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** for assistive technology users

First and not least: Mobile challenges

Touch interaction barriers:

- **Target size** not standardized and difficult to use
- **Complex gestures** exclude motor-impaired users
- **One-handed operation** limitations

Mobile context issues:

- **Small screens** affect content hierarchy
- **Orientation changes** disrupt navigation
- **Performance impact** on battery and processing

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:

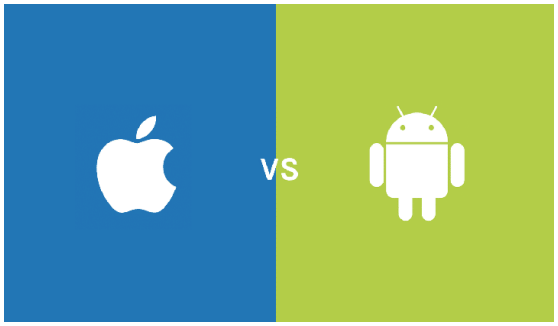
- **Outdated foundation:** MCAG missing WCAG 2.1/2.2 mobile criteria
- **Implementation void:** No practical framework for mobile developers
- **Knowledge fragmentation:** Scattered resources, unclear costs

Result: Mobile developers lack comprehensive accessibility implementation guidance

Platform implementation gap

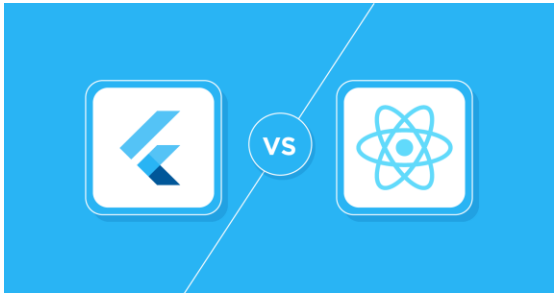
Platform differences:

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



Framework responses:

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



The developer challenge:

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

AccessibleHub – Bridging the gap

Research Questions as standard approach:

- **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 **interactive accessibility manual** for mobile developers

- **Every screen analyzed** for accessibility patterns and costs
- **Educational platform** bridging theory to practice

A comprehensive toolkit for implementing accessibility in React Native

AccessibleHub – Overview

Core application sections:

- 🔧 **Accessible Components**: UI implementations with copyable code
- 📚 **Best Practices**: Educational content on accessibility challenges
- 🛠️ **Tools Screen**: Resource catalog for testing and evaluation
- 📊 **Framework Comparison**: Evidence-based evaluation methodology
- 👥 **Community Integration**: Social learning and collaborative resources

Research innovation: Every screen analyzed as case study

- **20+ components** tested with TalkBack and VoiceOver
- **Cross-platform validation** ensuring patterns work universally

Systematic analysis approach

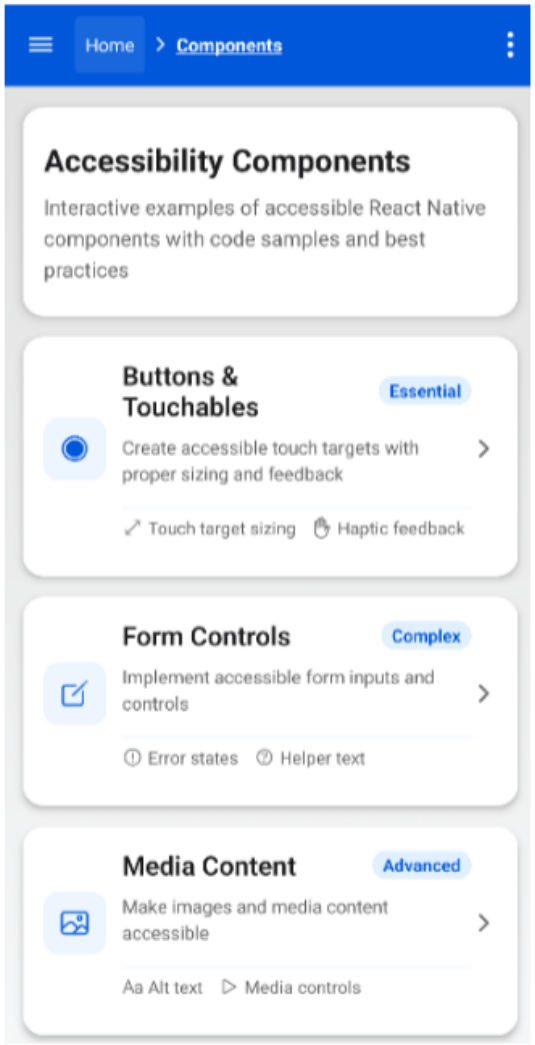
The transformation challenge: WCAG guidelines are abstract and difficult to implement directly in mobile code

Layer	Input	Output	Example
Theoretical Foundation	WCAG abstract principles	Success criteria	"Content must be perceivable"
Implementation Patterns	Success criteria	React Native code	`accessibilityLabel="Save document"`
Screen-Based Analysis	Code patterns	Quantified metrics	13.3% overhead for buttons

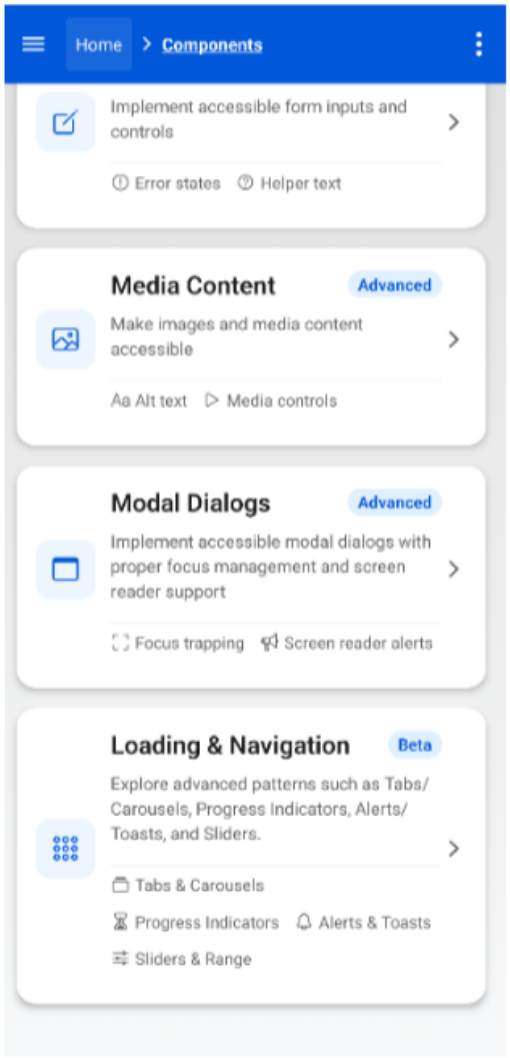
Basic workflow:

Abstract WCAG → Implementation Patterns → Quantified Metrics → Educational Platform

Systematic analysis – Example (1)



(a) Components screen - Top section



(b) Components screen - Bottom section

Systematic analysis – Example (2)

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	`accessibility Role="scrollview", accessibility Label="Accessibility Components Screen"`

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

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

Accessibility implementation costs

Key finding: Accessibility implementation requires 12-23% additional code across component types

Component Type	Complexity Level	Code Overhead	Primary Contributors
Media	Low	12.7%	Alt text, captions
Buttons	Low	13.3%	Semantic roles, labels
Dialogs	Medium	16.2%	Focus management
Forms	Medium	21.5%	State management, error handling
Advanced	High	22.7%	Custom controls, gestures

Critical insights:

- **Even complex components stay under 25%**
- **Correlation** between interaction complexity and implementation cost
- **Manageable overhead** for significant usability improvements
- **First quantitative framework** for mobile accessibility cost assessment

Beyond WCAG: Extended Principles

Extending WCAG Principles:

🔄 **Multi-dimensional Evaluation:** Beyond compliance - includes empirical testing + real-world usability

🔍 **Methodology Transparency:** Clear evaluation documentation, test procedures, accountability

📚 **Academic Grounding:** Peer-reviewed research integration, formal standards connection

📊 **Progressive Disclosure:** Layered complexity for different developer skill levels

👥 **Social Learning Integration:** Community-driven knowledge sharing

Result: First comprehensive developer resource bridging accessibility theory with mobile practice

Formal evaluation metrics

Innovation: First structured methodology for quantifying mobile accessibility implementation across frameworks

Metric	Purpose	Implementation
CAS - Component Accessibility Score	Default accessibility assessment	Weighted: Screen readers (30%), Semantics (30%), Gestures (20%), Focus (20%)
IMO - Implementation Overhead	Code cost measurement	Direct LOC counting for equivalent implementations
CIF - Complexity Impact Factor	Development difficulty	Low/Medium/High classification with nesting depth analysis
SRSS - Screen Reader Support Score	Cross-platform compatibility	Empirical TalkBack/VoiceOver testing (1-5 scale)
WCR - WCAG Compliance Ratio	Standards adherence	A/AA/AAA criteria tracking across components
DTE - Development Time Estimate	Resource planning	Complexity-based time projections

Framework comparison

Architecture differences:

- **React Native:** Property-based model (`accessibilityLabel`, `accessibilityRole`)
- **Flutter:** Widget-based approach (explicit `Semantics` wrappers)

Implementation findings:

- **React Native:** Higher implementation overhead, better native integration
- **Flutter:** Lower baseline accessibility, requires more manual implementation
- **Code verbosity:** React Native more concise for accessibility features

Cross-Platform Insights: Framework selection significantly impacts accessibility implementation complexity and developer experience

Framework comparison results (1)

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

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)

Metric	React Native	Flutter	Key Finding
Default Accessibility	38%	32%	RN +6% advantage
Implementation Overhead	Baseline	+119% more code	RN significantly more efficient
Screen Reader Support	4.2/5	3.8/5	RN better cross-platform consistency
WCAG Compliance (AA)	92%	85%	RN +8.2% higher compliance

Critical insights:

- **React Native:** Property-based model - 45% less code, better platform integration
- **Flutter:** Widget-based model - more explicit semantics, higher implementation cost
- **Framework selection significantly impacts** accessibility development efficiency

Research impact and conclusions

Key contributions:

- **First quantitative framework** for mobile accessibility implementation cost assessment
- **Systematic methodology** transforming abstract WCAG principles into practical code patterns

Practical impact:

- **Developers:** Clear cost-benefit analysis for accessibility implementation prioritization
- **Organizations:** Evidence-based framework selection guidance for accessible mobile projects
- **Academic community:** Reproducible methodology for accessibility evaluation