iOS Testing - Known issues and possible solutions

Documentation

Overview

This document outlines accessibility inconsistencies discovered during iOS testing of the React Native accessibility toolkit application. While the application maintains 86% WCAG 2.2 Level AAA compliance and functions perfectly on Android with TalkBack, several iOS VoiceOver-specific issues were identified that require targeted solutions.

Testing Environment

- **iOS Version**: iOS 18.5
- **Screen Reader**: VoiceOver
- **Test Devices**: iPhone XR, VoiceOver enabled
- **Comparison Baseline**: Android 14 and 15 with TalkBack on Pixel 7 (100% functional)

Identified Issues

1. Copy Code Functionality Inconsistency

Problem: The copy code functionality works correctly in Dialog and Form components but fails to be accessible in Button, Media, and Advanced components, where VoiceOver reads the entire code block as a single unit.

Root Cause: VoiceOver's stricter accessibility container hierarchy prevents proper focus management on nested TouchableOpacity elements within certain component types.

Solution:

```
// Fix for copy button accessibility on iOS
const CodeBlock = ({ children, language }) => {
 return (
  <View style={styles.codeContainer}>
   <View style={styles.codeHeader}>
    <Text style={styles.languageLabel}>{language}</Text>
    <TouchableOpacity
     style={styles.copyButton}
     onPress={handleCopy}
     accessible={true}
     accessibilityRole="button"
     accessibilityLabel="Copy code to clipboard"
     accessibilityHint="Copies the code snippet to your clipboard"
     // Critical for iOS VoiceOver
     accessibilityElementsHidden={false}
     importantForAccessibility="yes"
    >
     <Text style={styles.copyText}>Copy</Text>
    </TouchableOpacity>
   </View>
   <ScrollView
    style={styles.codeContent}
    accessible={true}
    accessibilityRole="text"
    accessibilityLabel="Code snippet"
    // Prevent interference with copy button
    accessibilityElementsHidden={false}
   >
    <Text style={styles.codeText}>{children}</Text>
   </ScrollView>
  </View>
 );
};
```

2. Home Navigation Focus Order Issue

Problem: VoiceOver navigation follows an incorrect sequence (first, third, second) instead of the expected linear order, disrupting the user experience.

Root Cause: VoiceOver uses spatial-geometric traversal combined with accessibility hierarchy, while TalkBack follows a more predictable linear approach based on layout order.

Solution:

```
// Fix navigation order in index.tsx
import { Platform } from 'react-native';

const HomeScreen = () => {
  return (
    <View style={styles.container}>
      <View
        style={styles.statsContainer}
        // Force linear ordering on iOS
        accessible={Platform.OS === 'ios'}
        accessibilityRole={Platform.OS === 'ios' ? 'summary' : undefined}
      >
        {/* First stat */}
        <View
          style={[styles.statItem, { accessibilityOrder: 1 }]}
          accessible={true}
          accessibilityRole="text"
          accessibilityLabel="20 Components Ready to Use"
        >
          <Text style={styles.statNumber}>20</Text>
          <Text style={styles.statLabel}>Components</Text>
        </View>

        {/* Second stat - Center */}
        <View
          style={[styles.statItem, styles.centerStat, { accessibilityOrder: 2 }]}
          accessible={true}
```

```
        accessibilityRole="text"
        accessibilityLabel="86% WCAG 2.2 Level AAA Compliance"
      >
        <Text style={styles.statNumber}>86%</Text>
        <Text style={styles.statLabel}>WCAG 2.2</Text>
      </View>

      {/* Third stat */}
      <View
        style={[styles.statItem, { accessibilityOrder: 3 }]}
        accessible={true}
        accessibilityRole="text"
        accessibilityLabel="85% Screen Reader Test Coverage"
      >
        <Text style={styles.statNumber}>85%</Text>
        <Text style={styles.statLabel}>Screen Reader</Text>
      </View>
    </View>
  </View>
 );
};

const styles = StyleSheet.create({
 statsContainer: {
  flexDirection: 'row',
  justifyContent: 'space-around',
  alignItems: 'center',
  // iOS VoiceOver needs consistent positioning
  ...(Platform.OS === 'ios' && {
   alignItems: 'flex-start',
  }),
 },
 centerStat: {
  // Ensure center stat doesn't interfere with focus order
  zIndex: Platform.OS === 'ios' ? 1 : 0,
```

```
  },
});
```

3. Modal Content Accessibility Issue

Problem: In modal dialogs, the Overview, Methodology, and other tab sections are not individually selectable. VoiceOver reads them as a single block instead of allowing navigation to individual elements.

Root Cause: VoiceOver's aggressive modal focus trapping and stricter accessibility container boundaries prevent proper child element discovery.

Solution:

```
// Fix for modal accessibility on iOS
import { Platform, AccessibilityInfo } from 'react-native';

const ScreenReaderModal = ({ visible, onClose }) => {
  const [activeTab, setActiveTab] = useState('overview');

  useEffect(() => {
    if (visible && Platform.OS === 'ios') {
      // Announce modal opening to VoiceOver
      AccessibilityInfo.announceForAccessibility('Screen Reader Testing modal
opened');
    }
  }, [visible]);

  const handleTabPress = (tabName) => {
    setActiveTab(tabName);

    // iOS VoiceOver needs explicit focus management
    if (Platform.OS === 'ios') {
      AccessibilityInfo.announceForAccessibility(`${tabName} tab selected`);
    }
  };
```

```
  return (
    <Modal
      visible={visible}
      transparent
      animationType="slide"
      accessibilityViewIsModal={true}
      onRequestClose={onClose}
    >
      <View style={styles.modalContainer}>
        <View
          style={styles.modalContent}
          accessible={false} // Let children handle their own accessibility
          accessibilityRole="dialog"
          accessibilityLabel="Screen Reader Testing Details"
        >
          {/* Header */}
          <View style={styles.modalHeader}>
            <Text
              style={styles.modalTitle}
              accessible={true}
              accessibilityRole="header"
              accessibilityLevel={1}
            >
              Screen Reader Testing
            </Text>
            <TouchableOpacity
              style={styles.closeButton}
              onPress={onClose}
              accessible={true}
              accessibilityRole="button"
              accessibilityLabel="Close modal"
              accessibilityHint="Closes the screen reader testing modal"
            >
              <Text style={styles.closeText}>×</Text>
            </TouchableOpacity>
          </View>
```

```
        {/* Tab Navigation */}
        <View
          style={styles.tabContainer}
          accessible={false} // Let individual tabs be accessible
          accessibilityRole="tablist"
        >
          {['Overview', 'Details', 'Methodology', 'References'].map((tab, index) =>
(
            <TouchableOpacity
              key={tab}
              style={[
                styles.tab,
                activeTab === tab.toLowerCase() && styles.activeTab
              ]}
              onPress={() => handleTabPress(tab.toLowerCase())}
              accessible={true}
              accessibilityRole="tab"
              accessibilityLabel={`${tab} tab`}
              accessibilityHint={`Shows ${tab.toLowerCase()} information`}
              accessibilityState={{
                selected: activeTab === tab.toLowerCase()
              }}
              // iOS specific: ensure proper focus order
              accessibilityElementsHidden={false}
              importantForAccessibility="yes"
            >
              <Text style={[
                styles.tabText,
                activeTab === tab.toLowerCase() && styles.activeTabText
              ]}>
                {tab}
              </Text>
            </TouchableOpacity>
          ))}
        </View>
```

```
        {/* Tab Content */}
        <ScrollView
          style={styles.tabContent}
          accessible={true}
          accessibilityRole="tabpanel"
          accessibilityLabel={`${activeTab} content`}
          // iOS: Enable individual element selection
          accessibilityElementsHidden={false}
        >
          {renderTabContent(activeTab)}
        </ScrollView>
      </View>
    </View>
  </Modal>
 );
};

const renderTabContent = (activeTab) => {
 switch (activeTab) {
  case 'overview':
   return (
    <View accessible={false}>
     <Text
      style={styles.sectionTitle}
      accessible={true}
      accessibilityRole="header"
      accessibilityLevel={2}
     >
      Screen Reader Testing
     </Text>
     <Text
      style={styles.sectionText}
      accessible={true}
      accessibilityRole="text"
     >
```

```
          Results from empirical testing with VoiceOver (iOS) and TalkBack (Andr
oid) screen readers, evaluating real-world accessibility.
        </Text>
      </View>
    );
  case 'methodology':
    return (
      <View accessible={false}>
        <Text
          style={styles.sectionTitle}
          accessible={true}
          accessibilityRole="header"
          accessibilityLevel={2}
        >
          Testing Methodology
        </Text>
        <Text
          style={styles.sectionText}
          accessible={true}
          accessibilityRole="text"
        >
          Systematic evaluation using standardized accessibility testing protocol
s.
        </Text>
      </View>
    );
  // Add other cases as needed
 }
};
```

Platform-Specific Accessibility Differences

VoiceOver vs TalkBack Behavioral Differences

Aspect	TalkBack (Android)	VoiceOver (iOS)
Focus Management	Linear, predictable traversal	Spatial-geometric with hierarchy priority
Container Handling	Permissive with nested elements	Stricter accessibility boundaries
Modal Behavior	Lenient focus trapping	Aggressive modal focus management
Element Discovery	More forgiving of hierarchy issues	Requires explicit accessibility guidance

General iOS Accessibility Guidelines

1. Explicit Accessibility Properties

Always use explicit accessibility properties for iOS:

```
// Good for iOS
<TouchableOpacity
  accessible={true}
  accessibilityRole="button"
  accessibilityLabel="Clear description"
  accessibilityHint="Action that will be performed"
  accessibilityElementsHidden={false}
  importantForAccessibility="yes"
>
```

2. Focus Management

Implement explicit focus management for state changes:

```
// iOS focus management
useEffect(() => {
  if (Platform.OS === 'ios' && shouldAnnounceFocus) {
    AccessibilityInfo.announceForAccessibility('State changed');
```

```
  }
}, [stateVariable]);
```

3. Container Accessibility

Use `accessible={false}` on containers to let children handle their own accessibility:

```
// Container pattern for iOS
<View accessible={false}>
  <Text accessible={true} accessibilityRole="header">Title</Text>
  <Text accessible={true} accessibilityRole="text">Content</Text>
</View>
```

Conclusion

While the application maintains excellent accessibility standards overall, these iOS-specific issues highlight the importance of platform-specific testing and implementation strategies. The solutions provided address the core architectural differences between VoiceOver and TalkBack, ensuring consistent accessibility across both platforms.

The key insight is that **VoiceOver requires more explicit accessibility guidance** than TalkBack, which is generally more forgiving of accessibility hierarchy inconsistencies. Future development should account for these platform differences from the initial design phase.