11:47

Components > Dialog

Modal Dialogs - Interactive Example

Build dialogs with focus trapping, screen reader support, and proper roles.

Example Dialog

This is an example of an accessible dialog with proper focus management, keyboard interactions, and screen reader announcements.

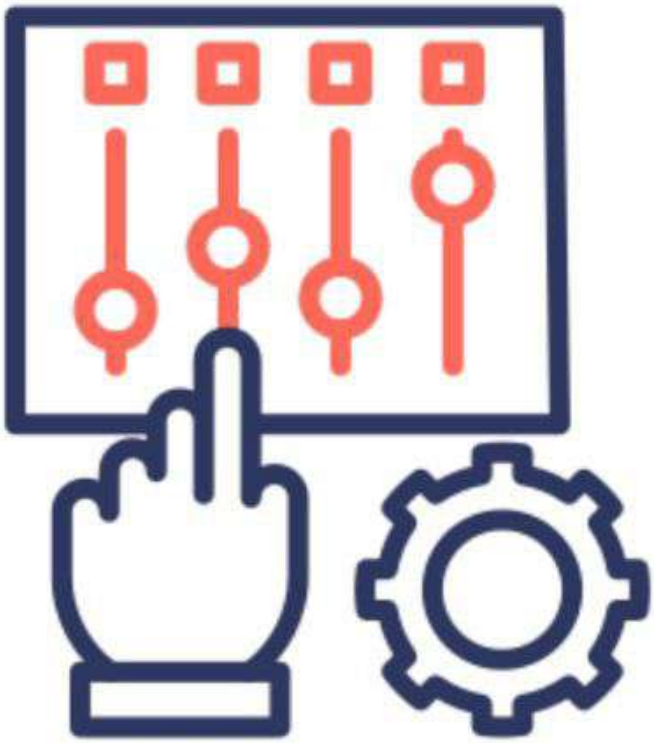
Cancel Confirm

```
const AccessibleDialog =
({ visible, onClose, title,
children }) => {

  return (
    <Modal
      visible={visible}
      transparent
      animationType="fade"

onRequestClose={onClose}
```

Media Content - Interactive Example

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

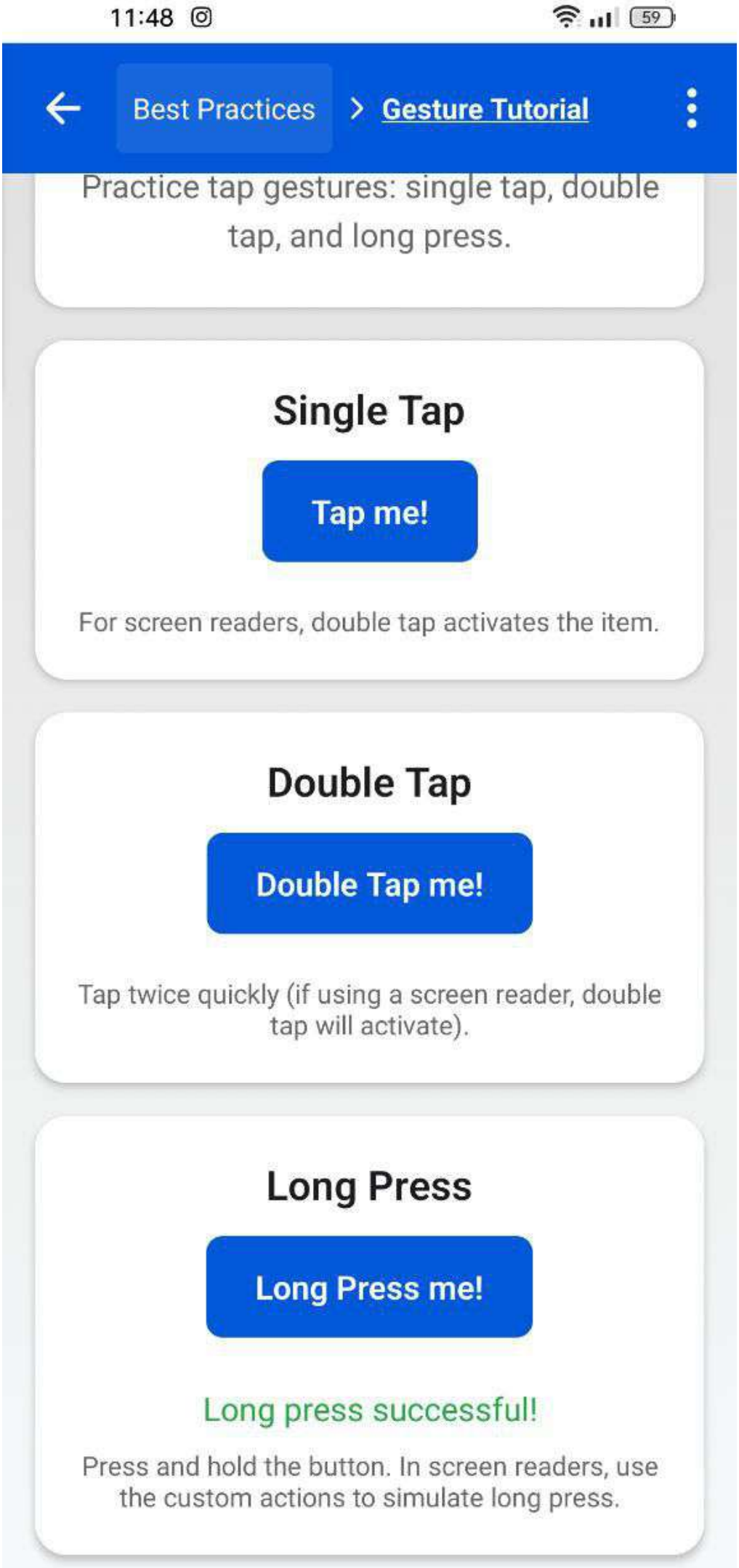
Media Demo



Hide Alt Text

Alternative Text:

A placeholder image (third


11:48
59
Best Practices
Gesture Tutorial
Practice tap gestures: single tap, double tap, and long press.
Single Tap
Tap me!
For screen readers, double tap activates the item.
Double Tap
Double Tap me!
Tap twice quickly (if using a screen reader, double tap will activate).
Long Press
Long Press me!
Long press successful!
Press and hold the button. In screen readers, use the custom actions to simulate long press.

- Help users navigate and find content

Understandable

Information and the operation of user interface must be understandable.

- Make text readable and understandable
- Make content appear and operate in predictable ways
- Help users avoid and correct mistakes

Robust

Content must be robust enough that it can be interpreted by a wide variety of user agents, including assistive technologies.

- Maximize compatibility with current and future user tools

```
interpolate({
      inputRange: [0, 100],
      outputRange: ['0%',
'100%'],
    }),
  }}
/>
```

Alerts & Toasts

Trigger Alert

Something happened!

JSX Copy

```
const [showToast,
setShowToast] =
useState(false);

function showToastMessage()
{
  setShowToast(true);
  AccessibilityInfo.announ
ceForAccessibility('Alert:
Something happened');
  setTimeout(() =>
setShowToast(false), 2000);
}
```

11:47

Components > Dialog

Modal Dialogs - Interactive Example

Build dialogs with focus trapping, screen reader support, and proper roles.

Dialog Demo

Success!

Your action has been confirmed.

JSX Copy

```
const AccessibleDialog =
({ visible, onClose, title,
children }) => {

  return (
    <Modal
      visible={visible}
      transparent
      animationType="fade"

onRequestClose={onClose}
```

Mobile Accessibility Best Practices

Essential guidelines for creating accessible React Native applications

WCAG Guidelines

Documentation

Understanding and implementing WCAG 2.2 guidelines in mobile apps

Success Criteria Examples

Semantic Structure

Code Examples

Creating meaningful and well-organized content hierarchies

Hierarchy Implementation

Gesture Tutorial

Interactive Guide

Learn and test common accessibility gestures

Advanced Accessible Components

Demonstrating Tabs/Carousels, Progress Indicators, Alerts/Toasts, and Sliders in one screen

Tabs & Carousels

Tab One | Tab Two | Tab Three

Current tab: Tab One

JSX Copy

```jsx
const [selectedTab,
setSelectedTab] =
useState(0);
const tabs = ['Tab One',
'Tab Two', 'Tab Three'];

<View
style={{ flexDirection:
'row' }}
accessibilityRole="tablist">
  {tabs.map((tab, idx) => (
    <TouchableOpacity
      key={idx}
```

Screen Reader Support

Comprehensive guide for optimizing your app for VoiceOver and TalkBack

VoiceOver (iOS)

TalkBack (Android)

Essential Gestures

Single tap

Select an item

Double tap

Activate selected item

Three finger swipe up/down

Scroll content

Three finger tap

Speak current page

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

Accessibility Features

Alternative Text

Descriptive text that conveys the content and function of the image

Role Announcement

Screen readers announce the element as an image

Touch Target

Interactive images should have adequate touch targets

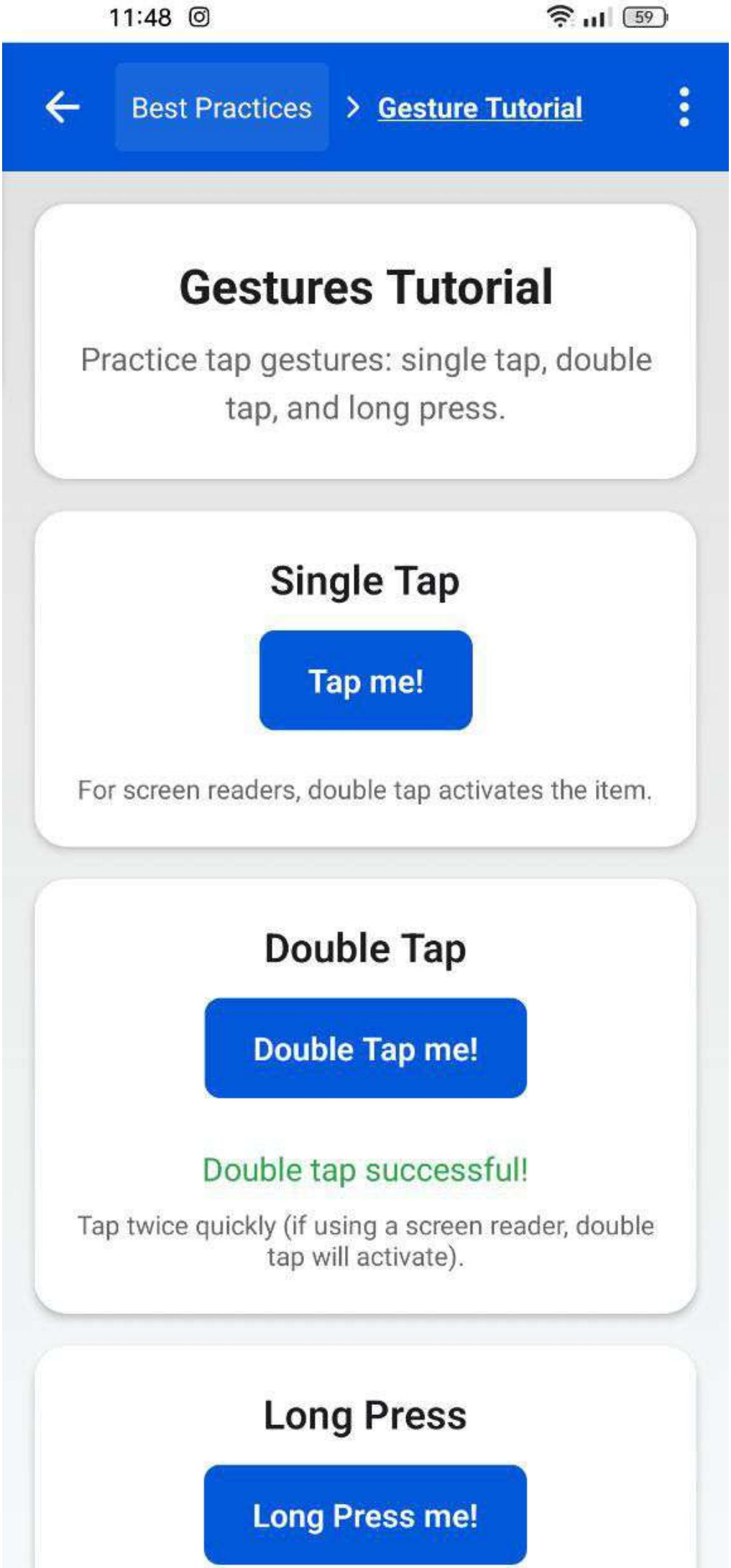
Sliders & Range Inputs

30%

0% 25% 50% 75% 100%

JSX Copy

```
<View
  accessible={true}

accessibilityLabel={`Slider
control, current value $
{sliderValue} percent`}
>

{/* Button Controls for
Screen Reader Users */}

<View
style={{ flexDirection:
'row', justifyContent:
'space-between' }}>
    <TouchableOpacity
      onPress={() => {
        const newValue =
Math.max(0, sliderValue -
```


11:48
59
Best Practices
Gesture Tutorial
Gestures Tutorial
Practice tap gestures: single tap, double tap, and long press.
Single Tap
Tap me!
For screen readers, double tap activates the item.
Double Tap
Double Tap me!
Double tap successful!
Tap twice quickly (if using a screen reader, double tap will activate).
Long Press
Long Press me!

```
Text>
            </
TouchableOpacity>
          </View>
        </View>
      </View>
    </Modal>
  );
};
```

Accessibility Features

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

```
    importantForAccessibilit
y="no-hide-descendants"
  />
</View>
```

Accessibility Features

Tab Navigation

Proper role and state management for tab controls

Progress Updates

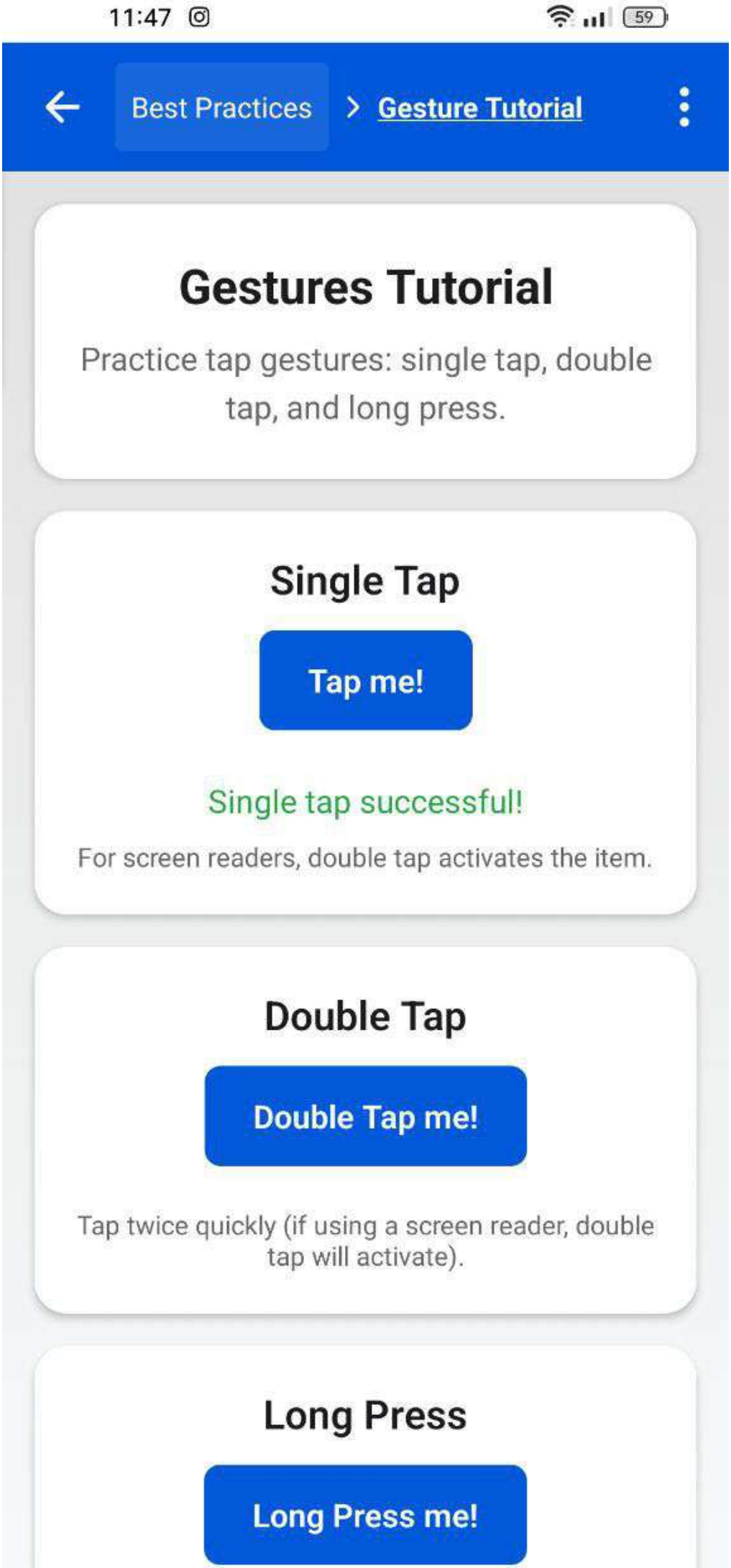
Live announcements of progress changes

Alert Notifications

Immediate feedback for important events

Slider Controls

Accessible range inputs with value announcements


11:47
59
Best Practices
Gesture Tutorial
Gestures Tutorial
Practice tap gestures: single tap, double tap, and long press.
Single Tap
Tap me!
Single tap successful!
For screen readers, double tap activates the item.
Double Tap
Double Tap me!
Tap twice quickly (if using a screen reader, double tap will activate).
Long Press
Long Press me!

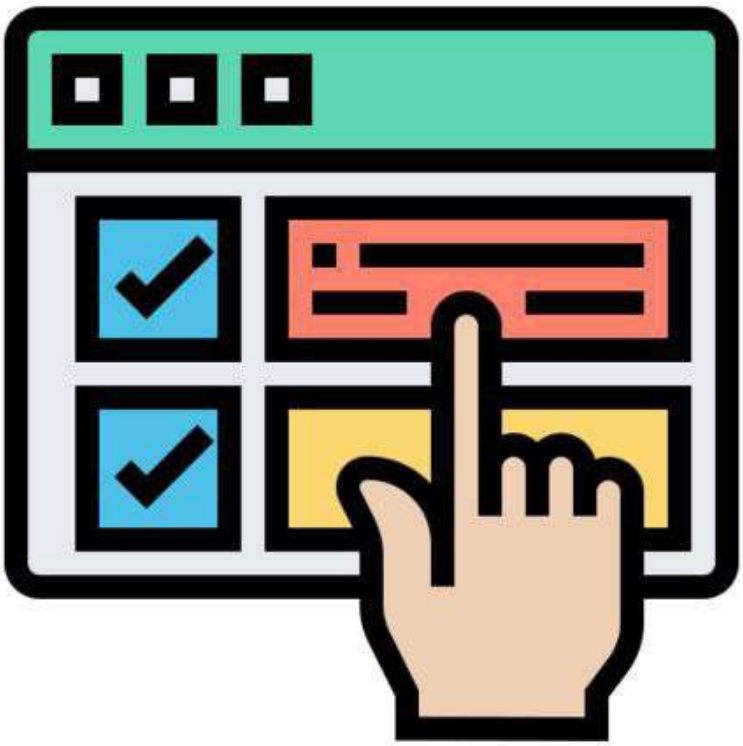
11:46

Components > Media Content

Media Content - Interactive Example

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

Media Demo



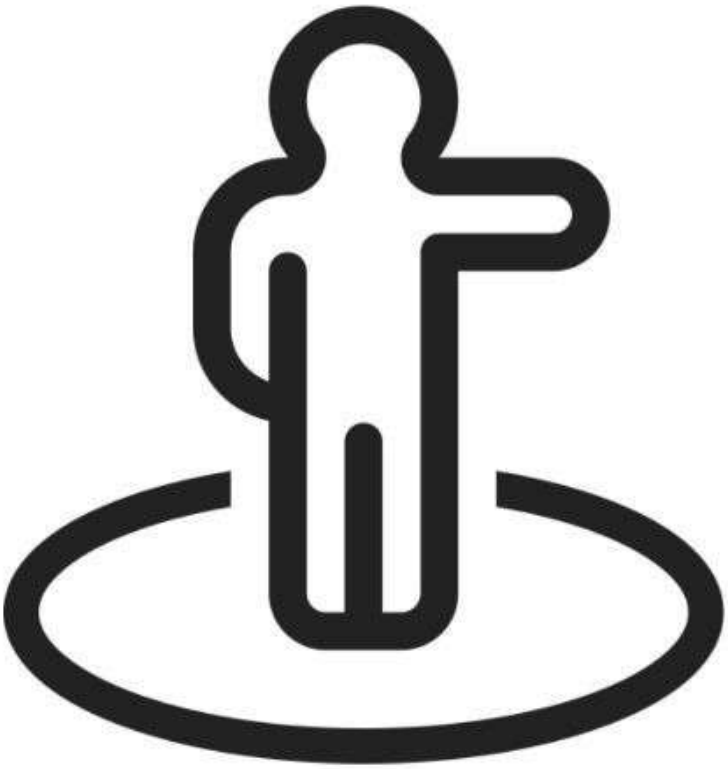
Show Alt Text

Code Implementation

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 (second

11:48

Best Practices > Screen Reader Support

Screen Reader Support

Comprehensive guide for optimizing your app for VoiceOver and TalkBack

VoiceOver (iOS)

TalkBack (Android)

Essential Gestures

Single tap

Move focus and announce

Double tap

Activate focused item

Swipe right/left

Move to next/previous item

Two finger swipe up/down

Scroll content

11:49

Home > Framework Comparison

Framework Comparison

Evidence-based comparison of accessibility features in React Native and Flutter

React Native | Flutter

Overview | Accessibility

Implementation Details

Analysis of accessibility implementation requirements for React Native

Implementation Overview

Implementation Complexity:	5.0/5
Total Lines of Code Required:	21
Default Accessible Components:	1/3

Heading Elements WCAG: 1.3.1, 2.4.6

11:45

Home

Development Resources

Best Practices

Comprehensive WCAG 2.2 implementation guidelines for React Native

WCAG 2.2 | Guidelines

Testing Tools

Essential tools and methods for accessibility verification

TalkBack | VoiceOver

Framework Comparison

Detailed analysis of accessibility support across mobile frameworks

React Native | Flutter

Join the A11y Community

Connect with developers committed to making React Native accessible for everyone. Share, learn, and contribute to projects that create inclusive mobile experiences.

Explore Open Issues

Projects Seeking Contributors

ESLint A11y Plugin

237 contributors 14 open issues

Static analysis tool that catches accessibility issues in your React Native apps during development.

linting static-analysis open-source

Contribute

React Native Testing Library

112 contributors 23 open issues

Solution for testing React Native components that

By Accessibility Research Team

Reduced Motion & Inclusive Animations

Allow users to disable or reduce motion. React Native can respect OS-level reduce-motion settings, or provide a manual toggle.

Show Details

By React Native Community

Join Community Channels

A11y Stack Exchange

8.7K+ members

Q&A platform with tagged questions specific to React Native accessibility challenges.

Accessibility Twitter Community

15K+ members

Follow the #ReactNativeA11y hashtag to stay updated with the latest discussions.

11:49

Home > Framework Comparison

Framework Comparison

Evidence-based comparison of accessibility features in React Native and Flutter

React Native | Flutter

</> Implementation | Methodology

Research Methodology

This comparison is based on empirical testing and analysis of official documentation.

Accessibility Testing Methodology

Combined analysis of official documentation, practical testing with screen readers (VoiceOver iOS 16, TalkBack Android 13), and WCAG 2.2 compliance verification

Sources:

- Official framework documentation
- Perinello & Gaggi (2024), 'Accessibility of Mobile User Interfaces using Flutter and React

11:49

Home > Framework Comparison

Framework Comparison

Evidence-based comparison of accessibility features in React Native and Flutter

Accessibility Score Methodology

Methodology | Calculation | References

The accessibility score is calculated based on screenreader compatibility, semantic richness, gesture handling, and focus management capabilities.

Test Devices: iPhone 13 (iOS 16.5), Pixel 6 (Android 13)

4.5

Semantic Support

Extensive semantic property support

- accessibilityLabel
- accessibilityHint

VoiceOver (iOS) Built-in

Development Tools

Accessibility Inspector

Contrast Analyzer

Tool to verify color contrast ratios according to WCAG guidelines:

- Assess text contrast
- Verify UI component contrast
- Support for WCAG 2.2 standards

Practical Usage:

The Contrast Analyzer is essential for ensuring text readability, helping you choose color combinations that meet standards for users with low vision.

Documentation

Accessibility Linter

Testing Checklist

Automated Testing

Skip to Main Content

Why Focus Order Matters

Proper focus order helps screen reader and keyboard users navigate without confusion. A skip link allows bypassing repetitive blocks, ensuring more efficient access to primary content.

Main Content

Below are interactive elements in a logical sequence.

Focusable Button 1

Enter feedback

Submit Feedback

Logical Focus Order

Demonstrate skip links and consistent navigation order.

Skip to Main Content

Why Focus Order Matters

Proper focus order helps screen reader and keyboard users navigate without confusion. A skip link allows bypassing repetitive blocks, ensuring more efficient access to primary content.

Main Content

Below are interactive elements in a logical sequence.

Focusable Button 1

Enter feedback

11:47

Home > Best Practices

Gesture Tutorial

Interactive Guide

Learn and test common accessibility gestures

Gesture Patterns

Interactive Demo

Screen Reader Support

Guidelines

Optimizing your app for VoiceOver and TalkBack

Platform-specific

Gestures

Logical Focus Order

Interactive Guide

Managing focus and keyboard navigation effectively

Focus Flow

Interactive Demo

Contrast Analyzer

Accessibility Linter

Testing Checklist

Automated Testing

Accessibility Scanner (Android)

An app for scanning accessibility issues on Android devices:

- Identify potential accessibility improvements
- Provide recommendations based on WCAG
- Easy to use on any Android device

Practical Usage:

Use Accessibility Scanner to run quick tests on your Android builds, ensuring that all UI components meet accessibility standards and receive actionable recommendations.

Documentation

Mobile Accessibility Tools

Explore essential tools for testing and improving the accessibility of your mobile apps.

Screen Readers

TalkBack (Android) Built-in

Native Android screen reader. Essential gestures:

- Single tap: Select an element
- Double tap: Activate selected element
- Swipe right/left: Navigate between elements

Practical Usage:

TalkBack allows you to test navigation and interactions for users with visual impairments, ensuring every component has clear labels and hints.

Documentation

VoiceOver (iOS) Built-in

Development Tools

WCAG 2.2 Guidelines

Essential principles for building accessible mobile apps

Perceivable

Information and user interface components must be presentable to users in ways they can perceive.

- Provide text alternatives for non-text content
- Provide captions and other alternatives for multimedia
- Create content that can be presented in different ways without losing meaning
- Make it easier for users to see and hear content

Operable

11:48

Best Practices > Screen Reader Support

Screen Reader Support

Comprehensive guide for optimizing your app for VoiceOver and TalkBack

VoiceOver (iOS)

TalkBack (Android)

Implementation Guide

Semantic Structure

- Use proper heading hierarchy
- Implement meaningful landmarks
- Group related elements logically

View Code Examples →

Aa Content Descriptions

- Provide clear accessibilityLabels
- Include meaningful hints
- Describe state changes

View Guidelines →

A11y Stack Exchange

8.7K+ members

Q&A platform with tagged questions specific to React Native accessibility challenges.

Accessibility Twitter Community

15K+ members

Follow the #ReactNativeA11y hashtag to stay updated with the latest discussions.

Developer Toolkit

Essential resources to improve accessibility in your next React Native project:

iOS Guidelines

Apple's official accessibility guidelines and best practices

Android Guidelines

Google's official accessibility documentation and best practices

Semantic Structure

Building meaningful and well-organized content hierarchies

Content Hierarchy

Proper headings and landmarks help users quickly parse content. Avoid styling text as a heading without providing a semantic role. In React Native, use **accessibilityRole="heading"** for key titles.

```
// Example of multiple heading levels
            <View accessibilityRole="header">
              <Text accessibilityRole="heading" /* Level 1 equivalent */>
                Main Title (H1)
              </Text>
            </View>

            <View accessibilityRole="main">
              <Text accessibilityRole="heading" /* Level 2 equivalent */>
```

11:49

Home > Framework Comparison

Framework Comparison

Evidence-based comparison of accessibility features in React Native and Flutter

React Native | Flutter

</> Implementation | Methodology

Research Methodology

This comparison is based on empirical testing and analysis of official documentation.

Accessibility Testing Methodology

Combined analysis of official documentation, practical testing with screen readers (VoiceOver iOS 16, TalkBack Android 13), and WCAG 2.2 compliance verification

Sources:

- Official framework documentation
- Perinello & Gaggi (2024), 'Accessibility of Mobile User Interfaces using Flutter and React

11:49

Home > Framework Comparison

Framework Comparison

Evidence-based comparison of accessibility features in React Native and Flutter

React Native | Flutter

Overview | Accessibility

React Native

by Meta (Facebook)

Version 0.73

A framework for building native applications using React

Language
JavaScript/TypeScript

Learning Curve
Moderate

Hot Reload
Yes

```
<View accessibilityRole="main">
  ...
</View>
```

Landmarks & ARIA Roles

Define distinct areas (e.g., navigation, complementary, contentinfo) to aid comprehension. In React Native, you can mimic these with **accessibilityRole** or custom logic.

- Use **accessibilityRole="navigation"** for top-level nav
- Provide **accessibilityRole="complementary"** for sidebars
- Mark footers with **accessibilityRole="contentinfo"**

Resources & Next Steps

Learn more about headings, landmarks, and ARIA roles:

- W3C WAI: https://www.w3.org/WAI/
- ARIA Roles: https://www.w3.org/TR/wai-aria-1.2/

11:45

Home > Settings

VISUAL SETTINGS

Dark Mode
Enable dark theme for low light conditions

High Contrast Mode
Increase contrast for improved readability

READABILITY ENHANCEMENTS

Large Text
Increase text size for improved readability

Reduce Motion
Minimize animations for improved comfort

COLOR & TOUCH SETTINGS

Color Filter

```
    backgroundColor:
colors.primary,
    borderRadius: 8,
    justifyContent: 'center',
    alignItems: 'center',
  }}
>
  <Text style={{ color:
isDarkMode ? colors.surface :
colors.background }}>
    Submit
  </Text>
</TouchableOpacity>
```

Accessibility Features

Minimum Touch Target

44x44 points ensures the button is easy to tap

Screen Reader Label

Clear description announces the button's purpose

Action Hint

Additional context about what happens on activation

Form Controls - Interactive Example

Build accessible, validated forms with proper labels, roles, hints, and date/time pickers.

Form Demo

Name

Email

Gender

Male Female

Preferred Contact Time

Morning Afternoon

Evening

Birth Date

Tap to select time

Communication Preferences

- [ ] Email
- [ ] Phone
- [ ] SMS

- [ ] Agree to terms and conditions

Complete all required fields to proceed

Submit

Code Implementation

JSX Copy

```
<View
accessibilityRole="form">

  {/* Input Field */}
  <Text style={styles.label}
>Name</Text>
  <TextInput
```

11:46

Home

WCAG 2.2 Compliance

Overview | Details | Methodology | References

WCAG 2.2 Compliance

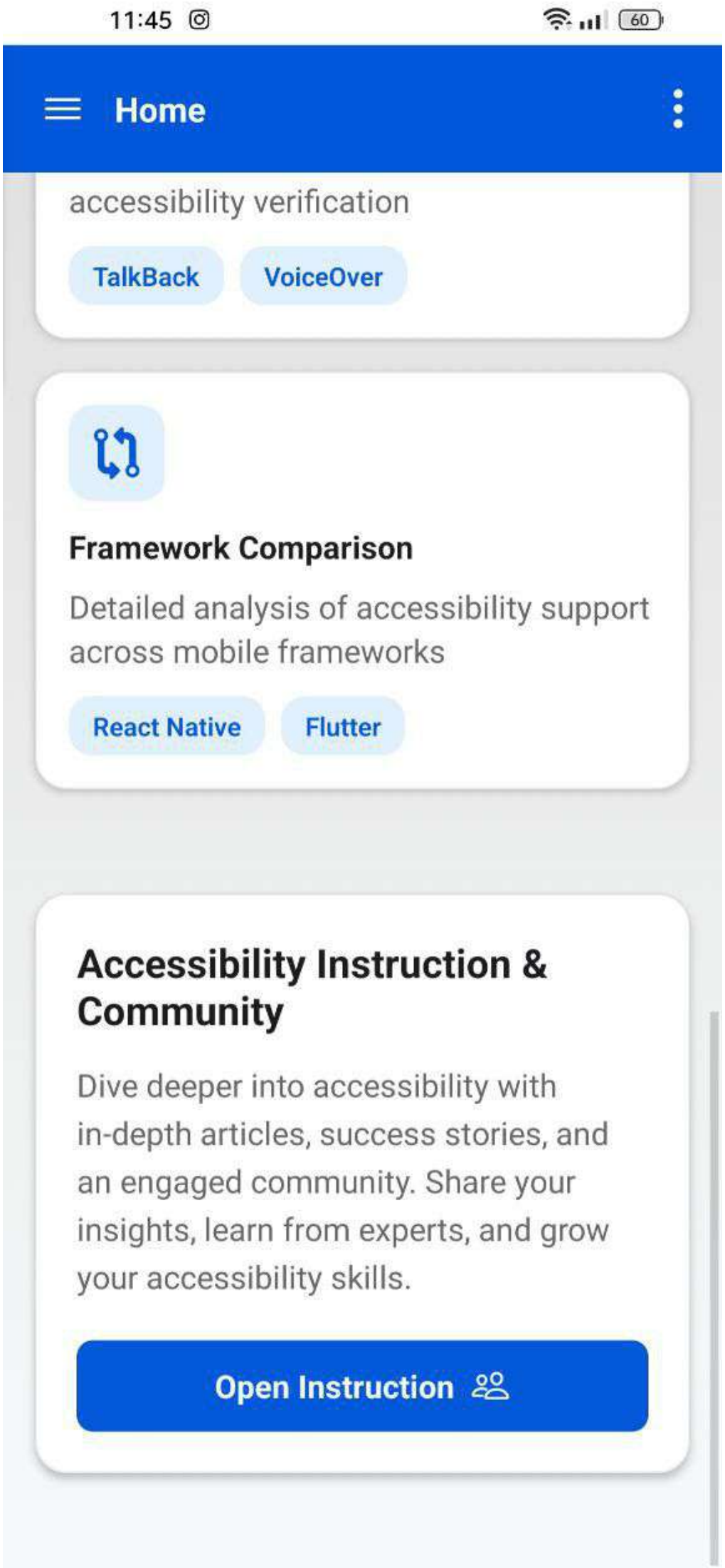
Analysis of conformance to Web Content Accessibility Guidelines (WCAG) 2.2, based on the methodology of Perinello & Gaggi (2024).

Criteria evaluated	43
Overall compliance	88%
Level A criteria implemented	25 / 26
Level AA criteria implemented	13 / 17

Compliance by Principle

1. Perceivable	11 / 13
2. Operable	16 / 17
3. Understandable	8 / 10

Development Resources


11:45
60
Home
accessibility verification
TalkBack
VoiceOver
Framework Comparison
Detailed analysis of accessibility support across mobile frameworks
React Native
Flutter
Accessibility Instruction & Community
Dive deeper into accessibility with in-depth articles, success stories, and an engaged community. Share your insights, learn from experts, and grow your accessibility skills.
Open Instruction

11:46

Home

The ultimate

Component Accessibility

Overview | Details | Methodology | References

UI Components

Assessment of accessibility implementation in React Native components used throughout the application.

Total components	20
Fully accessible	18 / 20
Partially accessible	2
With accessibility props	18
Identified accessibility issues	2

Development Resources

11:45

Home

The ultimate accessibility-driven toolkit for developers

A comprehensive resource for building inclusive React Native applications with verified accessibility standards – explore for more!

20	88%	85%
Compon ents	WCAG 2.2	Screen Reader
Ready to Use	Level AA	Test Coverage

Quick Start

Explore accessible component examples

Development Resources

11:46

Components > Button

Buttons & Touchables - Interactive Example

Learn how to implement an accessible, properly labeled button with minimal touch target and role/hint.

Button Demo

Button pressed successfully

Code Implementation

JSX Copy

```
<TouchableOpacity
  accessibilityRole="button"
  accessibilityLabel="Submit form"
  accessibilityHint="Activates form
submission"
  style={{
    minHeight: 44,
    paddingHorizontal: 16,
    backgroundColor:
```

11:46

Components > Button

Buttons & Touchables - Interactive Example

Learn how to implement an accessible, properly labeled button with minimal touch target and role/hint.

Button Demo

Submit

Code Implementation

JSX Copy

```
<TouchableOpacity
  accessibilityRole="button"
  accessibilityLabel="Submit form"
  accessibilityHint="Activates form
submission"
  style={{
    minHeight: 44,
    paddingHorizontal: 16,
    backgroundColor:
```

11:46

Home > Settings

High Contrast Mode
Increase contrast for improved readability

READABILITY ENHANCEMENTS

Large Text
Increase text size for improved readability

Reduce Motion
Minimize animations for improved comfort

COLOR & TOUCH SETTINGS

Color Filter
Apply basic grayscale filtering

Large Touch Targets
Increase the tappable area of interactive elements

11:46

Home > Components

Media Content Advanced

Make images and media content accessible

Aa Alt text ▷ Media controls

Modal Dialogs Advanced

Implement accessible modal dialogs with proper focus management and screen reader support

Focus trapping

Screen reader alerts

Loading & Navigation Beta

Explore advanced patterns such as Tabs/Carousels, Progress Indicators, Alerts/Toasts, and Sliders.

Tabs & Carousels

Progress Indicators

Alerts & Toasts

Sliders & Range

Code Implementation

JSX ✓ Copied!

```
<TouchableOpacity
  accessibilityRole="button"
  accessibilityLabel="Submit form"
  accessibilityHint="Activates form
submission"
  style={{
    minHeight: 44,
    paddingHorizontal: 16,
    backgroundColor:
colors.primary,
    borderRadius: 8,
    justifyContent: 'center',
    alignItems: 'center',
  }}
>
  <Text style={{ color:
isDarkMode ? colors.surface :
colors.background }}>
    Submit
  </Text>
</TouchableOpacity>
```

Accessibility Features

Minimum Touch Target

44x44 points ensures the button is

11:46

Home > Components

Accessibility Components

Interactive examples of accessible React Native components with code samples and best practices

Buttons & Touchables

Essential

Create accessible touch targets with proper sizing and feedback

Touch target sizing

Haptic feedback

Form Controls

Complex

Implement accessible form inputs and controls

Error states

Helper text

Media Content

Advanced

Make images and media content accessible

11:46

Home

Component Accessibility

Overview Details Methodology References

Evaluation Methodology

This accessibility evaluation follows a formalized approach based on framework analysis, WCAG mapping, and empirical testing with screen readers.

Version
1.0.0

Last Updated
2025-03-13

Approach
Combined static analysis and empirical testing

WCAG Version
2.2

Conformance Target
Level AA

Weighting System

Development Resources

11:46

Home

Component Accessibility

Overview Details Methodology

References

Bibliographic References

This evaluation is based on established accessibility standards and research in mobile application accessibility.

Web Content Accessibility Guidelines (WCAG) 2.2

W3C

Standard, 2023

The official W3C standard for web content accessibility, defining success criteria and conformance requirements.

Accessibility of Mobile User Interfaces using Flutter and React Native

Lorenzo Perinello, Ombretta Gaggi

IEEE 21st Consumer Communications & Networking Conference (CCNC), 2024 (doi: 10.1109/CCNC51664.2024.10454681)

Comparative analysis of accessibility implementation in Flutter and React Native, with insights on the developer experience.

Development Resources

11:46

Home

Component Accessibility

Overview | Details | Methodology | References

Components by Type

Basic UI Components	8
Complex Components	12

Accessibility Properties

accessibilityLabel	16 components
accessibilityRole	14 components
accessibilityHint	10 components
accessibilityState	8 components

Component Distribution

Components are distributed across multiple screens in the application, with core accessibility features implemented consistently.

Development Resources

Accessibility Features

Input Labels

Clear, descriptive labels that properly associate with form controls

Semantic Roles

Proper role assignments for form controls (radio, checkbox, button)

Error States

Clear error messages and validation feedback for screen readers

Touch Targets

Adequate sizing for interactive elements (minimum 44x44 points)

State Management

Proper announcements for selection controls and submit button

Date/Time Pickers

Integration with native pickers, with announced changes for screen readers

encourages good accessibility practices.

testing automation integration

Contribute

Learning Resources

Official Documentation

The comprehensive guide to implementing accessibility features in React Native apps directly from the source.

Community Guidelines

Official community-driven guidelines for implementing and testing accessible React Native applications.

Inspiration Examples

Complex UI Focus Management