UNIVERSITY OF PADUA
UNIVERSITA' DEGLI STUDI DI PADOVA

Designing an accessibility learning toolkit - Bridging the gap between guidelines and implementation

Department of Mathematics "Tullio Levi-Civita"
Master Degree in Computer Science

Graduate: Gabriel Rovesti – ID: 2103389
Supervisor: Prof. Ombretta Gaggi

Graduation Session: 25/07/2025

DIPARTIMENTO MATEMATICA

First and not least: Mobile accessibility

Definition: Ability for users to fully perceive, understand, navigate, and interact with digital content regardless of capabilities

The mobile reality:

- 1.3 billion people worldwide live with disabilities (WHO, 2023)
- **Mobile-first** era: 6.8 billion smartphone users globally
- Mobile interfaces create new **accessibility barriers**: small screens, orientation changes, performances impact

Accessibility guidelines gap

Current standards:

- **WCAG 2.2 (2023):** 4 principles, 3 levels of conformance - **web-focused**
- **MCAG (2015):** Mobile adaptation - **based on WCAG 2.0**
- **WCAG2Mobile (2025):** Recent mobile guidance - **interpretations only**

The problem:

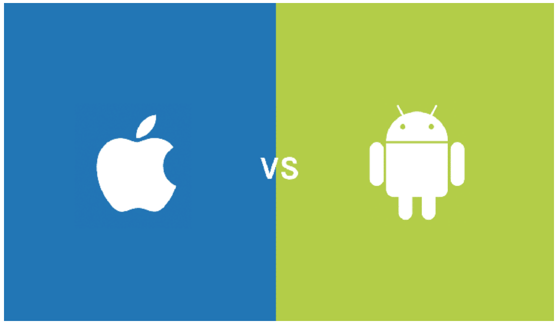
- Implementation void: No comprehensive mobile-native framework since MCAG (2015)
- Guidance gap: Only interpretations, not actionable implementation patterns
- Developer isolation: Scattered resources force ad-hoc accessibility solutions

Result: Lack of comprehensive accessibility implementation and consistent patterns

Platform implementation gap

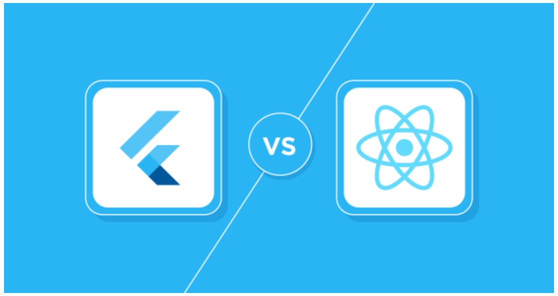
Platform differences:

- **iOS:** VoiceOver, Voice Control, Switch Control
- **Android:** TalkBack, Switch Access



Framework responses:

- **Flutter:** Single accessibility tree, platform adaptation layer
- **React Native:** Platform-specific accessibility props, native bridge



The developer challenge:

- **Budai (2024):** Flutter-focused component accessibility → fragmented knowledge
- **Gap identified:** No comprehensive learning resource bridging platforms

AccessibleHub – Bridging the gap

Research Questions (RQ):

- **RQ1:** Are React Native components accessible by default?
- **RQ2:** Can non-accessible components be made accessible?
- **RQ3:** What's the implementation cost (code overhead)?

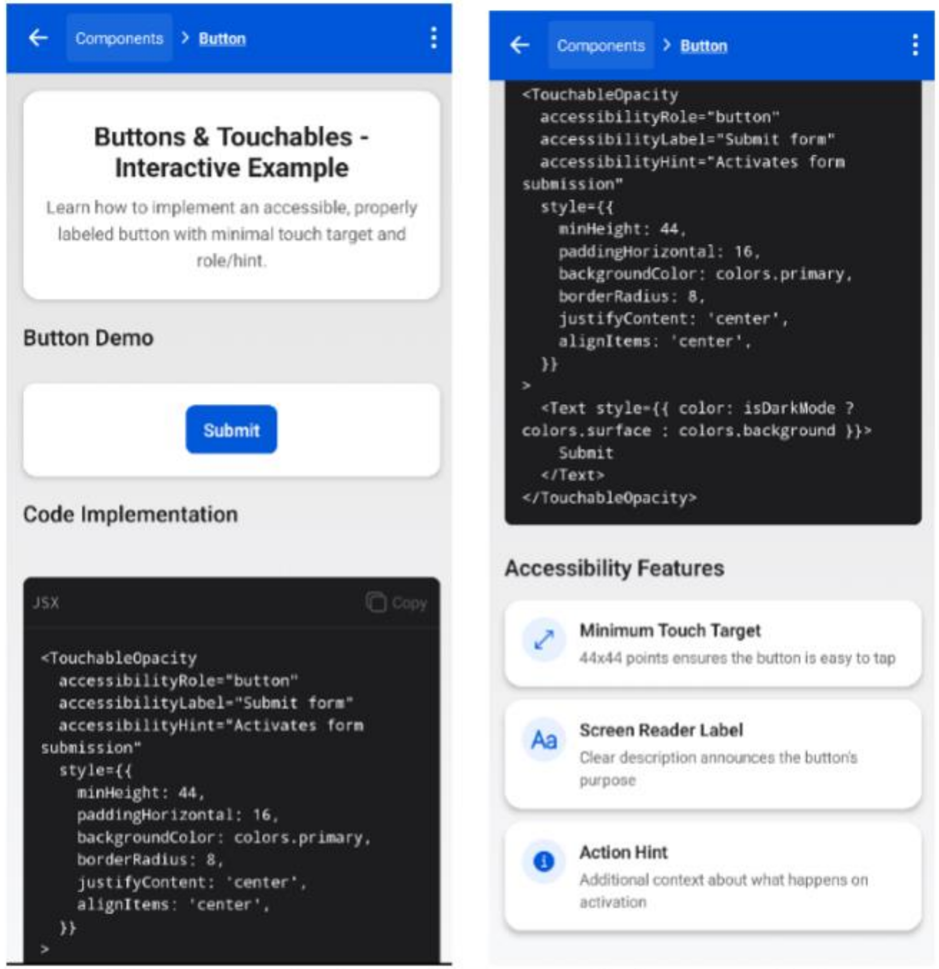
AccessibleHub: React Native application **tested on both Android and iOS** serving as **practical implementation guide** for mobile developers
- **Every screen analyzed** for accessibility patterns + implementation costs
- **Developer-first platform** bridging WCAG theory to executable code

A comprehensive toolkit for implementing accessibility in React Native

React Native v0.73 | WCAG 2.2 | Expo SDK

AccessibleHub – Overview

From theory...



(a) Button screen - Part 1 (b) Button screen - Part 2

...to practice!

Live demos (Android & iOS): hands on!

Systematic analysis approach

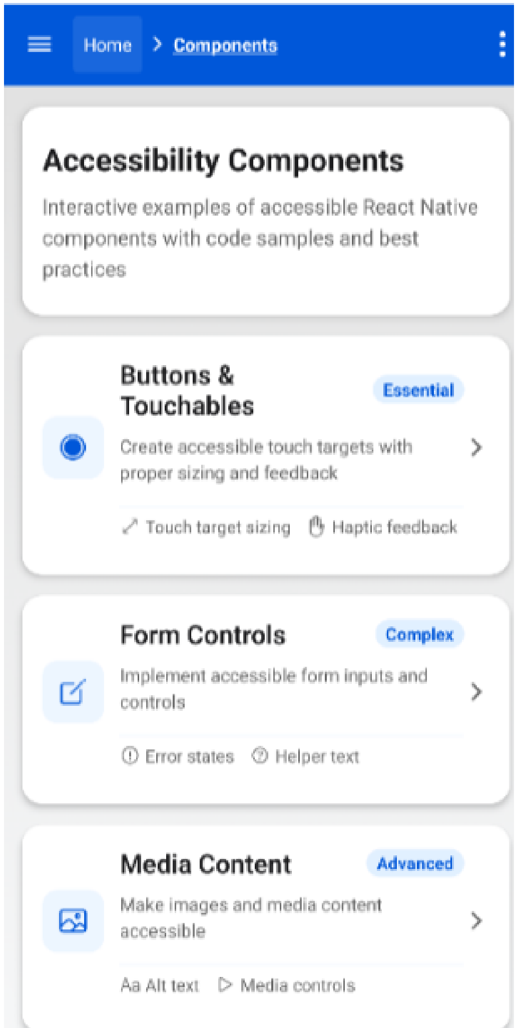
The transformation challenge: WCAG guidelines are abstract and difficult to implement directly in mobile code

Layer	Input	Output	Key Innovation
Theoretical Foundation	WCAG abstract principles	Success criteria	Systematic WCAG→mobile mapping
Implementation Patterns	Success criteria	React Native code	**Quantified formal metrics**
Screen-Based Analysis	Code patterns	Empirical metrics	**VoiceOver + TalkBack testing protocols**
Platform Integration	Metrics	Educational tool	**First unified measurement platform**

Basic workflow - Enabling data-driven accessibility decisions

Abstract WCAG ➞ Implementation Patterns ➞ Quantified Metrics

1 - Component-criteria mapping



(a) Components screen - Top section

Table 3.5: Components screen component-criteria mapping with WCAG2Mobile considerations

Component	Semantic Role	WCAG 2.2 Criteria	WCAG2Mobile Considerations	Implementation Properties
ScrollView Container (main screen container)	scrollview	2.1.1 Keyboard (A) 2.4.3 Focus Order (A)	Screen-based navigation patterns; Touch-based scrolling alternatives	`accessibilityRole="scrollview"`, `accessibilityLabel="Accessibility Components Screen"`
Hero Title (top center: "Accessibility Components")	header	1.4.3 Contrast (AA) 2.4.6 Headings (AA)	Screen title differentiation; Proper semantic structure in screen context	`accessibilityRole="header"`

2 - Empirical screen reader testing


```
{/* 2. Component card with comprehensive accessibility label */}
<TouchableOpacity
  style={themedStyles.card}
  onPress={() => handleComponentPress('/components/button', 'Buttons
      & Touchables')}
  accessibilityRole="button"
  accessibilityLabel="Buttons and Touchables component. Create
      accessible
    touch targets with proper sizing and feedback. Essential
        component type."
>
```

Table 3.6: Components screen screen reader testing with WCAG2Mobile focus

Test Case	VoiceOver (iOS)	TalkBack (Android)	WCAG2Mobile Considerations
Screen Title	✓ Announces “Accessibility Components, heading”	✓ Announces “Accessibility Components, heading”	SC 1.3.1 and 2.4.6 interpreted for screens instead of web pages
Component Card	✓ Announces complete label with component description and complexity	✓ Announces complete label with component description and complexity	SC 4.1.2 for mobile context requires comprehensive labels for navigation decisions

3 - Implementation overhead analysis

Table 3.7: Components screen accessibility implementation overhead

Accessibility Feature	Lines of Code	Percentage of Total Code	Complexity Impact
Semantic Roles	15 LOC	2.6%	Low
Descriptive Labels	28 LOC	4.9%	Medium
Element Hiding	18 LOC	3.2%	Low
Focus Management	22 LOC	3.9%	Medium
Contrast Handling	14 LOC	2.5%	Medium
Announcements	12 LOC	2.1%	Low
Breadcrumb Implementation	42 LOC	7.4%	High
Drawer Accessibility	35 LOC	6.2%	High
AAA-specific Enhancements	20 LOC	3.5%	Medium
Total	**206 LOC**	**36.3%**	**Medium-High**

Component implementation patterns

Key finding: Systematic correlation enables predictable accessibility cost estimation across frameworks

Component Type	Complexity Driver	React Native Baseline	Pattern Validation
Media	Content description	12.7% overhead	✓ Consistent (Low)
Buttons	Semantic roles, labels	13.3% overhead	✓ Consistent (Low)
Dialogs	Navigation control	16.2% overhead	✓ Consistent (Medium)
Forms	State management	21.5% overhead	✓ Consistent (Medium-High)
Advanced	Custom interactions	22.7% overhead	✓ Consistent (High)

Foundation for framework-agnostic accessibility cost prediction

Formal evaluation metrics

Innovation: Evidence-based methodology for quantifying mobile accessibility implementation across frameworks

🔢 **Implementation Overhead (IMO)**
Direct code cost measurement for equivalent functionality

📱 **Screen Reader Support Score (SRSS)**:
Likert scale based on VoiceOver/TalkBack functionality

✅ **WCAG Compliance Ratio (WCR)**:
Standards adherence tracking (A/AA/AAA levels)

⚙️ **Complexity Impact Factor (CIF)**:
Development difficulty classification (Low/Medium/High)

⏱️ **Development Time Estimate (DTE)**:
Resource planning with complexity adjustments

Framework comparison

UNIVERSITÀ DEGLI STUDI DI PADOVA

Architecture differences:

- **React Native:** Property-based model
  - `accessibilityLabel`, `accessibilityRole`
- **Flutter:** Widget-based approach
  - Explicit `Semantics` adapters

```
<Text accessibilityRole="header">Settings</Text>

<TouchableOpacity
  accessibilityLabel="Save document"
  onPress={handleSave}>
  Save
</TouchableOpacity>
```

VS

```
Semantics(
  header: true,
  child: Text("Settings")
)

Semantics(
  button: true, label: "Save document",
  child: GestureDetector(
    onTap: handleSave,
    child: Text("Save")
  )
)
```

React Native - Property integration

Flutter - Semantic wrappers

Framework comparison results - 1

Component	React Native	Flutter	Code Overhead	Screen Reader Support
Text Language	✓	X	Flutter +200%	RN: 4.2, FL: 3.7
Headings	X	X	Flutter +57%	RN: 4.3, FL: 4.0
Form Fields	X	X	Flutter +53%	RN: 4.0, FL: 3.8
Custom Gestures	X	X	Flutter +27%	RN: 3.8, FL: 3.2
OVERALL	**38%**	**32%**	**Flutter +119%**	**RN: 4.2, FL: 3.8**

✓ = Default accessibility support | X = Requires developer implementation

Key patterns identified:

- **Text language declaration**: Largest overhead difference (Flutter +200%)
- **Custom gestures**: Smallest gap (Flutter +27%) - both frameworks struggle
- **Default accessibility**: React Native provides more out-of-box features (38% vs 32%)
- **Screen reader consistency**: React Native scores higher across all component types

Framework comparison results - 2

Implementation Aspect	React Native	Flutter	Impact
Code Integration	Direct properties	Wrapper layers	Flutter +119% more code
Developer Experience	Component modification	Semantic layer addition	RN: Moderate vs FL: Steep
Default Accessibility	38% out-of-box	32% out-of-box	Both require intervention
Screen Reader Support	4.2/5.0	3.8/5.0	RN: Better consistency
WCAG Compliance (AA)	92%	85%	RN: +8.2% advantage

Insights for development choice:

- **React Native** for: Rapid development timeline, web accessibility knowledge, cross-platform consistency priority
- **Flutter** for: Complex custom components, dedicated long-term maintenance team, granular accessibility control

Research impact and conclusions

Key contributions:
- **Systematic evaluation methodology** on evidence-based formal metrics
- **First quantitative framework** for cross-platform accessibility assessments
- **Systematic methodology** bridging WCAG theory to mobile practice

Research answers:
- **RQ1**: No framework accessible by default – intervention required for both
- **RQ2**: Both achieve high WCAG compliance with proper implementation
- **RQ3**: React Native requires 45% less code for equivalent accessibility