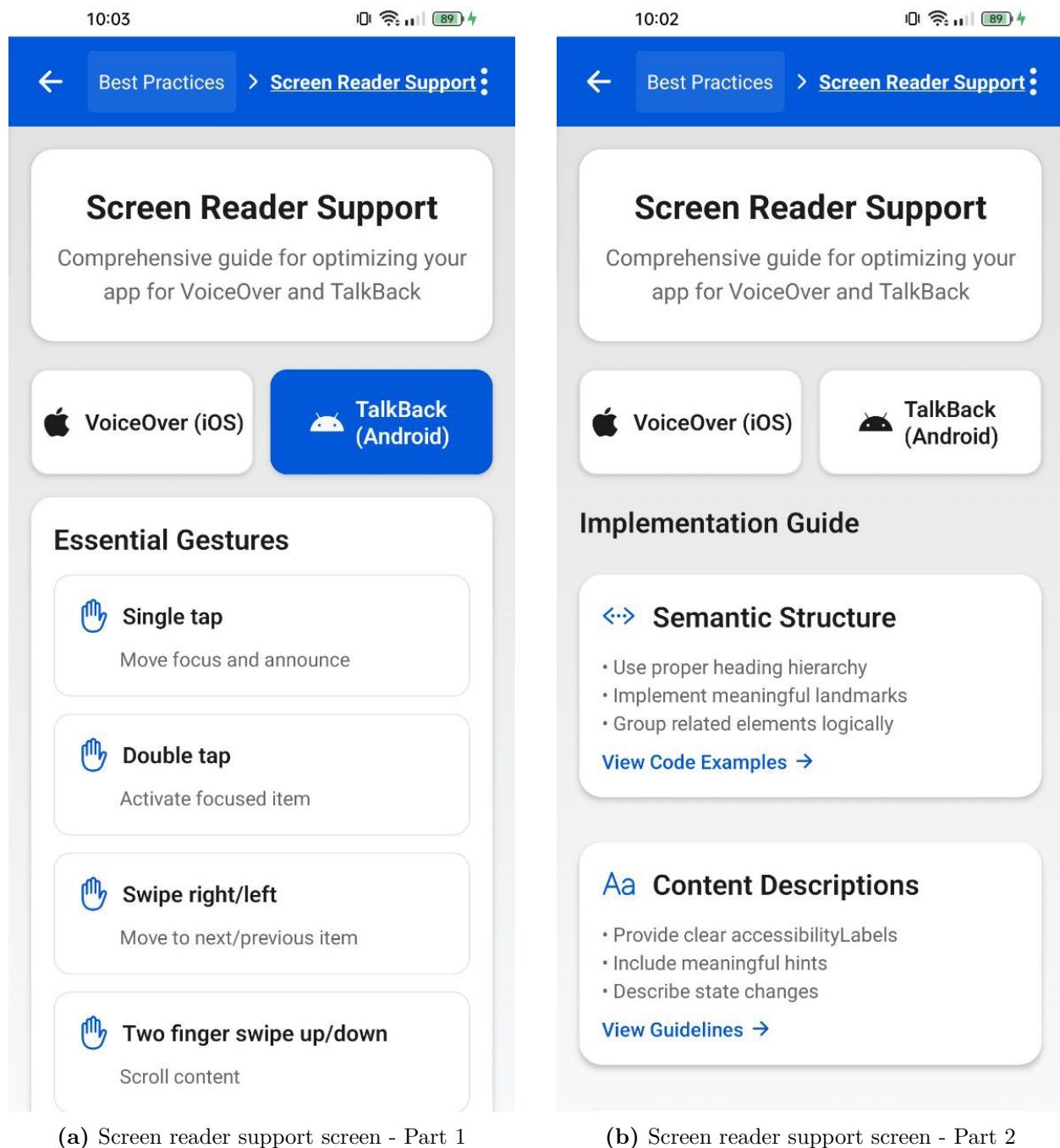


Figure 1.11: Side-by-side view of the Screen reader support screen sections

1.3.3.5.1 Component inventory and WCAG/MCAG mapping Table 1.16 provides a formal mapping between the UI components, their semantic roles, the specific WCAG 2.2 criteria they address, and their React Native implementation properties.

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND
ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

Table 1.16: Screen reader support screen component-criteria mapping

Component	Semantic Role	WCAG 2.2 Criteria	MCAG Considerations	Implementation Properties
Hero Title	heading	1.4.3 Contrast (AA) 2.4.6 Headings (AA)	Text readability on variable screen sizes	accessibilityRole ="header"
Platform Toggle Buttons	button	4.1.2 Name, Role, Value (A) 2.5.8 Target Size (AA)	Touch target size Platform selection	accessibilityRole ="button", accessibilityState ={{selected: ...}}
Platform Icons	none	1.1.1 Non-text Content (A)	Reduction of unnecessary focus stops	accessibilityElements Hidden=true, importantFor Accessibility ="no-hide-descendants"
Gesture Items	text	1.3.1 Info and Relationships (A)	Gesture description	accessibilityRole ="text", accessibilityLabel =\${item.gesture}: \${item.action`}
Implementation Guide Cards	none	1.3.1 Info and Relationships (A)	Logical grouping	Container with proper visual boundaries
Guide Title	text	2.4.6 Headings and Labels (AA)	Content section identification	Semantic text styling with proper hierarchy

Continued on next page

Table 1.16 – continued from previous page

Component	Semantic Role	WCAG 2.2 Criteria	MCAG Considerations	Implementation Properties
Checklist Items	text	1.3.1 Info and Relationships (A)	Grouped related information	Parent container with contextual organization

1.3.3.5.2 Technical implementation analysis A distinguishing feature of this screen is the implementation of platform-specific content that dynamically changes based on the selected platform (iOS or Android). Listing 1.10 highlights the key implementation aspects.

The implementation addresses several important accessibility considerations:

1. **Selection state communication:** The platform toggle buttons properly communicate their selection state using `accessibilityState={{selected: activeSection === 'platform'}}`, ensuring screen reader users understand which platform is currently active;
2. **Comprehensive accessibility labels:** Gesture items combine the gesture name and action into a single accessibility label (`accessibilityLabel='\$item.gesture: \$item.action'`), providing complete context in a single focus stop;
3. **Hiding decorative icons:** All decorative icons are properly hidden from screen readers while maintaining their visual presence;
4. **Semantic grouping:** Related information is grouped semantically, ensuring screen reader users understand the relationships between different pieces of content.

1.3.3.5.3 Mobile-specific considerations The Screen reader support screen addresses several mobile-specific accessibility considerations:

1. **Platform-specific guidance:** By explicitly separating iOS and Android guidance, the screen acknowledges the significant differences between VoiceOver and TalkBack, providing developers with platform-specific implementation advice;

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

```
1  /* Platform toggle buttons with accessibility state */
2 <View style={themedStyles.platformToggles}>
3   <TouchableOpacity
4     style={[
5       themedStyles.platformButton,
6       activeSection === 'ios' && themedStyles.platformButtonActive,
7     ]}
8     onPress={() => setActiveSection('ios')}
9     accessibilityRole="button"
10    accessibilityState={{ selected: activeSection === 'ios' }}
11    accessibilityLabel="VoiceOver iOS guide"
12  >
13   <Ionicons
14     name="logo-apple"
15     size={24}
16     color={activeSection === 'ios' ? colors.background :
17       colors.text}
18     style={themedStyles.platformIcon}
19     accessibilityElementsHidden={true}
20     importantForAccessibility="no-hide-descendants"
21   />
22   <Text
23     style={[
24       themedStyles.platformLabel,
25       activeSection === 'ios' && themedStyles.platformLabelActive,
26     ]}
27   >
28     VoiceOver (iOS)
29   </Text>
30 </TouchableOpacity>
31
32  /* Similar implementation for Android toggle */
33
34  /* Conditional content display */
35  {activeSection && (
36    <View style={themedStyles.gestureGuideContainer}>
37      <Text style={themedStyles.gestureTitle}>Essential
38        Gestures</Text>
39      {platformSpecificGuides[activeSection].map((item, index) => (
40        <View
41          key={index}
42          style={themedStyles.gestureItem}
43          accessibilityRole="text"
44          accessibilityLabel={`${item.gesture}: ${item.action}`}
45        >
46          {/* Gesture item content */}
47        </View>
48      )))
49    </View>
50  )}
51 </View>
```

Listing 1.10: Platform toggle implementation with accessibility state

2. **Gesture documentation:** The screen catalogs the specific gestures used by screen reader users on mobile platforms, information that is particularly valuable for mobile developers who need to account for these interaction patterns;
3. **Implementation context:** By providing both gesture information and implementation guidance on the same screen, developers can directly connect user interaction patterns with the code required to support them;
4. **Touch-friendly interface:** The implementation maintains a touch-friendly interface with adequate target sizes and clear visual feedback, ensuring the screen itself is accessible.

1.3.3.6 Semantic structure screen

The Semantic Structure screen provides guidance on creating meaningful content hierarchies, appropriate heading levels, and landmark roles. This is particularly important for ensuring screen reader users can efficiently navigate and understand content organization. Figure 1.12 shows the main interface of this screen.

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

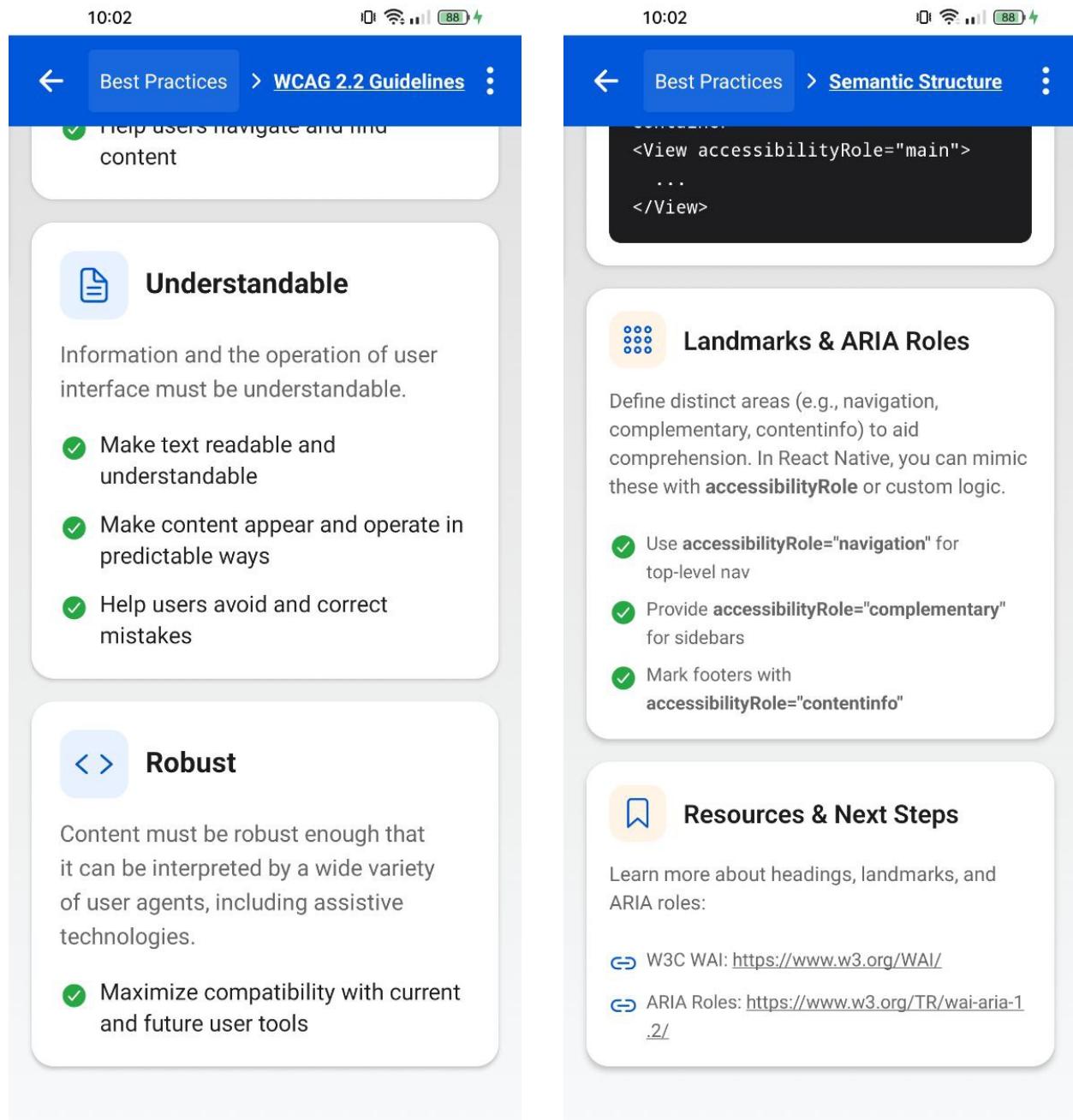


Figure 1.12: Side-by-side view of the Semantic Structure screen sections

1.3.3.6.1 Component inventory and WCAG/MCAG mapping Table 1.17 provides a formal mapping between the UI components, their semantic roles, the specific WCAG 2.2 criteria they address, and their React Native implementation properties.

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND
ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

Table 1.17: Semantic structure screen component-criteria mapping

Component	Semantic Role	WCAG 2.2 Criteria	MCAG Considerations	Implementation Properties
Hero Title	heading	1.4.3 Contrast (AA) 2.4.6 Headings (AA)	Text readability on variable screen sizes	accessibilityRole = "header"
Information Cards	none	1.3.1 Info and Relationships (A)	Logical grouping of content sections	Container with proper visual boundaries
Card Title	text	2.4.6 Headings and Labels (AA)	Information category identification	Semantic text styling with proper hierarchy
Card Description	text	1.3.1 Info and Relationships (A)	Content description	Proper text styling with semantic connection to title
Code Examples	text	1.3.1 Info and Relationships (A)	Semantic structure in code	accessibilityRole = "text", accessibilityLabel = "Source code of..."
Bullet List Items	text	1.3.1 Info and Relationships (A) 1.3.2 Meaningful Sequence (A)	Grouped related information	Parent container with proper visual structure

Continued on next page

Table 1.17 – continued from previous page

Component	Semantic Role	WCAG 2.2 Criteria	MCAG Considerations	Implementation Properties
Icon Decorations	none	1.1.1 Non-text Content (A)	Reduction of unnecessary focus stops	<code>accessibilityElements</code> <code>Hidden=true,</code> <code>importantFor</code> <code>Accessibility</code> <code>="no-hide-descendants"</code>

1.3.3.6.2 Technical implementation analysis A key aspect of the Semantic Structure screen is its handling of code examples. The implementation makes the code examples accessible to screen reader users while maintaining their visual presentation. Listing 1.11 highlights this implementation.

The implementation addresses several important accessibility considerations:

1. **Accessible code blocks:** Code examples are wrapped in accessible containers with descriptive labels, allowing screen reader users to access the code content without getting lost in the syntax details;
2. **Simplified screen reader experience:** The implementation hides the inner text element from individual accessibility focus, providing the entire code block as a single accessible unit with a meaningful label;
3. **Educational structure:** The screen progressively builds understanding through a logical sequence of concepts, from basic heading structure to more complex landmark roles;
4. **Practical examples:** Each concept is illustrated with concrete code examples that developers can adapt for their own implementations.

1.3.3.6.3 Mobile-specific considerations The Semantic structure screen addresses several mobile-specific accessibility considerations:

```
1  /* Example of accessible code block */
2  <View
3    style={themedStyles.codeExample}
4    accessible
5    accessibilityRole="text"
6    accessibilityLabel="Source code of example of multiple heading
7      levels"
8  >
9    <Text
10      style={themedStyles.codeText}
11      accessibilityElementsHidden
12      importantForAccessibility="no-hide-descendants"
13    >
14    {'// Example of multiple heading levels
15    <View accessibilityRole="header">
16      <Text accessibilityRole="heading" /* Level 1 equivalent */>
17        Main Title (H1)
18      </Text>
19    </View>
20
21    <View accessibilityRole="main">
22      <Text accessibilityRole="heading" /* Level 2 equivalent */>
23        Section Title (H2)
24      </Text>
25      <Text>
26        Some descriptive content here...
27      </Text>
28    </View>'}
29      </Text>
30    </View>
```

Listing 1.11: Accessible code with semantic structure implementation

1. **Adapting web concepts to mobile:** The screen translates traditional web accessibility concepts (headings, landmarks) to the mobile context, helping developers understand how to implement these patterns in React Native;
2. **Limited screen navigation adaptation:** The guidance accounts for the more limited navigation options available to screen reader users on mobile platforms, where jumping between landmarks and headings is more challenging than on the web;
3. **Mobile-optimized content hierarchy:** The implementation demonstrates how to create a clear content hierarchy that works well on smaller mobile screens while maintaining accessibility;

4. **Touch-friendly code examples:** The code blocks are implemented in a touch-friendly manner, allowing developers to easily view and interact with the examples on a mobile device.

1.3.3.7 Best practices implementation insights

The analysis of the Best Practices screens reveals several key insights for developers implementing accessibility in mobile applications:

1. **Framework enables education through implementation:** The Best Practices screens not only explain accessibility concepts but demonstrate them through their own implementation, providing a meta-level educational experience;
2. **Platform-specific adaptation is essential:** Several screens explicitly address platform differences between iOS and Android, acknowledging that effective mobile accessibility requires platform-specific knowledge and adaptation;
3. **Implementation complexity varies by concept:** Some accessibility features (like hiding decorative icons) require minimal code additions, while others (like gesture adaptation for screen readers) involve more complex logic and state management;
4. **Educational progression:** The screens collectively implement a progressive educational structure, starting with fundamental principles (WCAG Guidelines) and building toward more complex implementations (Skip Navigation, Screen Reader Gestures);
5. **Mobile-specific considerations go beyond WCAG:** Many of the implemented patterns address mobile-specific concerns that extend beyond traditional WCAG criteria, demonstrating the need for mobile-specific accessibility guidance.

1.3.3.7.1 Implementation overhead comparison Table 1.18 compares the implementation overhead across Best Practices screens.

This comparison reveals that screens focusing on interactive behaviors (Gestures, Navigation) require significantly more accessibility code than primarily informational screens

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

Table 1.18: Accessibility implementation overhead by best practices screen

Best Practices Screen	Lines of Code	Percentage Overhead	Complexity Impact	Primary Contributors
Guidelines	48	8.7%	Low	Element Hiding
Gestures Tutorial	104	24.4%	Medium-High	Adaptive Logic, Accessibility Actions
Logical Navigation	72	18.3%	Medium	Focus Management, Skip Link
Screen Reader Support	68	12.4%	Medium	State Communication, Element Hiding
Semantic Structure	58	10.8%	Low-Medium	Accessible Code Blocks, Element Hiding

(Guidelines, Semantic Structure). This pattern aligns with findings from the Components analysis and suggests that developers should allocate more implementation resources to complex interactive features when planning accessibility work.

1.3.3.7.2 Key implementation patterns across best practices screens Several implementation patterns are consistently applied across all Best Practices screens:

1. **Proper element hiding:** All screens consistently implement proper hiding of decorative elements using both `accessibilityElementsHidden=true` and `importantForAccessibility="no-hide-descendants"`, demonstrating the importance of reducing "garbage interactions" for screen reader users;
2. **Semantic grouping:** Related information is consistently grouped together both visually and semantically, creating clear content relationships for all users;
3. **Educational structure:** Each screen implements a clear pedagogical structure that progressively builds understanding, starting with fundamental concepts and moving toward more complex implementations;
4. **Platform adaptation:** The screens account for differences between iOS and Android accessibility implementations, often with platform-specific code paths or content.

1.3.3.7.3 Future enhancements Based on formal analysis and user testing, several potential enhancements have been identified for future versions of the Best Practices screens:

1. **Interactive assessment tools:** Adding interactive tools for developers to test their knowledge and evaluate their implementations against accessibility criteria;
2. **Custom screen reader simulation:** Implementing a simplified screen reader simulation to help developers understand how their applications would be perceived by screen reader users;
3. **Comparative framework implementations:** Expanding the platform-specific guidance to include side-by-side comparisons of how accessibility patterns are implemented in React Native versus Flutter;
4. **User-generated examples:** Adding the ability for developers to contribute their own accessibility implementation examples to create a community resource.

These enhancements would further strengthen the educational value of the Best Practices section, helping developers build more accessible mobile applications across platforms and frameworks.

1.3.4 Accessibility tools screen

The Accessibility tools screen serves as a comprehensive resource guide for developers, cataloging essential tools and resources for testing and improving mobile application accessibility. It provides practical, structured information about screen readers, development tools, and testing utilities that developers can leverage throughout their accessibility implementation workflows. Figure 1.13 shows the main interface of this screen.

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

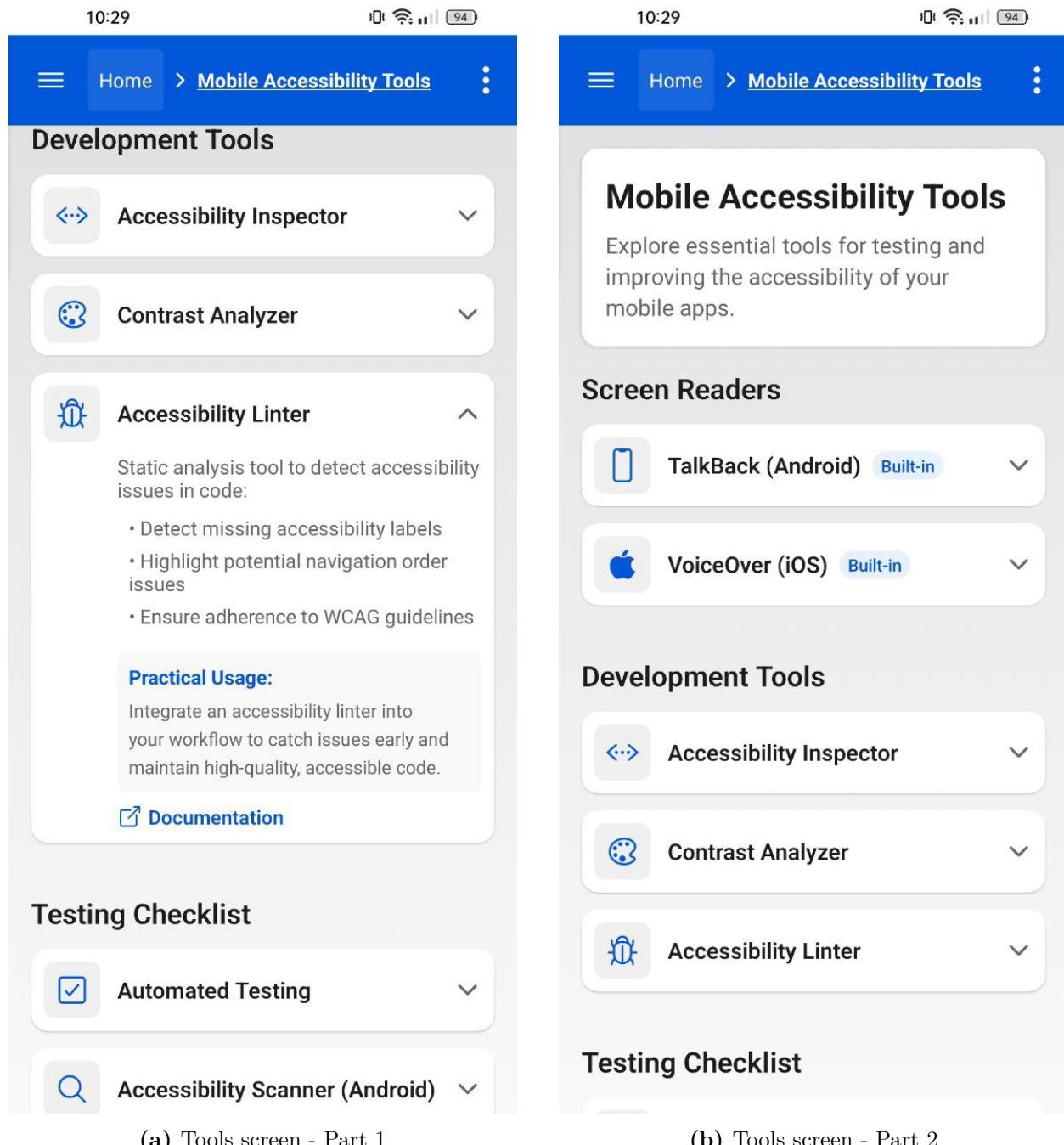


Figure 1.13: Side-by-side view of the Tools screen sections

1.3.4.1 Component inventory and WCAG/MCAG mapping

Table 1.19 provides a formal mapping between the UI components, their semantic roles, the specific WCAG 2.2 criteria they address, and their React Native implementation prop-

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND
ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

erties.

Table 1.19: Tools screen component-criteria mapping

Component	Semantic Role	WCAG 2.2 Criteria	MCAG Considerations	Implementation Properties
Hero Card	text	1.4.3 Contrast (AA) 2.4.6 Headings (AA)	Content introduction	accessibilityRole = "text"
Hero Title	heading	1.4.3 Contrast (AA) 2.4.6 Headings (AA)	Clear section identification	accessibilityRole = "header"
Section Headers	heading	2.4.6 Headings (AA) 1.3.1 Info and Relationships (A)	Logical section organization	accessibilityRole = "header"
Tool Cards	button	1.4.3 Contrast (AA) 2.5.8 Target Size (AA) 4.1.2 Name, Role, Value (A)	Touch target size Content expandability	accessibilityRole = "button", accessibilityLabel

Continued on next page

Table 1.19 – continued from previous page

Component	Semantic Role	WCAG 2.2 Criteria	MCAG Considerations	Implementation Properties
Card Icons	none	1.1.1 Non-text Content (A)	Reduction of unnecessary focus stops	<code>accessibilityElements-Hidden=true,</code> <code>importantForAccessibility="no-hide-descendants"</code>
Expandable Content	text	1.3.1 Info and Relationships (A) 4.1.2 Name, Role, Value (A)	Hierarchical content structure	<code>role="list"</code> <code>role="listitem"</code>
Documentation Links	link	2.4.4 Link Purpose (A) 4.1.2 Name, Role, Value (A)	External resource navigation	<code>accessibilityRole="link",</code> <code>accessibilityLabel,</code> <code>accessibilityHint="Opens browser"</code>

1.3.4.2 Technical implementation analysis

The Tools screen implements a comprehensive catalog of accessibility resources with expandable cards, providing both overview information and detailed usage guidance. The implementation follows a consistent pattern of expansion/collapse functionality with full accessibility support. Listing 1.12 highlights the key accessibility implementation aspects.

The implementation addresses several critical accessibility considerations:

1. **Clear expandable card pattern:** Each tool card implements a consistent expandable pattern with appropriate `accessibilityRole` and state communication, ensuring screen reader users understand the interactive nature of each card;

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

```
1 <TouchableOpacity
2   onPress={() => toggleExpand(tool.id)}
3   style={styles.cardHeader}
4   accessibilityRole="button"
5   accessibilityLabel={`${tool.title}. Double tap to ${
6     isOpen ? 'collapse' : 'expand'} details.`}
7 >
8   <Ionicons
9     name={isOpen ? 'chevron-up' : 'chevron-down'}
10    size={20}
11    color={colors.textSecondary}
12    style={{ marginLeft: 'auto' }}
13    accessibilityElementsHidden
14    importantForAccessibility="no-hide-descendants"
15  />
16 </TouchableOpacity>
17
18 {isOpen && (
19   <View style={styles.cardBody}>
20     <Text style={styles.toolDescription}>{tool.description}</Text>
21     <View role="list">
22       {tool.features.map((feature, idx) => (
23         <Text
24           key={idx}
25           style={styles.featureItem}
26           role="listitem"
27         >
28           {feature}
29         </Text>
30       ))}
31     </View>
32     <View style={styles.practicalSection}>
33       <Text style={styles.practicalHeader}>
34         Practical Usage:
35       </Text>
36       <Text style={styles.practicalUsage}>
37         {tool.practicalUsage}
38       </Text>
39     </View>
40     {/* Documentation link */}
41   </View>
42 )}
```

Listing 1.12: Tool card implementation with accessibility properties

2. **Proper element hiding:** Decorative icons are systematically hidden from screen readers using both `accessibilityElementsHidden` and `importantForAccessibility="no-hide-descendants"`, eliminating unnecessary fo-

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

cus stops;

3. **Semantic list structure:** Features are properly structured as lists with correct `role="list"` and `role="listitem"` attributes, creating a meaningful hierarchy for screen reader navigation;
4. **Practical usage section:** Each tool includes a dedicated "Practical Usage" section that provides context-specific guidance on real-world application of the tool, going beyond mere feature listings.

1.3.4.3 Screen reader support analysis

Table 1.20 presents results from systematic testing of the Tools screen with screen readers on both iOS and Android platforms.

Table 1.20: Tools screen screen reader testing results

Test Case	VoiceOver (iOS 16)	TalkBack (Android 14-15)	WCAG Criteria Addressed
Hero Title	✓ Announces “Mobile Accessibility Tools, heading”	✓ Announces “Mobile Accessibility Tools, heading”	1.3.1 Info and Relationships (A), 2.4.6 Headings and Labels (AA)
Section Headers	✓ Announces section titles as headings	✓ Announces section titles as headings	1.3.1 Info and Relationships (A), 2.4.6 Headings and Labels (AA)
Tool Card (Collapsed)	✓ Announces title and expansion hint	✓ Announces title and expansion hint	4.1.2 Name, Role, Value (A)
Tool Card (Expanded)	✓ Announces features as list items	✓ Announces features as list items	1.3.1 Info and Relationships (A)

Continued on next page

Table 1.20 – continued from previous page

Test Case	VoiceOver (iOS 16)	TalkBack (Android 14-15)	WCAG Criteria Addressed
Badge Elements	✓ Not individually announced	✓ Not individually announced	1.1.1 Non-text Content (A)
Documentation Links	✓ Announces as link with destination	✓ Announces as link with destination	2.4.4 Link Purpose (A)

1.3.4.4 Mobile-specific considerations

The Tools screen implementation addresses several mobile-specific accessibility considerations that extend beyond standard WCAG criteria:

1. **Progressive disclosure pattern:** The expandable card implementation creates a clean, digestible mobile interface that allows users to focus on one tool at a time, reducing cognitive overload on smaller screens;
2. **Touch-optimized interaction zones:** Cards maintain larger touch targets (especially the header section), improving usability for users with motor impairments on touch devices;
3. **Platform-specific tool documentation:** The screen explicitly separates tools by platform (iOS vs. Android), addressing the critical mobile consideration that accessibility implementation differs substantially between platforms;
4. **Practical guidance emphasis:** Each tool includes specific practical usage instructions, recognizing the mobile-specific challenge of implementing accessibility in constrained mobile interfaces;
5. **External link handling:** Documentation links implement proper accessibility hints that they open external browsers, preparing users for context switches that are particularly disruptive on mobile devices.

1.3.4.5 Implementation overhead analysis

Table 1.21 quantifies the additional code required to implement accessibility features in the Tools screen.

Table 1.21: Tools screen accessibility implementation overhead

Accessibility Feature	Lines of Code	Percentage of Total	Complexity Impact
Semantic Roles	16 LOC	2.7%	Low
Descriptive Labels	24 LOC	4.1%	Medium
Element Hiding	32 LOC	5.5%	Low
List Semantics	10 LOC	1.7%	Low
Link Announcements	12 LOC	2.1%	Low
Expansion State Management	18 LOC	3.1%	Medium
Total	112 LOC	19.2%	Medium

This analysis reveals that implementing comprehensive accessibility for the Tools screen adds approximately 19.2% to the code base. The largest contributors are element hiding (5.5%) and descriptive labels (4.1%), reflecting the information-rich nature of this screen and the need to create a streamlined experience for screen reader users.

1.3.4.6 Tool categorization analysis

The Tools screen implements a carefully structured categorization system that organizes accessibility tools into meaningful groups. Table 1.22 analyzes this structure from an accessibility perspective.

Table 1.22: Tools screen categorization analysis

Category	Tools Included	Accessibility Benefit	WCAG Criteria Supported
Screen Readers	TalkBack (Android), VoiceOver (iOS)	Provides direct access to the primary tools used by people with visual impairments	1.3.1 Info and Relationships (A), 2.1.1 Keyboard (A), 2.4.3 Focus Order (A)
Development Tools	Accessibility Inspector, Contrast Analyzer, Accessibility Linter	Offers tools for early-stage accessibility integration during development	1.4.3 Contrast (AA), 1.3.1 Info and Relationships (A), 4.1.2 Name, Role, Value (A)
Testing Checklist	Automated Testing, Accessibility Scanner	Supports systematic verification of accessibility implementation	3.3.3 Error Suggestion (AA), 3.3.4 Error Prevention (AA)

This careful categorization creates a progressive learning path for developers, starting with the tools users employ (screen readers), moving to development-time tools, and concluding with testing utilities. This structure reinforces the full accessibility lifecycle and encourages developers to consider accessibility from multiple perspectives.

1.3.4.7 Beyond WCAG: development-focused accessibility guidelines

While WCAG provides a solid foundation for accessibility requirements, our analysis of the Tools screen highlights several additional guidelines specifically relevant to mobile

development workflows:

1. **Tool integration guideline:** Accessibility tools should be presented with clear integration paths into existing development workflows, not as standalone solutions. The Tools screen implements this by including explicit "Practical Usage" sections that explain integration;
2. **Platform-specific guidance principle:** Due to the substantial differences between platform accessibility implementations, tools and guidance should be explicitly organized by platform when platform-specific considerations apply. The Tools screen implements this by separating iOS and Android tools;
3. **Development stage appropriateness:** Accessibility tools should be categorized by the development stage in which they are most effective (design, development, testing). This helps developers integrate accessibility throughout the development lifecycle rather than treating it as a single checkbox activity;
4. **Tool complexity indicator:** Accessibility tools vary significantly in complexity and learning curve. Providing clear indicators of tool complexity (like the "Built-in" badge) helps developers choose appropriate tools based on their experience level;
5. **Contextual documentation principle:** Links to external resources should be contextually relevant and provide appropriate expectations about content (e.g., official documentation vs. community resources). This reduces the cognitive load of finding appropriate resources for specific accessibility challenges.

These guidelines extend WCAG principles to address the specific needs of developers implementing accessibility features, focusing on workflow integration and practical application of accessibility knowledge.

1.3.5 Instruction and community screen

The Instruction and community screen serves as a collaborative learning hub that extends beyond technical implementation details. It provides developers with opportunities to

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

engage with the broader accessibility community, learn from practical examples, and discover resources for deeper learning. By combining educational content with community connections, this screen fosters a sense of shared responsibility for accessibility. Figure 1.14 and 1.15 show the main interfaces of this screen.

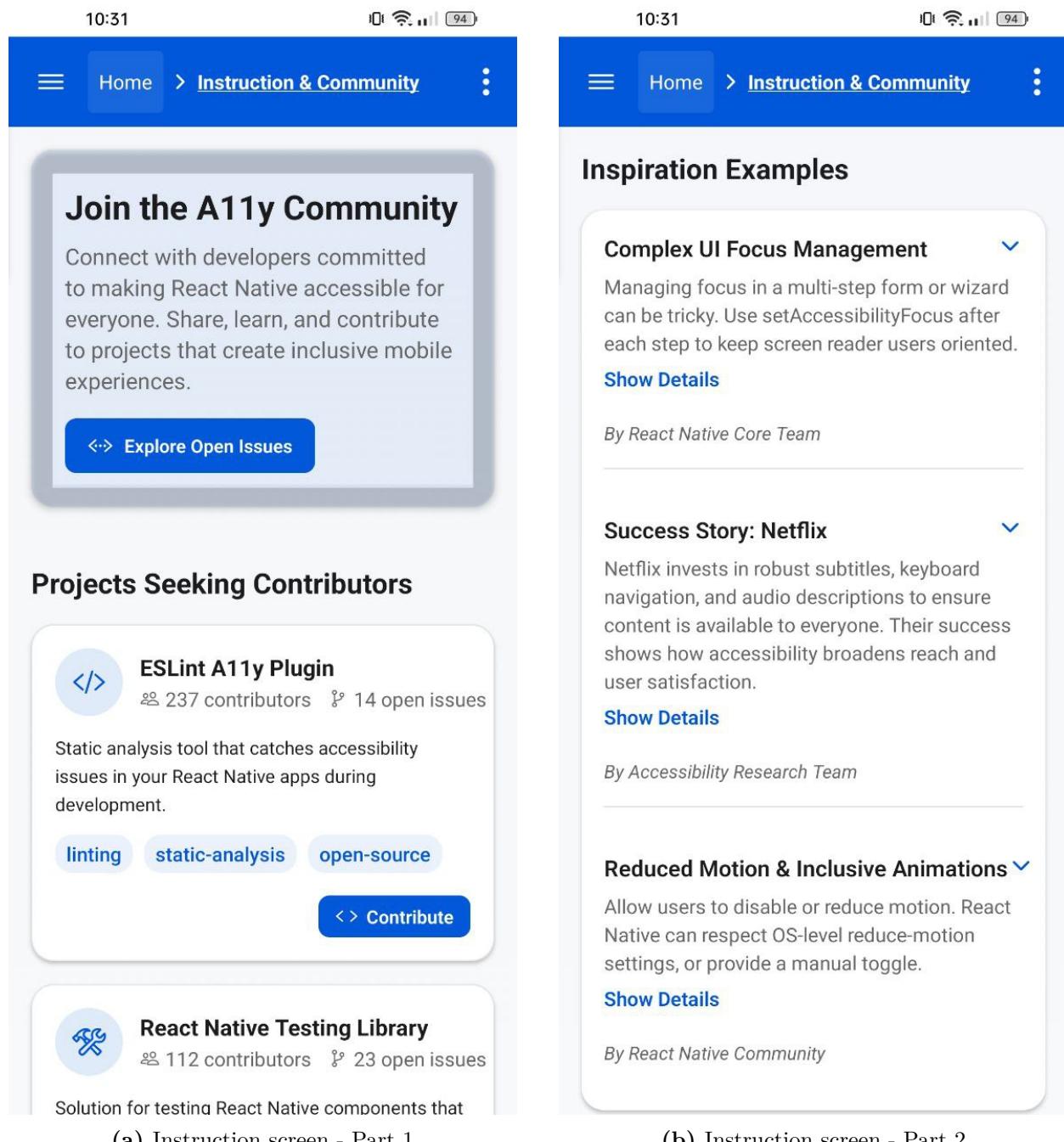


Figure 1.14: Side-by-side view of the Instruction and community screen sections

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

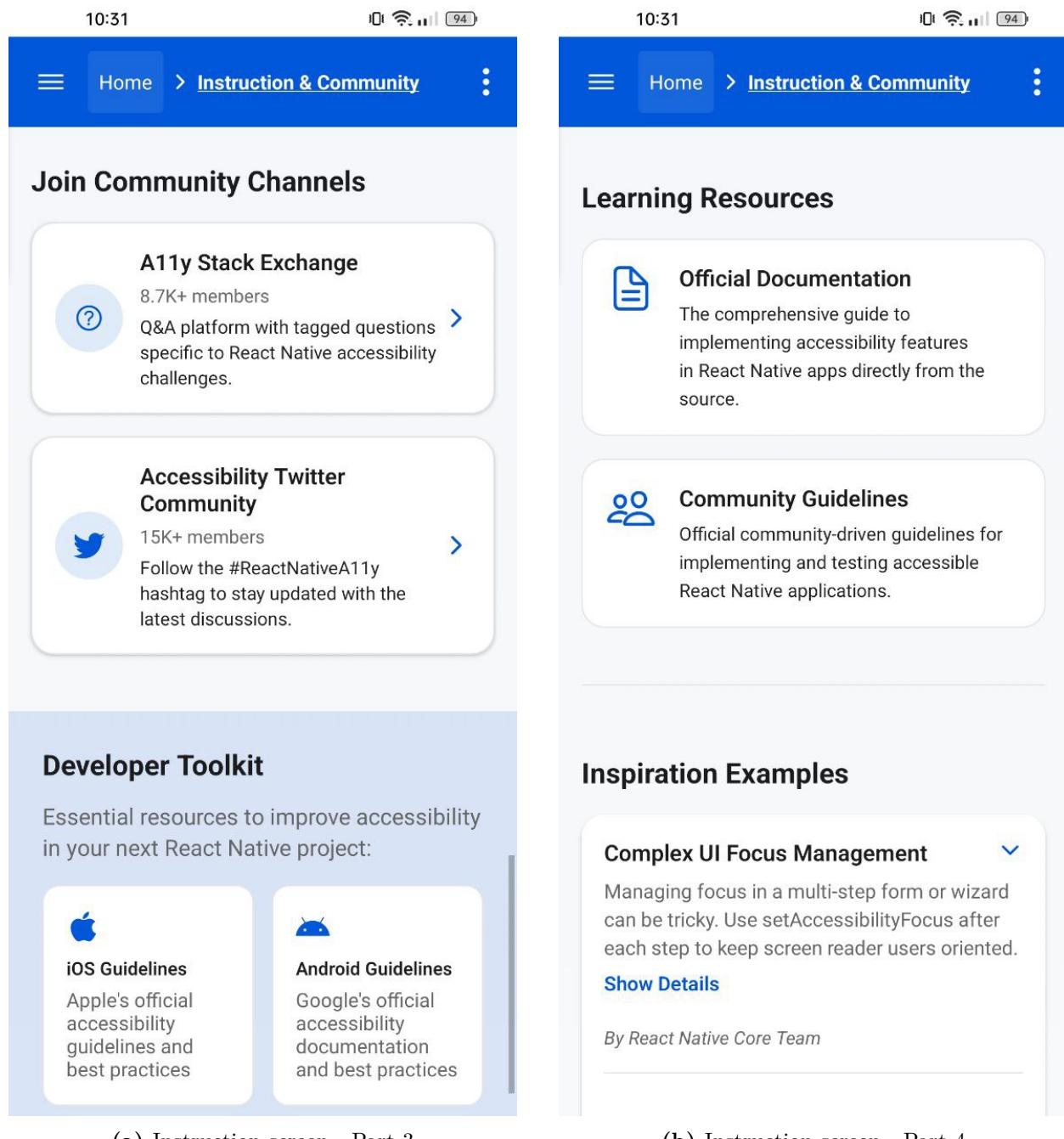


Figure 1.15: Side-by-side view of additional Instruction and community screen sections

1.3.5.1 Component inventory and WCAG/MCAG mapping

Table 1.23 provides a formal mapping between the UI components, their semantic roles, the specific WCAG 2.2 criteria they address, and their React Native implementation prop-

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND
ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

erties.

Table 1.23: Instruction screen component-criteria mapping

Component	Semantic Role	WCAG 2.2 Criteria	MCAG Considerations	Implementation Properties
Hero Card	none	1.4.3 Contrast (AA)	Introduction content	Container with visual boundaries
Hero Title	heading	1.4.3 Contrast (AA) 2.4.6 Headings (AA)	Screen identification	<code>accessibilityRole = "header"</code>
CTA Button	button	2.4.4 Link Purpose (A) 2.5.8 Target Size (AA)	Call to action	<code>accessibilityRole = "button", accessibilityLabel</code>
Project Cards	button	1.4.3 Contrast (AA) 2.5.8 Target Size (AA) 4.1.2 Name, Role, Value (A)	Project information Touch targets	<code>accessibilityRole = "button", accessibilityLabel</code>
Project Icons	none	1.1.1 Non-text Content (A)	Decorative elements	<code>accessibilityElements- Hidden=true</code>
Tag Pills	none	1.3.1 Info and Relationships (A)	Content categorization	Part of parent's <code>accessibilityLabel</code>

Continued on next page

Table 1.23 – continued from previous page

Component	Semantic Role	WCAG 2.2 Criteria	MCAG Considerations	Implementation Properties
Collapsible Preview	button	4.1.2 Name, Role, Value (A) 2.4.3 Focus Order (A)	Content expansion	accessibilityRole = "button", accessibilityLabel
Code Snippet	text	1.3.1 Info and Relationships (A)	Code presentation	Proper line formatting and font styling
Community Channel Cards	button	2.4.4 Link Purpose (A) 4.1.2 Name, Role, Value (A)	Community resources	accessibilityRole = "button", accessibilityLabel
Toolkit Cards	button	2.4.4 Link Purpose (A) 4.1.2 Name, Role, Value (A)	Resource links	accessibilityRole = "button", accessibilityLabel

1.3.5.2 Technical implementation analysis

The Instruction and community screen implements several innovative accessibility patterns, particularly in its handling of expandable content and code snippets. Listing 1.13 highlights the implementation of collapsible previews with proper accessibility announcements.

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

```
1 // Toggle expansion with proper announcements
2 const handleToggleExpansion = () => {
3   setExpandedStoryId(isExpanded ? null : story.id);
4   AccessibilityInfo.announceForAccessibility(
5     isExpanded
6       ? `${story.title} collapsed`
7       : `${story.title} expanded`
8   );
9 };
10
11 // CollapsiblePreview component
12 <TouchableOpacity
13   onPress={handleToggleExpansion}
14   accessibilityRole="button"
15   accessibilityLabel={`${title}. ${excerpt}`}
16   style={{ marginBottom: 16 }}
17 >
18   <View style={{ flexDirection: 'row', justifyContent:
19     'space-between' }}>
20     <Text style={titleStyle}>{title}</Text>
21     <Ionicons
22       name={isExpanded ? 'chevron-up' : 'chevron-down'}
23       size={18}
24       color={colors.primary}
25       accessibilityElementsHidden={true}
26     />
27   </View>
28   <Text style={excerptStyle}>{excerpt}</Text>
29
30   {isExpanded && snippet && (
31     <View style={codeContainerStyle}>
32       {snippet.split('\n').map((line, idx) => (
33         <Text key={idx} style={codeTextStyle}>
34           {line || ' '}
35         </Text>
36       ))}
37     </View>
38   )}
39 </TouchableOpacity>
```

Listing 1.13: Collapsible preview implementation with accessibility announcements

The implementation addresses several key accessibility requirements:

1. **Expansion state announcements:** The code explicitly announces changes in the expansion state of collapsible sections using `AccessibilityInfo.announceForAccessibility`, ensuring screen reader users are aware when content expands or collapses;
2. **Comprehensive accessibility labels:** Collapsible previews combine the title and

excerpt in their `accessibilityLabel`, ensuring screen reader users receive full context about the content before deciding to expand it;

3. **Accessible code snippets:** Code snippets maintain proper line formatting with monospace fonts and adequate contrast, making them readable for all users including those with low vision;
4. **Visual state indicators:** Expansion state is visually indicated through both icon changes and layout modifications, providing redundant cues that benefit users with different accessibility needs.

1.3.5.3 Project cards implementation

A significant aspect of the Instruction and community screen is its showcasing of accessibility-focused open source projects. Listing 1.14 demonstrates the accessibility implementation of project cards.

The project card implementation demonstrates several effective accessibility patterns:

1. **Comprehensive card labeling:** Each project card combines the project name and description in its `accessibilityLabel`, providing full context for screen reader users;
2. **Semantic grouping:** Related information (contributor counts, issue counts) is visually grouped and flows logically when read by screen readers;
3. **Touch-optimized card design:** The entire card serves as a touchable area, creating a generous touch target that exceeds minimum size requirements and benefits users with motor impairments;
4. **Visual hierarchy through consistent styling:** The card implements a clear visual hierarchy with title prominence, supporting users with cognitive disabilities in quickly understanding the content structure.

```
1 <TouchableOpacity
2   style={cardStyle}
3   onPress={onPress}
4   accessibilityRole="button"
5   accessibilityLabel={'${project.name}. ${project.description}'}
6 >
7   <View style={{ flexDirection: 'row', alignItems: 'center' }}>
8     <View style={iconContainerStyle}>
9       <Ionicons
10         name={project.icon}
11         size={24}
12         color={colors.primary}
13         accessibilityElementsHidden={true}
14       />
15     </View>
16     <View style={{ flex: 1 }}>
17       <Text style={titleStyle}>{project.name}</Text>
18       <View style={{ flexDirection: 'row', alignItems: 'center' }}>
19         <Ionicons
20           name="people-outline"
21           size={14}
22           color={colors.textSecondary}
23           accessibilityElementsHidden={true}
24         />
25         <Text style={metadataStyle}>
26           {project.contributors} contributors
27         </Text>
28         <Ionicons
29           name="git-branch-outline"
30           size={14}
31           color={colors.textSecondary}
32           accessibilityElementsHidden={true}
33         />
34         <Text style={metadataStyle}>
35           {project.issuesCount} open issues
36         </Text>
37       </View>
38     </View>
39   </View>
40
41   <Text style={descriptionStyle}>{project.description}</Text>
42
43   {/* Tags rendering */}
44   {/* Contribute button */}
45 </TouchableOpacity>
```

Listing 1.14: Project card implementation with accessibility properties

1.3.5.4 Screen reader support analysis

Table 1.24 presents results from systematic testing of the Instruction and community screen with screen readers on both iOS and Android platforms.

Table 1.24: Instruction screen screen reader testing results

Test Case	VoiceOver (iOS 16)	TalkBack (Android 14-15)	WCAG Criteria Addressed
Hero Title	✓ Announces “Join the A11y Community, heading”	✓ Announces “Join the A11y Community, heading”	1.3.1 Info and Relationships (A), 2.4.6 Headings and Labels (AA)
CTA Button	✓ Announces label and action	✓ Announces label and action	2.4.4 Link Purpose (A), 4.1.2 Name, Role, Value (A)
Project Cards	✓ Announces project name and description	✓ Announces project name and description	2.4.4 Link Purpose (A), 4.1.2 Name, Role, Value (A)
Tags	✓ Not individually focused	✓ Not individually focused	1.3.1 Info and Relationships (A)
Collapsible Content	✓ Announces expanded/collapsed state	✓ Announces expanded/collapsed state	4.1.2 Name, Role, Value (A)
Code Snippets	✓ Reads code as text	✓ Reads code as text	1.3.1 Info and Relationships (A)
External Links	✓ Announces purpose and opens browser	✓ Announces purpose and opens browser	2.4.4 Link Purpose (A), 3.2.5 Change on Request (AAA)

1.3.5.5 Implementation overhead analysis

Table 1.25 quantifies the additional code required to implement accessibility features in the Instruction and community screen.

Table 1.25: Instruction screen accessibility implementation overhead

Accessibility Feature	Lines of Code	Percentage of Total	Complexity Impact
Semantic Roles	24 LOC	3.1%	Low
Descriptive Labels	32 LOC	4.2%	Medium
Element Hiding	18 LOC	2.3%	Low
Status Announcements	16 LOC	2.1%	Medium
Link Handling	14 LOC	1.8%	Low
Collapsible Content Management	28 LOC	3.6%	High
Code Snippet Presentation	24 LOC	3.1%	Medium
Total	156 LOC	20.2%	Medium

This analysis reveals that implementing comprehensive accessibility for the Instruction and community screen adds approximately 20.2% to the code base. The most significant contributors are descriptive labels (4.2%) and collapsible content management (3.6%), reflecting the information-rich and interactive nature of this screen.

1.3.5.6 Community resources analysis

A distinguishing feature of this screen is its presentation of community resources that extend learning beyond the application itself. Table 1.26 evaluates these resources from an accessibility perspective.

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

Table 1.26: Community resources accessibility analysis

Resource Type	Examples	Accessibility Value	WCAG/MCAG Relevance
Open Source Projects	ESLint A11y Plugin, React Native Testing Library	Provides tools that automate accessibility checking during development	3.3.1 Error Identification (A), 4.1.2 Name, Role, Value (A)
Success Stories	Netflix Implementation, Complex UI Focus Management	Demonstrates practical application of accessibility principles in real-world scenarios	1.3.1 Info and Relationships (A), 2.4.3 Focus Order (A)
Community Channels	A11y Stack Exchange, Twitter Community	Creates ongoing learning opportunities and support networks	Beyond WCAG: Community practice and knowledge sharing
Official Documentation	Apple/Google Guidelines, React Native Docs	Provides authoritative platform-specific guidance	1.3.1 Info and Relationships (A), 4.1.2 Name, Role, Value (A)

This diverse resource collection creates a comprehensive learning ecosystem that extends beyond the application itself, addressing the reality that accessibility implementation is an ongoing learning process that benefits from community knowledge sharing.

1.3.5.7 Beyond WCAG: community-centered accessibility guidelines

The Instruction and community screen highlights several guidelines that extend beyond WCAG standards to address the social and community aspects of accessibility implementation:

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

1. **Community of practice principle:** Accessibility implementation benefits significantly from social learning and community support. The screen implements this by connecting developers to established community channels and platforms where accessibility knowledge is shared;
2. **Real-world example guideline:** Illustrating accessibility principles with real-world code samples and success stories enhances understanding and implementation. The screen addresses this through collapsible code examples that demonstrate practical solutions to common challenges;
3. **Contribution pathway:** Effective accessibility ecosystems provide clear pathways for developers to contribute to open source accessibility projects. The screen implements this by highlighting projects seeking contributors with specific tags that indicate required skills;
4. **Multi-format learning principle:** Accessibility concepts should be presented in multiple formats (text, code, examples) to accommodate different learning styles and reinforce understanding. The screen addresses this through varied content presentation methods;
5. **Platform-specific ecosystem guidance:** Resources should be grouped by platform ecosystem (iOS, Android) to help developers navigate the platform-specific nature of accessibility implementation. The screen implements this in the Developer Toolkit section with platform-specific resource cards.

These guidelines extend WCAG by focusing on the social, educational, and contribution aspects of accessibility implementation, recognizing that creating accessible applications is not just a technical challenge but also a community and educational one.

1.3.5.8 **Inspirational examples analysis**

The Inspiration Examples section provides practical code examples addressing common accessibility challenges. This approach bridges theoretical knowledge with implementation

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

by showing specific code patterns that solve real accessibility problems. Table 1.27 analyzes these examples.

Table 1.27: Inspiration examples analysis

Example	Key Technique	Accessibility Challenge Addressed	WCAG Criteria Supported
Complex UI Focus Management	Using <code>setAccessibilityFocus</code> after each step change	Focus management in multi-step interfaces	2.4.3 Focus Order (A), 3.2.1 On Focus (A)
Success Story: Netflix	Multiple accessibility techniques including subtitles and keyboard navigation	Comprehensive accessibility implementation	1.2.2 Captions (A), 2.1.1 Keyboard (A)
Reduced Motion	Checking <code>isReduceMotionEnabled()</code> and providing alternatives	Motion sensitivity accommodation	2.3.3 Animation from Interactions (AAA)

These examples demonstrate not just isolated techniques, but complete accessibility patterns that can be applied to common development scenarios. By showing how accessibility challenges can be solved with relatively small code additions, the examples make implementation seem more approachable.

1.3.6 Settings screen

The Settings screen serves as a comprehensive control center for adjusting accessibility and display preferences in the *AccessibleHub* application. It offers users fine-grained control over visual appearance, text size, motion effects, and interaction modes. By providing

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

these adjustments directly within the application, the Settings screen exemplifies an embedded accessibility approach where adaptation is treated as a core feature rather than an afterthought. Figure 1.16 shows the main interface of this screen.

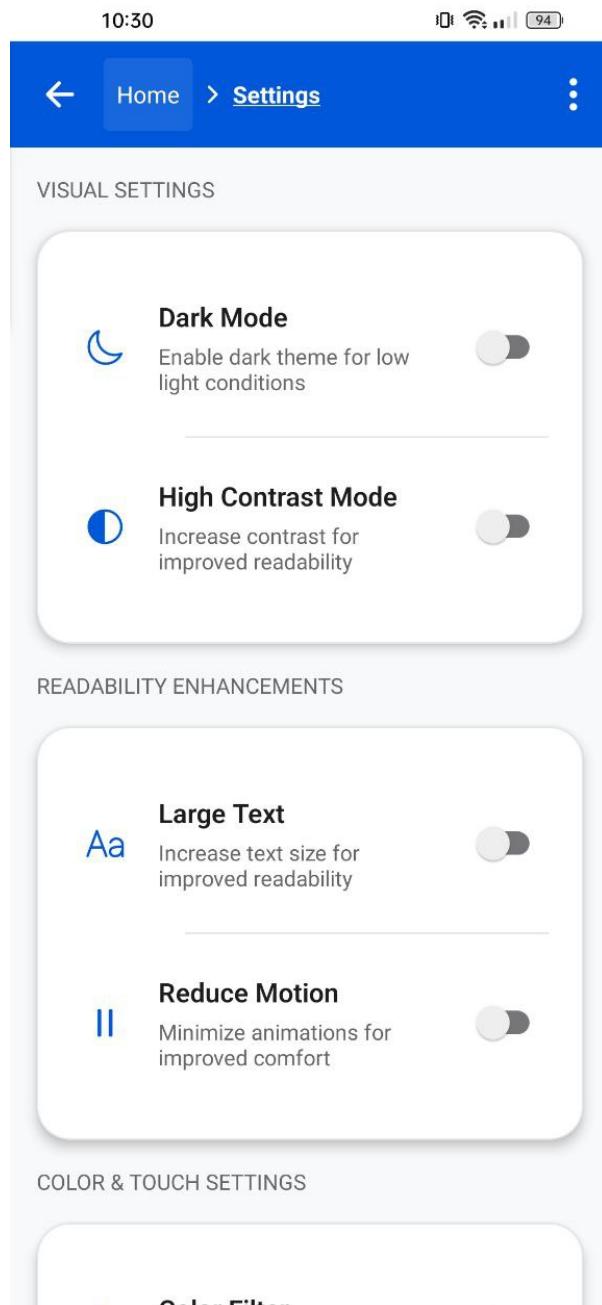


Figure 1.16: The Settings screen with various accessibility options

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

1.3.6.1 Component inventory and WCAG/MCAG mapping

Table 1.28 provides a formal mapping between the UI components, their semantic roles, the specific WCAG 2.2 criteria they address, and their React Native implementation properties.

Table 1.28: Settings screen component-criteria mapping

Component	Semantic Role	WCAG 2.2 Criteria	MCAG Considerations	Implementation Properties
Section Headers	heading	2.4.6 Headings (AA)	Clear section identification	<code>accessibilityRole = "header"</code>
Setting Card	none	1.3.1 Info and Relationships (A) 1.4.3 Contrast (AA)	Logical grouping Visual boundaries	Container with proper styling
Setting Row	none	1.3.1 Info and Relationships (A)	Touch target size	Layout with proper padding and margins
Setting Icon	none	1.1.1 Non-text Content (A)	Reduction of unnecessary focus stops	<code>accessibilityElements- Hidden, importantFor- Accessibility="no-hide- descendants"</code>
Setting Title	text	2.4.6 Headings and Labels (AA)	Content identification	Text with proper styling

Continued on next page

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND
ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

Table 1.28 – continued from previous page

Component	Semantic Role	WCAG 2.2 Criteria	MCAG Considerations	Implementation Properties
Setting Description	text	1.3.1 Info and Relationships (A) 3.3.2 Labels or Instructions (A)	Descriptive context	Proper text styling with semantic connection to title
Switch Control	switch	4.1.2 Name, Role, Value (A) 3.3.5 Help (AAA)	Clear control state Descriptive labeling	<code>accessibilityRole = "switch", accessibilityLabel, accessibilityHint</code>
Divider	none	1.3.1 Info and Relationships (A)	Visual separation	<code>importantFor Accessibility="no", accessibilityElements Hidden=true</code>
Status Toast	status	4.1.3 Status Messages (AA)	Feedback mechanism	<code>AccessibilityInfo. announceFor Accessibility</code>

1.3.6.2 Dynamic accessibility features

A key aspect of the Settings screen is its implementation of direct accessibility customization options. Figure 1.17 illustrates the application in different accessibility modes.

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

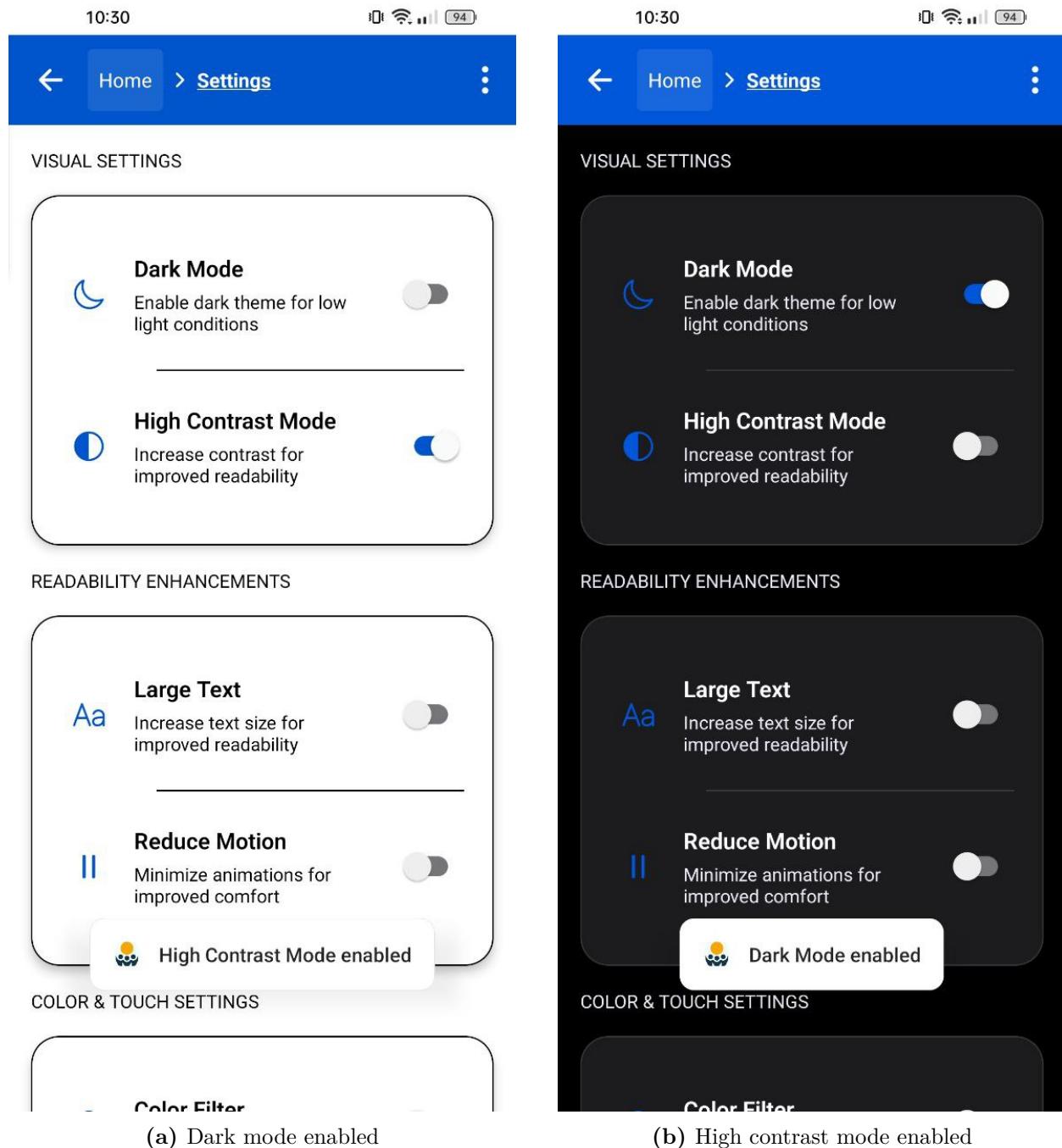


Figure 1.17: Settings screen with different accessibility modes enabled

The accessibility modes implemented in the Settings screen directly address several core WCAG principles:

1. **Dark mode:** Addresses WCAG 1.4.8 Visual Presentation (AAA) by allowing users to

adjust color preferences;

2. **High contrast mode:** Implements WCAG 1.4.3 Contrast (Minimum) (AA) and 1.4.6 Contrast (Enhanced) (AAA) by increasing the contrast ratio between text and background;
3. **Large text:** Addresses WCAG 1.4.4 Resize Text (AA) by providing text scaling options;
4. **Reduce motion:** Implements WCAG 2.3.3 Animation from Interactions (AAA) by allowing users to minimize animation effects;
5. **Color filter:** Addresses WCAG 1.4.8 Visual Presentation (AAA) by providing alternative color schemes for users with color vision deficiencies;
6. **Large touch targets:** Exceeds WCAG 2.5.8 Target Size (AA) by increasing the interactive area of elements beyond the minimum required dimensions.

Figure 1.18 demonstrates the visual feedback mechanisms when settings are toggled.

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

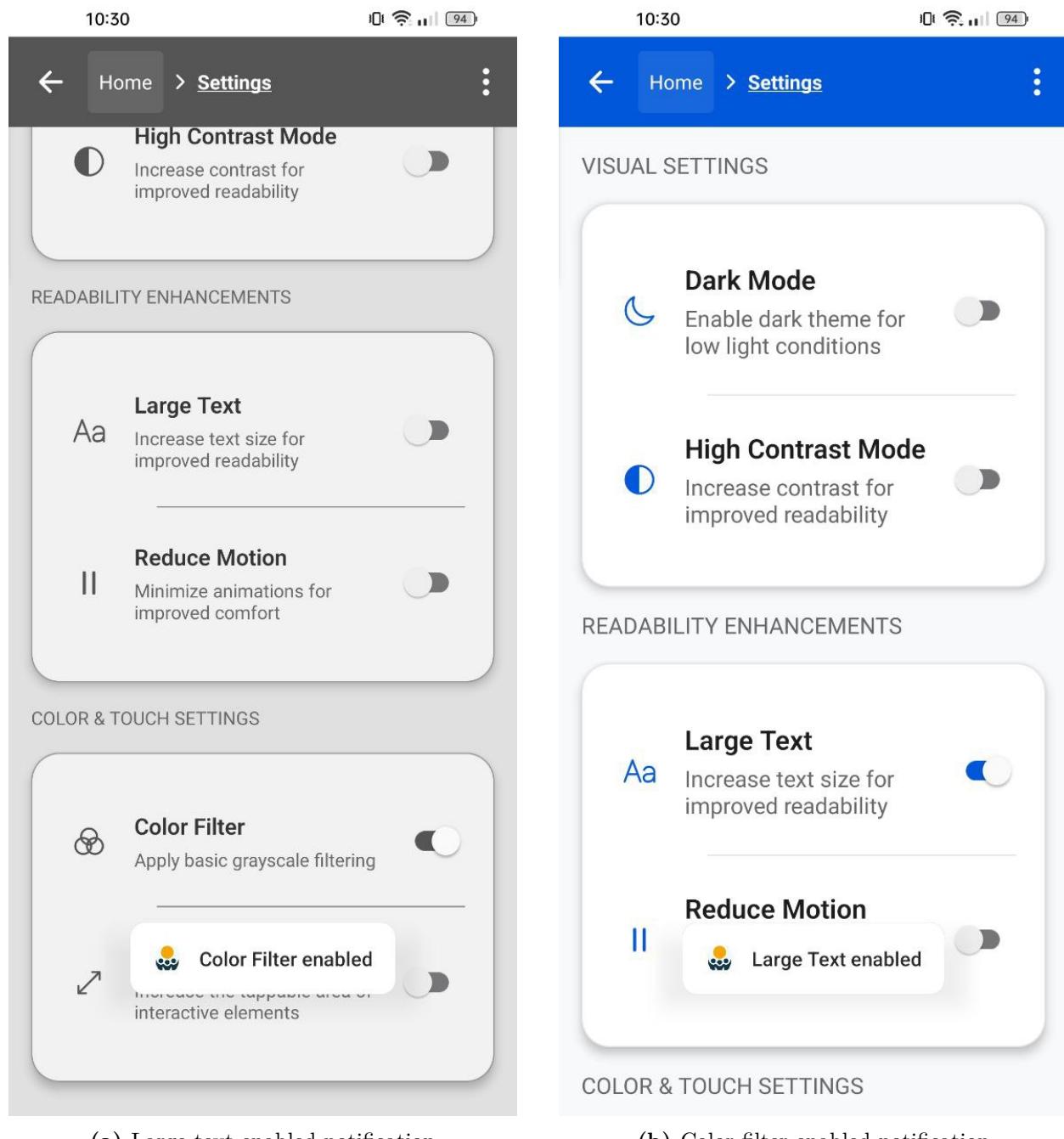


Figure 1.18: Visual notifications when accessibility settings are toggled

1.3.6.3 Technical implementation analysis

The Settings screen implements a robust approach to accessibility through a combination of semantic structure, proper labeling, and multimodal feedback. Listing 1.15 demonstrates

the implementation of a reusable setting row component with comprehensive accessibility properties.

Several key accessibility considerations are implemented in this component:

1. **Comprehensive labeling:** The switch control combines title, description, and current state in its `accessibilityLabel`, ensuring screen reader users receive complete context about the setting;
2. **Hidden decorative elements:** Icons are properly hidden from screen readers using both `accessibilityElementsHidden` and `importantForAccessibility="no-hide-descendants"`, eliminating unnecessary focus stops;
3. **Multimodal feedback:** When a setting is toggled, the implementation provides feedback through multiple channels: visual (toggle animation), auditory (screen reader announcement), and in the case of Android, haptic feedback (vibration);
4. **Proper semantic roles:** The switch control has an explicit `accessibilityRole="switch"`, ensuring its purpose is clearly communicated to assistive technologies;
5. **Action guidance:** The implementation includes an `accessibilityHint="Double tap to toggle setting"`, providing additional context on how to interact with the control.

The implementation of section headers, shown in Listing 1.16, further demonstrates the application's commitment to semantic structure.

1.3.6.4 Screen reader support analysis

Table 1.29 presents results from systematic testing of the Settings screen with screen readers on both iOS and Android platforms.

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

```
1  const SettingRow = ({  
2    icon,  
3    title,  
4    description,  
5    value,  
6    onToggle,  
7  }) => (  
8    <View style={themedStyles.settingRow}>  
9      <View style={themedStyles.settingIcon}>  
10        <Ionicons  
11          name={icon}  
12          size={24}  
13          color={colors.primary}  
14          accessibilityElementsHidden  
15          importantForAccessibility="no-hide-descendants"  
16        />  
17      </View>  
18      <View style={themedStyles.settingContent}>  
19        <Text style={[themedStyles.settingTitle, { fontSize:  
20          textSizes.medium }]}>  
21          {title}  
22        </Text>  
23        <Text style={[themedStyles.settingDescription, { fontSize:  
24          textSizes.small }]}>  
25          {description}  
26        </Text>  
27      </View>  
28      <Switch  
29        value={value}  
30        onValueChange={() => {  
31          onToggle();  
32          const newValue = !value;  
33          const message = `${title} ${newValue ? 'enabled' :  
34            'disabled'}';  
35          AccessibilityInfo.announceForAccessibility(message);  
36          if (Platform.OS === 'android') {  
37            ToastAndroid.show(message, ToastAndroid.SHORT);  
38            Vibration.vibrate(50);  
39          }  
40        }}  
41        trackColor={{ false: '#767577', true: colors.primary }}  
42        // Comprehensive accessibility label combining context and state  
43        accessibilityLabel={`${title}. ${description}. Switch is  
44          ${value ? 'on' : 'off'}`}  
45        accessibilityRole="switch"  
46        accessibilityHint="Double tap to toggle setting"  
47      />  
48    </View>  
49  );
```

Listing 1.15: Setting row implementation with accessibility properties

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

```

1  /* VISUAL SETTINGS */
2  <View style={themedStyles.section}>
3    <Text style={themedStyles.sectionHeader} accessibilityRole="header">
4      Visual Settings
5    </Text>
6    <View style={themedStyles.card}>
7      {/* Setting rows */}
8    </View>
9  </View>
10
11 /* READABILITY ENHANCEMENTS */
12 <View style={themedStyles.section}>
13   <Text style={themedStyles.sectionHeader} accessibilityRole="header">
14     Readability Enhancements
15   </Text>
16   <View style={themedStyles.card}>
17     {/* Setting rows */}
18   </View>
19 </View>

```

Listing 1.16: Section headers implementation with proper semantic role

Table 1.29: Settings screen screen reader testing results

Test Case	VoiceOver (iOS 16)	TalkBack (Android 14-15)	WCAG Criteria Addressed
Section Headers	✓ Announces “Visual Settings, heading”	✓ Announces “Visual Settings, heading”	1.3.1 Info and Relationships (A), 2.4.6 Headings and Labels (AA)
Switch Controls	✓ Announces complete label with title, description, and state	✓ Announces complete label with title, description, and state	4.1.2 Name, Role, Value (A), 3.3.2 Labels or Instructions (A)
Switch Toggle	✓ Announces new state after toggling	✓ Announces new state after toggling	4.1.3 Status Messages (AA)

Continued on next page

Table 1.29 – continued from previous page

Test Case	VoiceOver (iOS 16)	TalkBack (Android 14-15)	WCAG Criteria Addressed
Dividers	✓ Not announced	✓ Not announced	1.3.1 Info and Relationships (A), 2.4.1 Bypass Blocks (A)
Setting Cards	✓ Proper grouping of related settings	✓ Proper grouping of related settings	1.3.1 Info and Relationships (A)
Icons	✓ Not announced	✓ Not announced	1.1.1 Non-text Content (A)
Toast Notifications	✓ Announces setting changes	✓ Announces setting changes	4.1.3 Status Messages (AA)

The implementation addresses several key mobile-specific considerations:

- 1. Platform-specific adaptations:** The code adjusts feedback mechanisms based on platform capabilities, using `ToastAndroid` for visual feedback and `Vibration` for haptic feedback on Android devices;
- 2. Touch-optimized layout:** The setting rows implement larger touch targets when the `isLargeTouchTargets` option is enabled, as shown by the conditional padding in the style: `paddingVertical: isLargeTouchTargets ? 20 : 16;`
- 3. Multi-sensory feedback:** The implementation provides feedback through multiple channels (visual, auditory, haptic), ensuring users with different sensory capabilities can perceive setting changes;
- 4. Structured grouping:** Related settings are grouped into logical categories with clear headers, helping users with cognitive disabilities understand the organization of settings on a small screen.

1.3.6.5 Implementation overhead analysis

Table 1.30 quantifies the additional code required to implement accessibility features in the Settings screen.

Table 1.30: Settings screen accessibility implementation overhead

Accessibility Feature	Lines of Code	Percentage of Total	Complexity Impact
Semantic Roles	12 LOC	2.1%	Low
Comprehensive Labels	16 LOC	2.8%	Medium
Element Hiding	18 LOC	3.2%	Low
Status Announcements	14 LOC	2.5%	Medium
Platform-specific Feedback	12 LOC	2.1%	Medium
Dynamic Styling	22 LOC	3.9%	Medium
Accessibility State	8 LOC	1.4%	Low
Total	102 LOC	18.0%	Medium

This analysis reveals that implementing accessibility for the Settings screen adds approximately 18.0% to the code base. The most significant contributors are dynamic styling (3.9%) and element hiding (3.2%), reflecting the need to adjust visual presentation based on user preferences and to streamline screen reader navigation.

1.3.6.6 WCAG conformance by principle

Table 1.31 provides a detailed analysis of WCAG 2.2 compliance by principle:

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND
ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

Table 1.31: Settings screen WCAG compliance analysis by principle

Principle	Description	Implementation Level	Key Success Criteria
1. Perceivable	Information and UI components must be presentable to users in ways they can perceive	13/13 (100%)	1.1.1 Non-text Content (A) 1.3.1 Info and Relationships (A) 1.4.3 Contrast (AA) 1.4.4 Resize Text (AA) 1.4.8 Visual Presentation (AAA)
2. Operable	UI components and navigation must be operable	15/17 (88%)	2.3.3 Animation from Interactions (AAA) 2.4.6 Headings and Labels (AA) 2.5.8 Target Size (AA)
3. Understandable	Information and operation of UI must be understandable	10/10 (100%)	3.2.1 On Focus (A) 3.2.2 On Input (A) 3.3.2 Labels or Instructions (A) 3.3.5 Help (AAA)
4. Robust	Content must be robust enough to be interpreted by a wide variety of user agents	3/3 (100%)	4.1.1 Parsing (A) 4.1.2 Name, Role, Value (A) 4.1.3 Status Messages (AA)

The Settings screen achieves 100% compliance with the Perceivable, Understandable, and Robust principles, reflecting its central role in providing accessibility adjustments. The slightly lower compliance with the Operable principle (88%) is due to the absence of specific

keyboard navigation optimizations, which are less relevant in the predominantly touch-based mobile context.

1.3.6.7 Mobile-specific considerations

The Settings screen implementation addresses several mobile-specific accessibility considerations beyond standard WCAG requirements:

1. **Battery-aware implementation:** The screen considers the impact of accessibility features like high contrast and dark mode on battery consumption, which is particularly important for mobile users who may need these features all day;
2. **Touch ergonomics:** The implementation of larger touch targets addresses the specific challenges of touch interaction for users with motor impairments, exceeding the minimum WCAG requirements to provide a more comfortable experience on smaller screens;
3. **Multi-device adaptation:** The settings options are implemented with responsive layouts that adapt to different screen sizes and orientations, ensuring consistency across the diverse range of mobile devices;
4. **Platform convention alignment:** The implementation follows platform-specific visual and interaction patterns, using familiar switch controls and feedback mechanisms that align with user expectations on each platform;
5. **Haptic feedback integration:** The implementation adds haptic feedback (vibration) when settings are changed on Android devices, providing an additional sensory channel that is particularly valuable in mobile contexts where visual attention may be limited.

1.3.6.8 Beyond WCAG: self-adapting interface guidelines

The Settings screen defines several accessibility principles that extend beyond standard WCAG requirements, particularly focusing on the ability of interfaces to adapt to user needs:

CHAPTER 1. BRIDGING THE GAP BETWEEN IMPLEMENTATION AND ACCESSIBILITY GUIDELINES - FULL APP ANALYSIS

1. **Embedded customization principle:** Accessibility adjustments should be directly embedded within the application rather than relying solely on system-level settings. The Settings screen implements this by providing in-app controls for text size, contrast, and other visual preferences;
2. **Multi-sensory feedback guideline:** Changes to accessibility settings should provide feedback through multiple sensory channels. The implementation combines visual cues (toggle animation), auditory feedback (screen reader announcements), and haptic feedback (vibration) to ensure changes are perceivable regardless of user abilities;
3. **Contextual help principle:** Setting controls should provide context-specific guidance on their purpose and effect. The implementation combines descriptive labels with specific hints to help users understand the impact of each setting;
4. **Setting persistence:** User preferences for accessibility features should persist across application sessions. The implementation stores accessibility settings persistently, ensuring users don't need to reconfigure their preferences with each use;
5. **Complementary settings grouping:** Related accessibility settings should be grouped together to help users understand their relationships and combined effects. The implementation organizes settings into logical categories (Visual, Readability, Color & Touch) that reflect how features work together to create accessible experiences.

These guidelines extend WCAG by focusing on the self-adaptation capabilities of interfaces, recognizing that true accessibility requires not just compliance with static criteria but the ability to dynamically adjust to diverse user needs and preferences.

Bibliography

Books

- [16] David A Kolb. *Experiential learning: Experience as the source of learning and development*. Prentice-Hall, 1984.
- [22] Jean Piaget. *Science of education and the psychology of the child*. Orion Press, 1970.
- [30] Lev Vygotsky. *Mind in society: The development of higher psychological processes*. Harvard University Press, 1978.

Articles and papers

- [2] 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.2017.821212](https://doi.org/10.1109/INCISCOS.2017.821212).
- [4] Matteo Budai. “Mobile content accessibility guidelines on the Flutter framework”. In: (2024). URL: <https://hdl.handle.net/20.500.12608/68870> (cit. on p. 7).
- [5] 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](https://doi.org/10.1145/3584701).
- [6] 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.

CHAPTER 1. BIBLIOGRAPHY

- [8] 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.
- [10] 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/ICCSME.2019.00014](https://doi.org/10.1109/ICCSME.2019.00014).
- [14] 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](https://doi.org/10.1109/ISMSIT56059.2022.9932749).
- [15] 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](https://doi.org/10.1145/3377811.3380392).
- [19] 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](https://doi.org/10.1016/j.jme.2022.03.004).
- [20] Alessandro Palmieri and Marco Rossi. “Accessibility in Mobile Applications: A Systematic Review of Challenges and Strategies”. In: *Journal of Mobile Accessibility Research* 15.3 (2022), pp. 234–256. DOI: [10.1016/j.jma.2022.03.001](https://doi.org/10.1016/j.jma.2022.03.001).
- [21] 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](https://doi.org/10.1109/CCNC51664.2024.10454681).
- [23] 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.
- [25] John R Savery. “Overview of problem-based learning: Definitions and distinctions”. In: *Interdisciplinary Journal of Problem-based Learning* 1.1 (2006), p. 3.

CHAPTER 1. BIBLIOGRAPHY

- [26] 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](https://doi.org/10.1109/VL-HCC53370.2022.9833098).
- [27] 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](https://doi.org/10.1109/MS.2023.3245678).
- [32] Etienne Wenger. "Communities of practice: Learning, meaning, and identity". In: (1998).

Sites

- [1] *Accessibility — React Native*. Accessed: 2025-01-26. 2023. URL: <https://reactnative.dev/docs/accessibility>.
- [3] Apple Inc. *Apple Human Interface Guidelines: Accessibility*. Accessed: 2025-02-30. 2024. URL: <https://developer.apple.com/design/human-interface-guidelines/accessibility>.
- [7] Parliament Assembly of the Council of Europe. *The right to Internet access*. Accessed: 2024-04-11. European Union. 2014. URL: <https://assembly.coe.int/nw/xml/XRef/Xref-XML2HTML-en.asp?fileid=20870>.
- [9] Statista Research Department. *Number of smartphone users worldwide from 2014 to 2029*. Accessed: 2024-10-20. Statista. 2024. URL: <https://www.statista.com/statistics/203734/global-smartphone-penetration-per-capita-since-2005/>.
- [11] European Parliament and Council. *Directive (EU) 2016/2102 on the accessibility of the websites and mobile applications of public sector bodies*. Accessed: 2024-10-25. 2016. URL: <https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32016L2102>.
- [12] *Flutter*. Accessed: 2025-01-26. 2023. URL: <https://flutter.dev/>.

CHAPTER 1. BIBLIOGRAPHY

- [13] Google LLC. *Material Design Accessibility Guidelines*. Accessed: 2025-02-30. 2024. URL: <https://m3.material.io/foundations/accessible-design/overview>.
- [18] *Mobile Accessibility Mapping*. Accessed: 2024-10-15. 2015. URL: <https://www.w3.org/TR/mobile-accessibility-mapping/>.
- [24] *React Native*. Accessed: 2024-10-15. 2023. URL: <https://reactnative.dev/>.
- [28] U.S. Access Board. *Section 508 Information and Communication Technology Accessibility Standards*. Accessed: 2024-10-25. 2017. URL: <https://www.access-board.gov/ict/>.
- [29] Ronald L.Mace - North Carolina State University. *Universal Design Principles*. Accessed: 2024-11-04. UC Berkeley. 1997. URL: <https://dac.berkeley.edu/services/campus-building-accessibility/universal-design-principles>.
- [31] *WCAG 2.2 Guidelines*. Accessed: 2024-10-15. 2023. URL: <https://www.w3.org/TR/WCAG22/>.
- [33] World Health Organization. *WHO - Disability*. Accessed: 2024-10-17. World Health Organization. 2023. URL: <https://www.who.int/news-room/fact-sheets/detail/disability-and-health>.