Creating Accessible React Native Apps

Insight into testing and implementing accessibility best practices for React Native apps

shopify

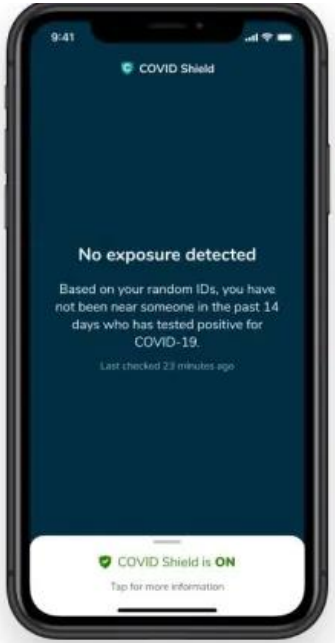
Scott Vinkle…

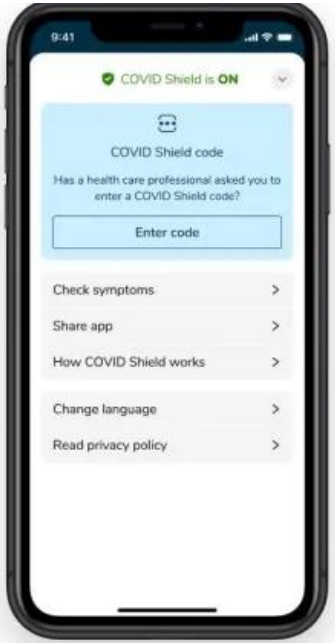
- Platform Accessibility Specialist
- @svinkle, ScottVinkle.me
- , , , ,

React Native is the Future of Mobile at Shopify

by Farhan Thawar • Mobile
Jan 29, 2020 • 10 minute read

COVID Shield





Download COVID Alert today

COVID Alert is Canada's free COVID-19 exposure notification app. It can alert you to possible exposures before you have symptoms.

Available for iOS and Android

Download on the App Store

GET IT ON Google Play

On this page

Agenda

1. **What?** …is digital accessibility
2. **Who?** People with disabilities, understanding disability, assistive technology
3. **How?** Testing with mobile screen readers, testing with simulators, what to watch for
4. **React Native specifics**
    a. Documentation
    b. Roles
    c. Labels
    d. States
    e. …and more!
5. **Questions?**

What is Digital Accessibility?

1.85 B

“With an estimated population of 1.85 billion, people with disabilities (PWD) are an emerging market larger than China”

$13 T

“The Disability Market influences over $13 trillion in annual disposable income.”

Design Delight from Disability - 2020 Annual Report: The Global Economics of Disability

Who requires Digital Accessibility?

Types of disabilities

- Visual
- Hearing
- Cognitive
- Motor
- And may more and/or a combination

Disability is not binary

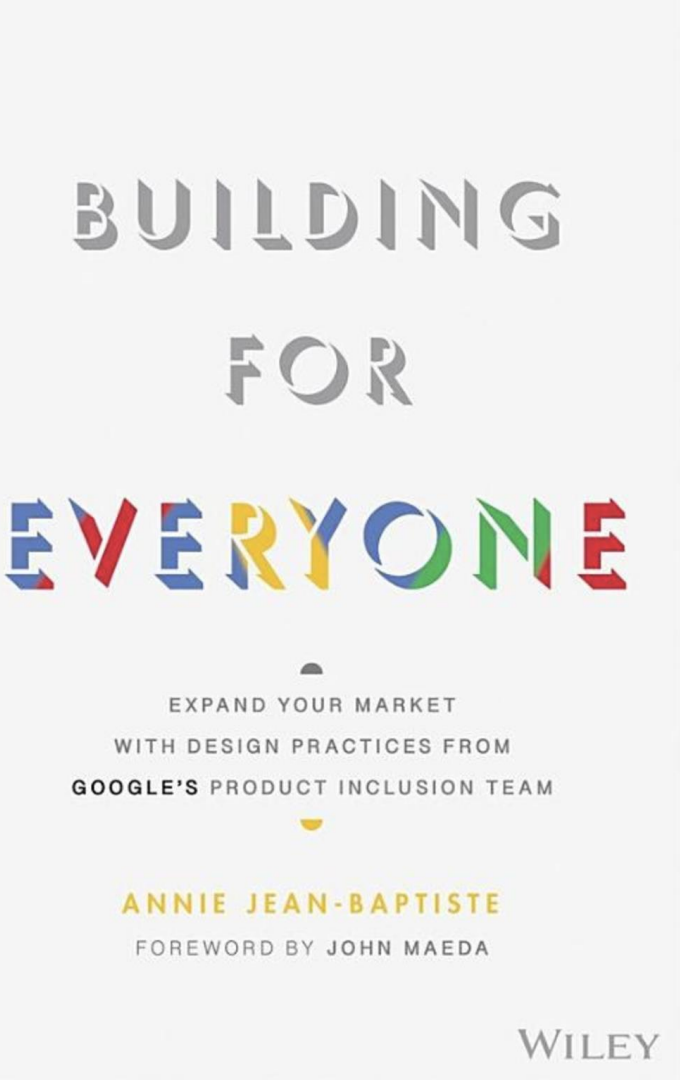
How do people with disabilities use technology?

Assistive technologies

- Screen readers
- Keyboard-only
- Screen magnification
- Voice dictation
- Many, many more

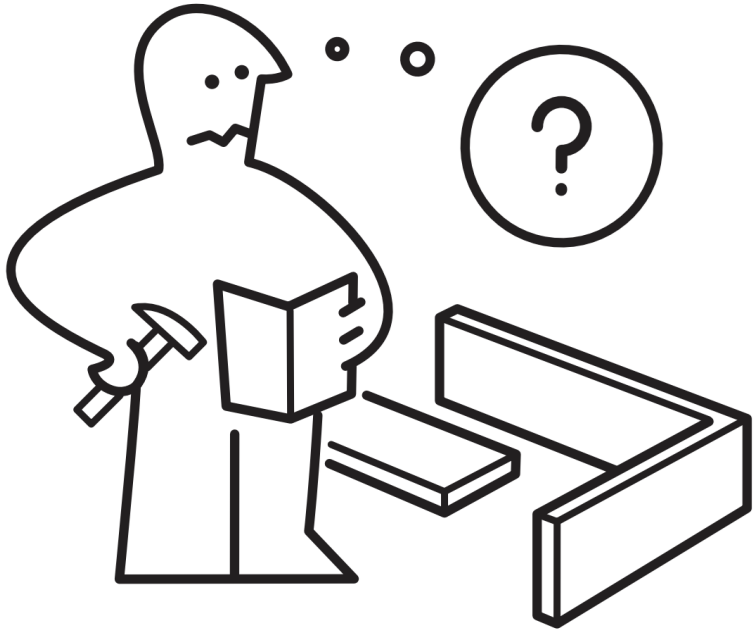
“We need to build for everyone, with everyone, not only because it is the right thing to do, but also because it drives innovation and growth while making the world a better, richer place.”

– Annie Jean-Baptiste



How do we make things accessible?


TEST ALL THE THINGS



1. What is this *thing*?

- Context must be shared to understand what *the thing* is
  - Role (ex., button)
  - Name (ex., “Submit”)
  - State (ex., disabled)

The accessible name/label/text equivalent

```
<a href="contact.html">Contact Us</a>
```

Contact Us

Contact Us

The element role

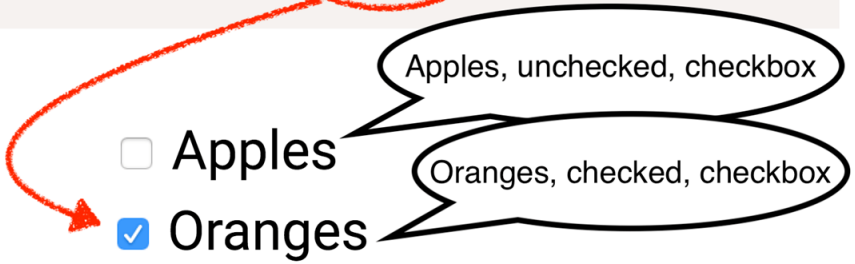
```
<a href="contact.html">Contact Us</a>
```

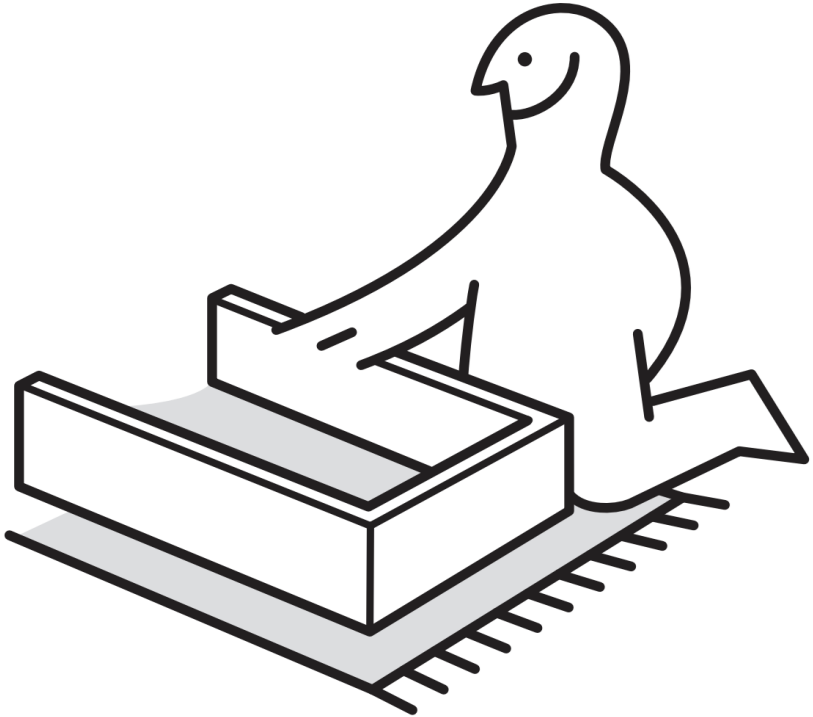
Contact Us, link

Contact Us

The current state

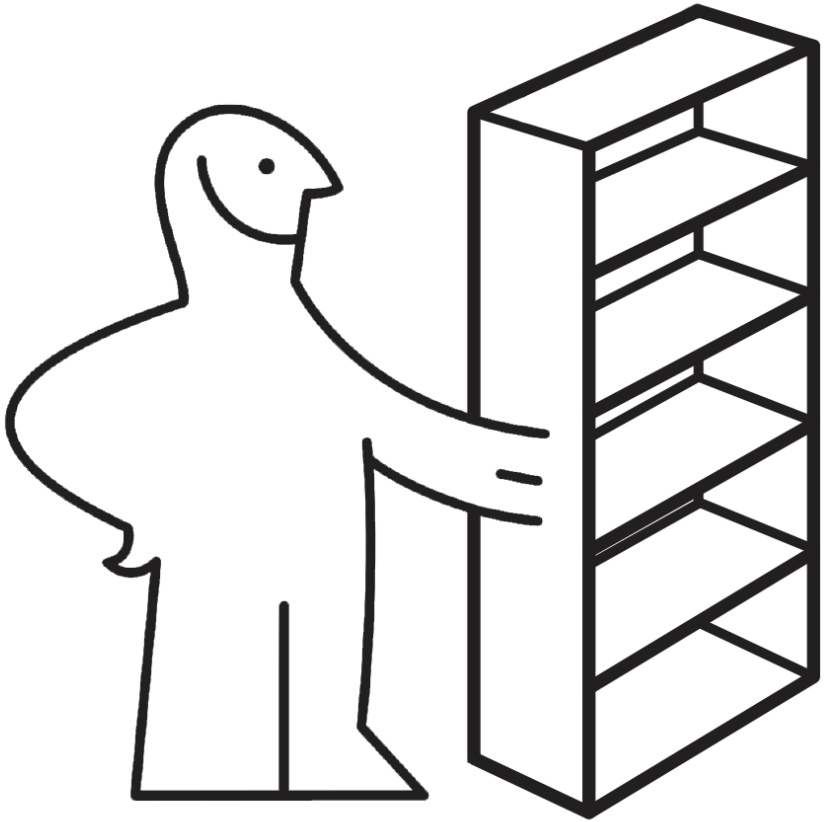
```
<label><input type="checkbox"> Apples</label>
<label><input type="checkbox" checked> Oranges</label>
```





2. What happens when I click *the thing*?

- Affordances are based on the visual and aural experience
- The semantics of the control in question, as well as the visual affordance, indicate to the user what might happen on click



3. Did clicking *the thing* meet my expectations?

- Did the interaction result in what the user had in mind
- Was the user “successful” or not
- Avoid having the user question app quality and self doubt – guide the user in being successful

1. Understanding what the thing is

2. Knowing what’s expected when the thing is clicked, and

3. Having user expectations met as a result.

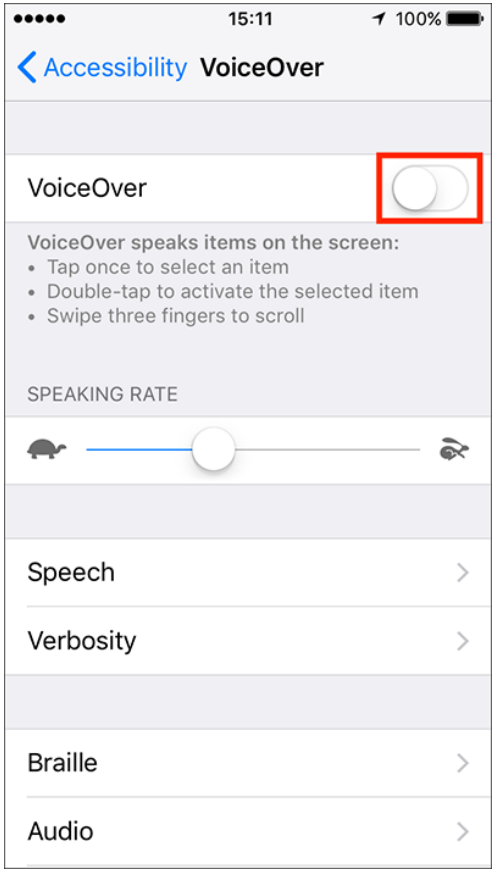
Testing with Mobile Screen Readers

Before you start…

Navigation methods:

1. **Explore:** Single finger, drag to find and content.
2. **Swipe:** Single finger, swipe gesture left or right to find content

When a control is in focus, **double tap** anywhere **to activate**.



iOS – VoiceOver

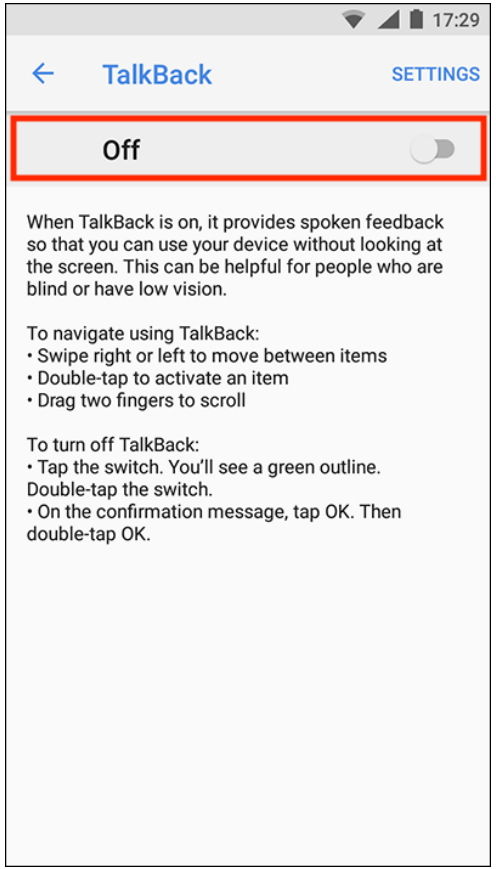
Enable VoiceOver:

1. Settings
2. Accessibility
3. VoiceOver

Shortcut:

1. Settings
2. General
3. Accessibility
4. Accessibility Shortcut

Triple-press the Home button.



Android – TalkBack

Enable TalkBack:

1. Settings
2. Accessibility
3. TalkBack

Shortcut:

1. Settings
2. Accessibility
3. TalkBack
4. TalkBack Shortcut

Press and hold both volume buttons.

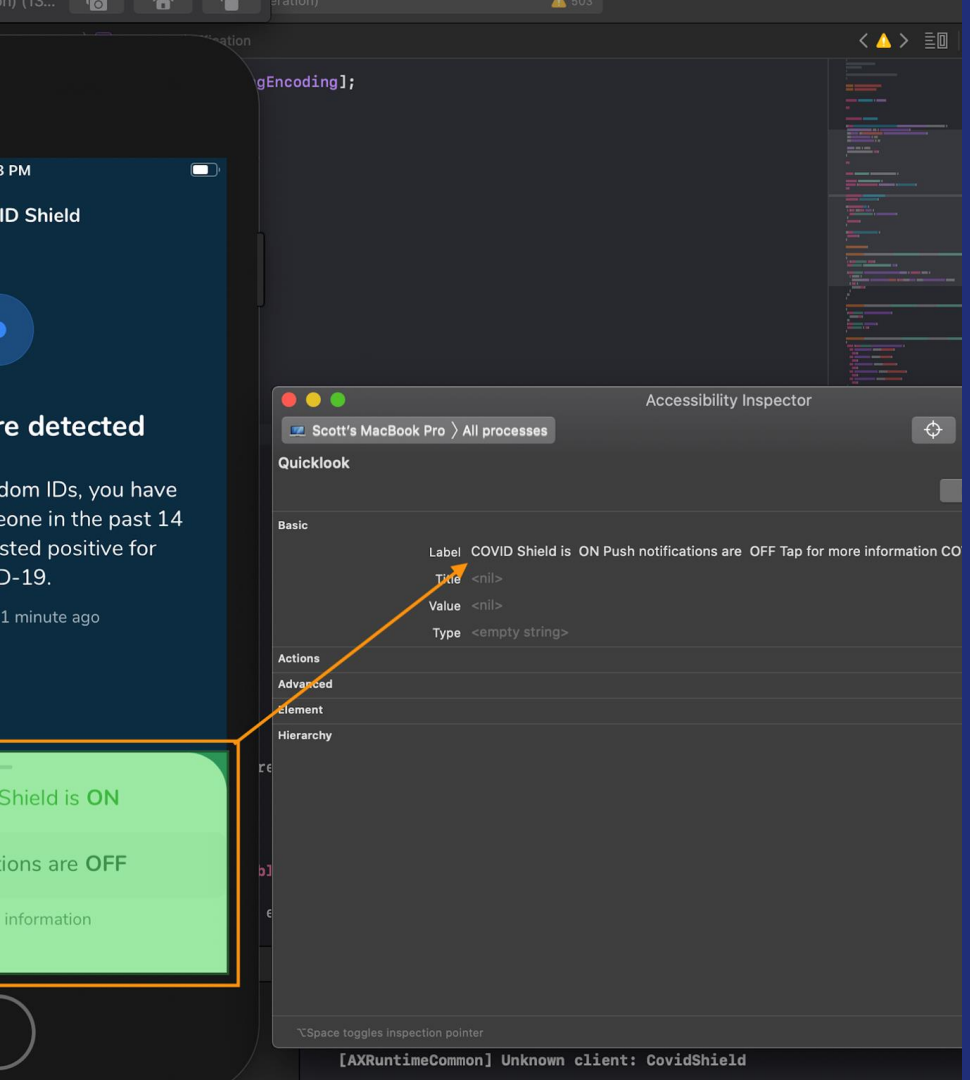
VoiceOver Common Gestures

Action	Gesture
Select/read the item	Touch/single tap
Activate the currently selected item	Double-tap
Move to the next item	Swipe-right
Move to the previous item	Swipe-left
Pause/resume reading	Two-finger tap
Scroll up	Three-finger swipe up
Scroll down	Three-finger swipe down

TalkBack Common Gestures

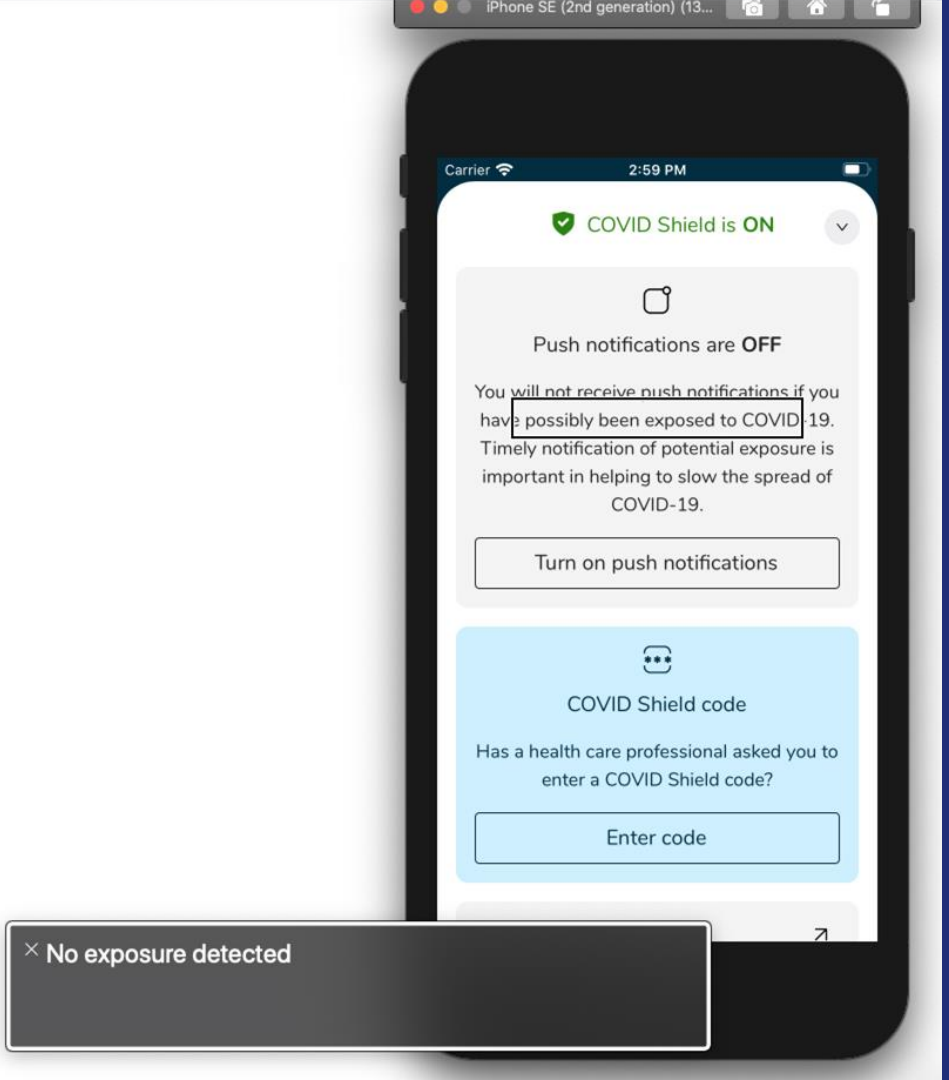
Action	Gesture
Select/read the item	Touch/single tap
Activate the currently selected item	Double-tap
Move to the next item	Swipe-right
Move to the previous item	Swipe-left
Scroll up	Two-finger slide up
Scroll down	Two-finger slide down

Testing with Simulators



macOS Accessibility Inspector

- Inspect the UI like browser dev tools
- Displays how your app “sounds” while using a screen reader
- Open via Spotlight Search and type “Accessibility Inspector”
- Click crosshair icon button then hover over the UI to be tested

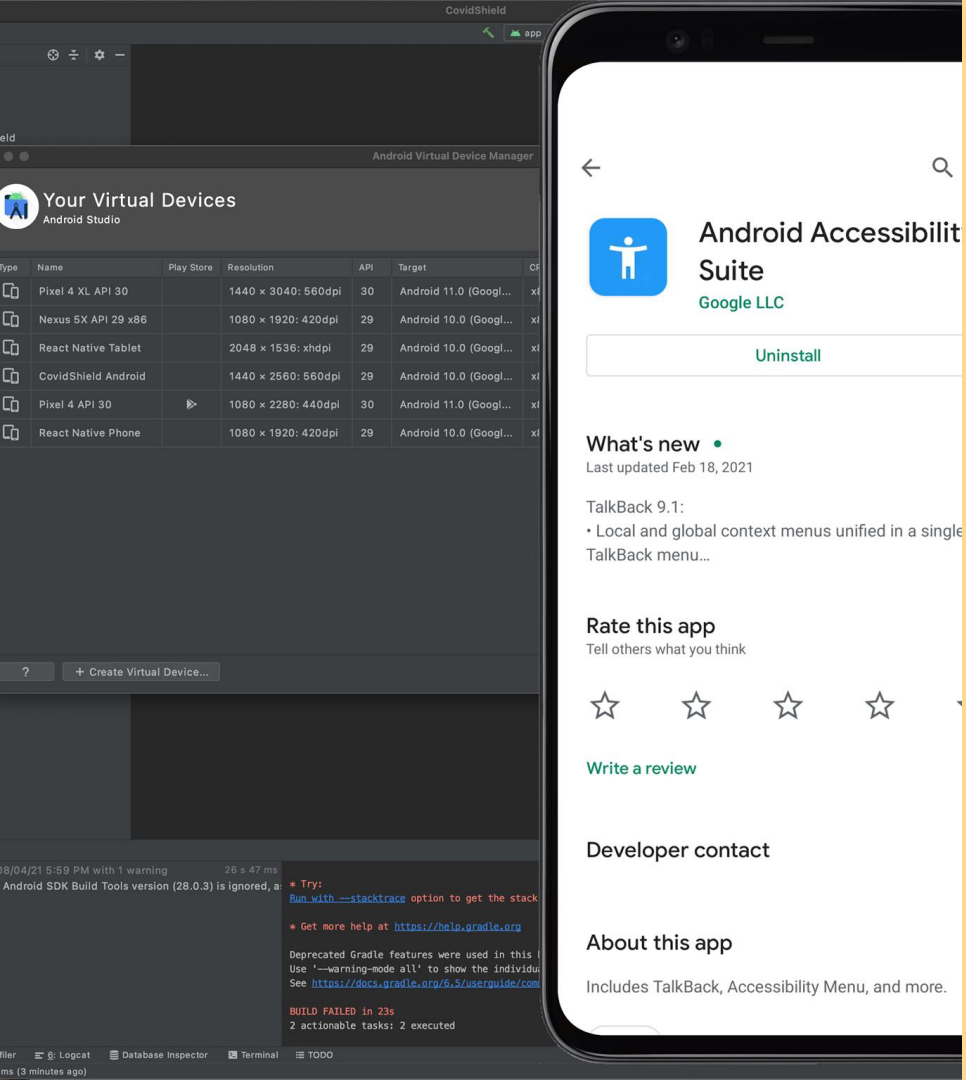


iOS Simulator VoiceOver

1. Focus on simulator window
2. Start (and stop) VoiceOver with **Cmd + F5**
3. Press **Ctrl + Opt** then Left or Right to move around
4. Press **Ctrl + Opt** then Space to interact

Gestures

- Pinch-zoom/swipe: Hold the **Opt** key and drag the mouse



Android Emulator TalkBack

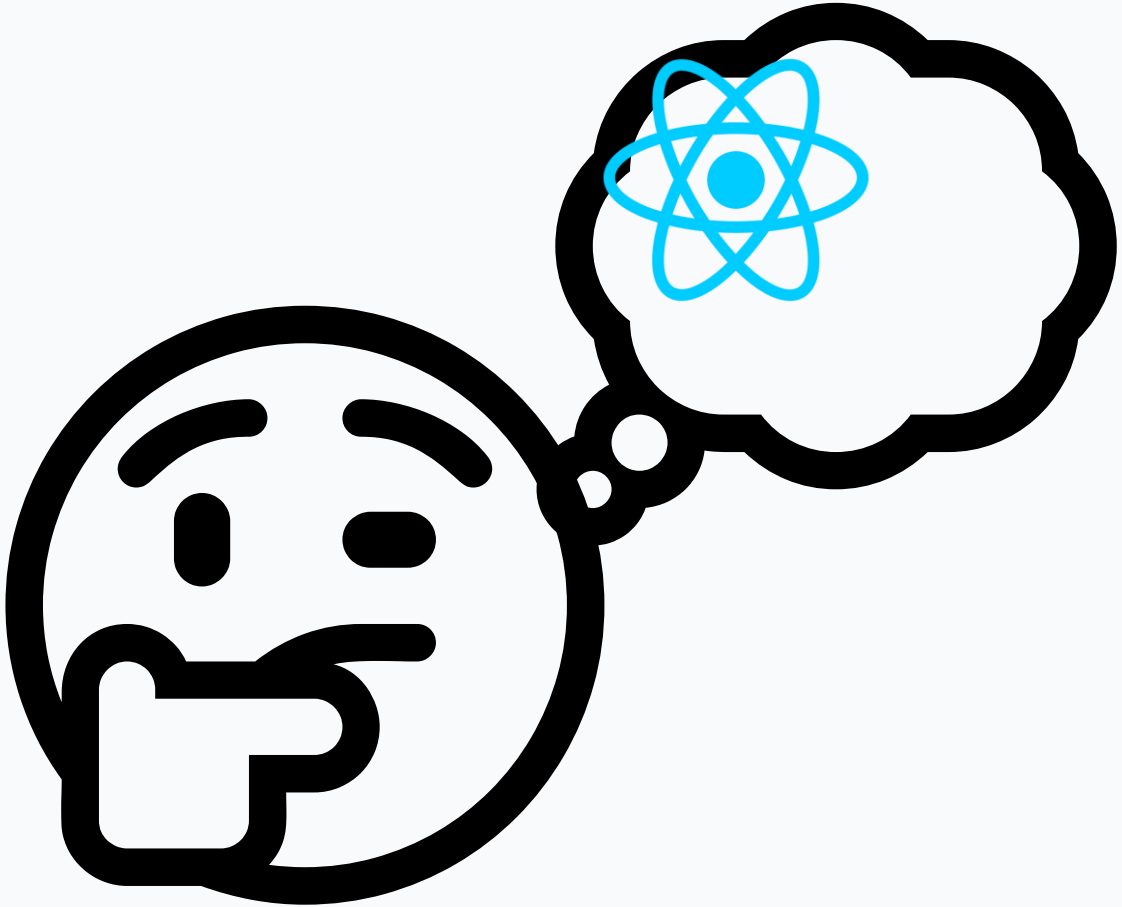
1. Install **Android Accessibility Suite**
2. Load your app
3. **Enable TalkBack**: Settings → Accessibility → TalkBack

Gestures

- Pinch-zoom/swipe: Hold the **Opt** key and drag the mouse
- Mouse click and drag to “swipe”

Tip!

- Use device image with Play Store!



Accessibility

Both Android and iOS provide APIs for integrating apps with assistive technologies like the bundled screen readers VoiceOver (iOS) and TalkBack (Android). React Native has complementary APIs that let your app accommodate all users.

> Android and iOS differ slightly in their approaches, and thus the React Native implementations may vary by platform.

Accessibility properties

`accessible`

When `true`, indicates that the view is an accessibility element. When a view is an accessibility element, it groups its children into a single selectable component. By default, all touchable elements are accessible.

On Android, `accessible={true}` property for a react-native View will be translated into native `focusable={true}`.

```
<View accessible={true}>
  <Text>text one</Text>
  <Text>text two</Text>
</View>
```

In the above example, we can't get accessibility focus separately on 'text one' and 'text two'. Instead we get focus on a parent view with 'accessible' property.

`accessibilityLabel`

When a view is marked as accessible, it is a good practice to set an accessibilityLabel on the view, so that people who use VoiceOver know what element they have selected. VoiceOver will read this string when a user selects the associated element.

To use, set the `accessibilityLabel` property to a custom string on your View, Text or Touchable:

```
<TouchableOpacity
  accessible={true}
  accessibilityLabel="Tap me!"
  onPress={onPress}>
  <View style={styles.button}>
    <Text style={styles.buttonText}>Press me!</Text>
  </View>
</TouchableOpacity>
```

In the above example, the `accessibilityLabel` on the TouchableOpacity element would default to "Press me!". The label is constructed by concatenating all Text node children separated by spaces.

`accessibilityHint`

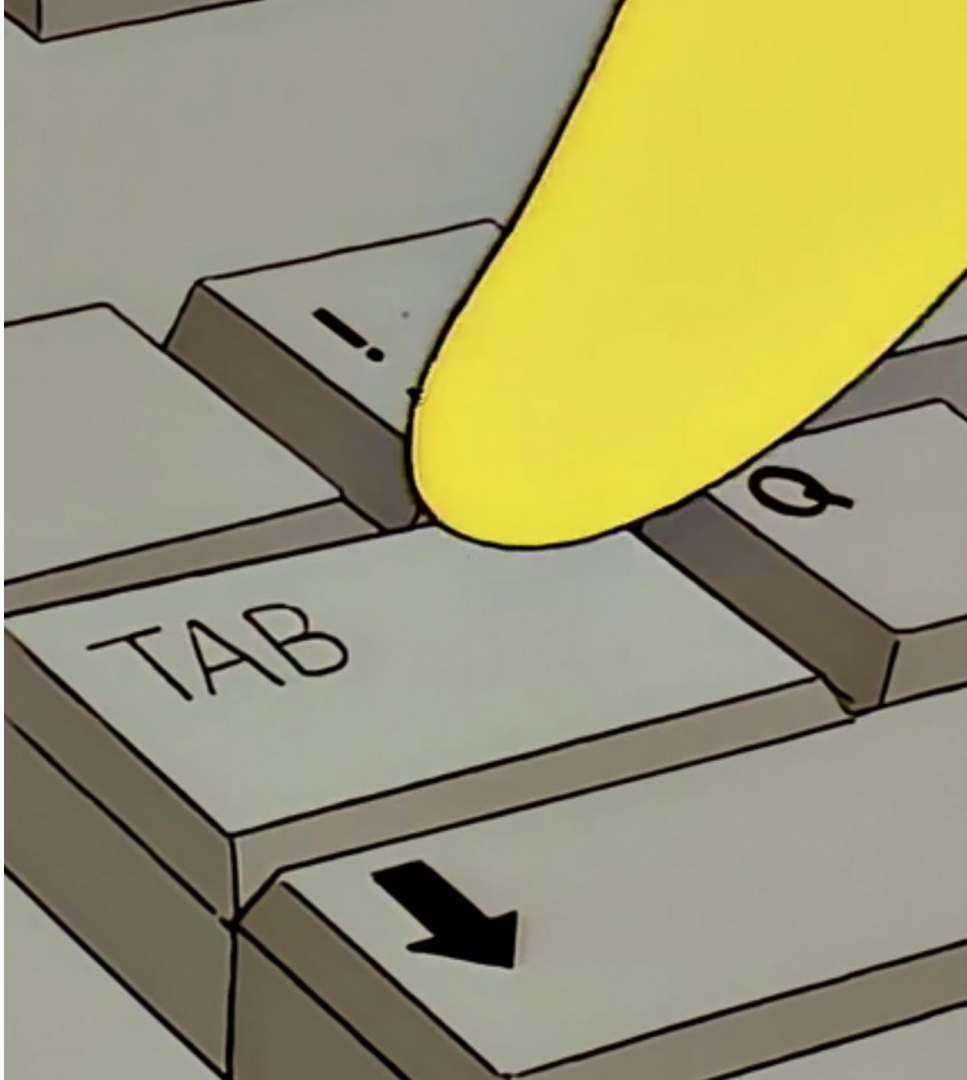
An accessibility hint helps users understand what will happen when they perform an action on the accessibility element when that result is not clear from the accessibility label.

ReactNative.dev/docs/accessibility

Adding semantics: role, name, state

Clickable things...

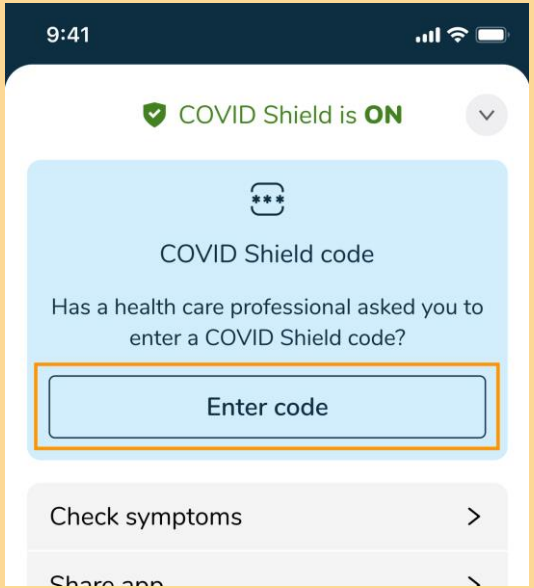
```
if (Platform.OS === 'android') {
  return (
    <Ripple
onPress={onPressHandler} ...>
      {content}
    </Ripple>
  );
}
return (
  <TouchableOpacity
onPress={onPressHandler} ...>
    {content}
  </TouchableOpacity>
);
```



Adding a role

- Apply the `accessibilityRole` prop to the clickable component
- Provide a string value appropriate for the context in question
- Value must be valid according to the API (`checkbox`, `radio`, etc)
- Equivalent to the `role` attribute in HTML

```
<TouchableOpacity
  accessibilityRole="button"
  …
>
  Enter code
</TouchableOpacity>
```



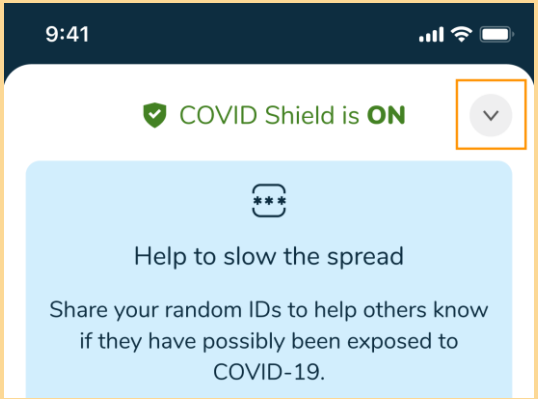
Before:
"Enter code"

After:
"Enter code, button"

Add a name

- Apply the `accessibilityLabel` prop to the clickable component
- Provide a string value describing the purpose of the control
- Value defined by the author
- Equivalent to the `aria-label` attribute in HTML

```
<TouchableOpacity
  accessibilityLabel="Close"
  accessibilityRole="button"
  …
>
  <!-- Icon… -->
</TouchableOpacity>
```



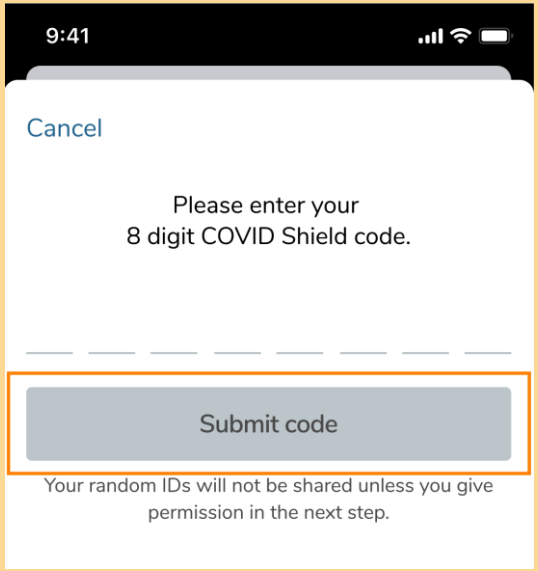
Before:
“…”

After:
“Close, button”

Add state

- Apply the `accessibilityState` prop to the clickable component
- Provide an object with boolean value for the state definition
- Object definition must be valid according to the API (`disabled`, `selected`, etc)
- Equivalent to [ARIA state attributes]() in HTML

```
<TouchableOpacity
  accessibilityRole="button"
  accessibilityState={disabled: true}
  …
>
  Submit code
</TouchableOpacity>
```



Before:
🗣 “Submit code”

After:
🗣 “Submit code, dimmed, button”

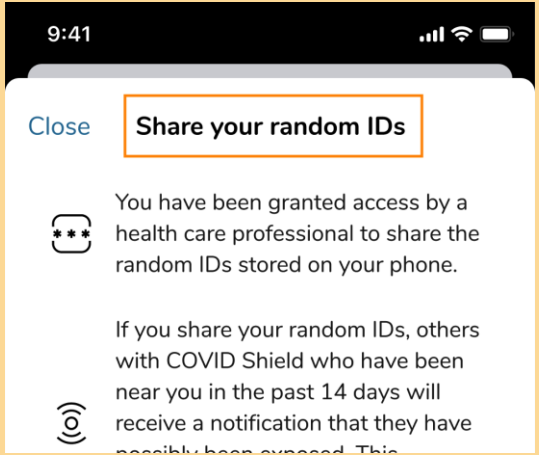
Headings

People who depend on assistive technology often navigate by headings first.

Adding a heading

- Apply the `accessibilityRole` prop to the component
- Provide the string value, “header”
- Heading level concept doesn’t exist
- Equivalent to the `role=”heading”` attribute in HTML

```
<Text
  …
  accessibilityRole="header"
>
  Share your random IDs
</Text>
```



Before:
“Share your random IDs”

After:
“Share your random IDs, heading”

Hint Text

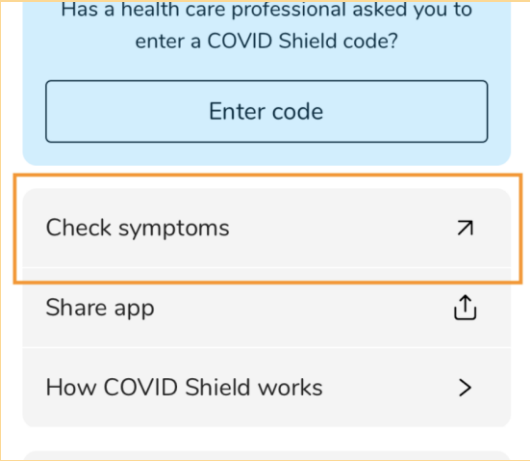
When a new browser tab/window/app opens on click, let the user know.

Give power to the user—let them decide how they'd like to proceed.

Including hint text

- Apply the `accessibilityHint` prop to the component
- Provide a string value with the hint text
- Value defined by the author
- Equivalent to the **upcoming** `aria-description` attribute in HTML

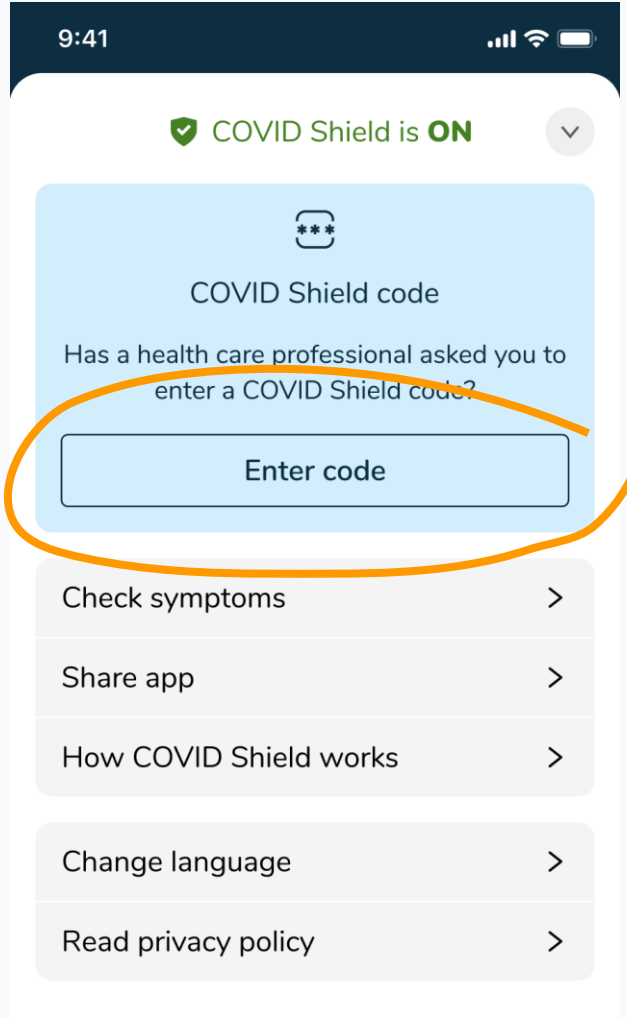
```
<TouchableOpacity
  …
  accessibilityHint="Opens in a new window"
  accessibilityRole="link"
>
  Check symptoms
</TouchableOpacity>
```

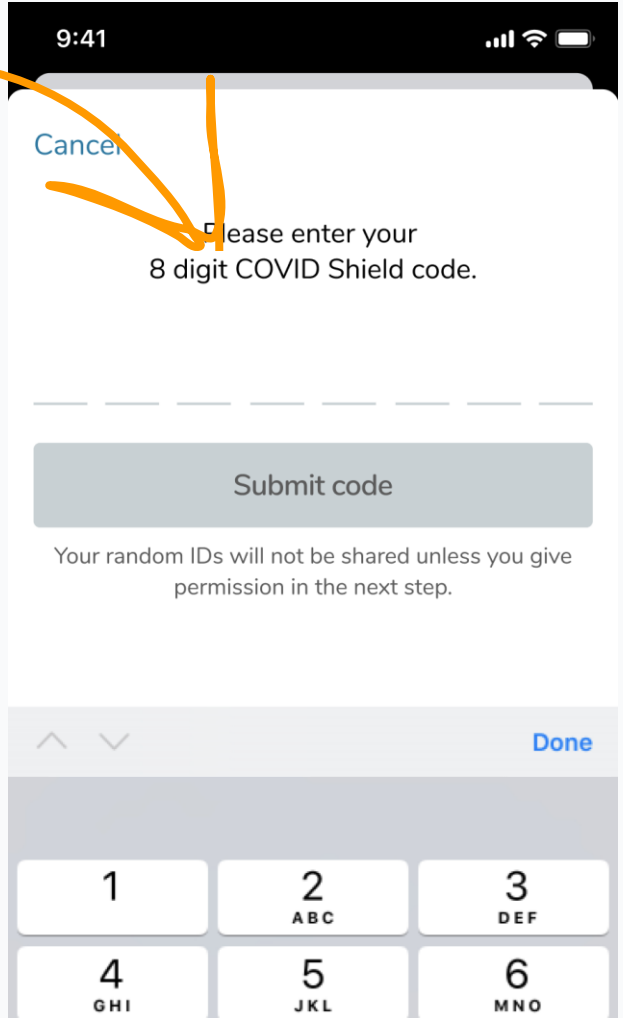


Before:
“Check symptoms”

After:
“Check symptoms, opens in a new window, link”

Focus Management


9:41
COVID Shield is ON
COVID Shield code
Has a health care professional asked you to enter a COVID Shield code?
Enter code
Check symptoms
Share app
How COVID Shield works
Change language
Read privacy policy


9:41
Cancel
Please enter your
8 digit COVID Shield code.
Submit code
Your random IDs will not be shared unless you give permission in the next step.
Done
1
2 ABC
3 DEF
4 GHI
5 JKL
6 MNO

What we learned from user testing of accessible client-side routing techniques with Fable Tech Labs

Marcy Sutton
July 11th, 2019

Lead DevRel/Community Software Engineer at Gatsby, cyclist, animal lover, pie baker.

Follow Marcy Sutton on Twitter

Tagged with accessibility, client-side-routing, cutting-edge-experiences, diversity-and-inclusion, react, user-testing View all Tags

In June 2019, I conducted 5 user testing sessions for accessibility research with Fable Tech Labs, a Toronto-based start-up that's "making it easier for digital teams to engage people with disabilities in product development."

The goal of this initiative was to gather feedback from users with disabilities on a set of prototypes with navigation techniques for JavaScript web apps. There are multiple variations recommended in the industry for accessible, client-rendered page changes, yet very little user research on those methods. Therefore, we wanted to find out **which techniques are the most ergonomic and intuitive to users with disabilities**, and **if any of the techniques presented barriers** detracting from their browsing experience.

With this data, we can make better recommendations for accessible client-rendered websites in general and at the JavaScript framework level. By adjusting Gatsby's implementation of client-side routing–currently leveraging @reach/router for React.js–to better support a range of people with disabilities, we can improve access for users and also potentially influence accessible UI patterns in other frameworks. Eventually (we hope), this work could encourage new browser APIs through web standards: "paving the accessible cowpaths" with solutions based in user research.

What is client-side routing?

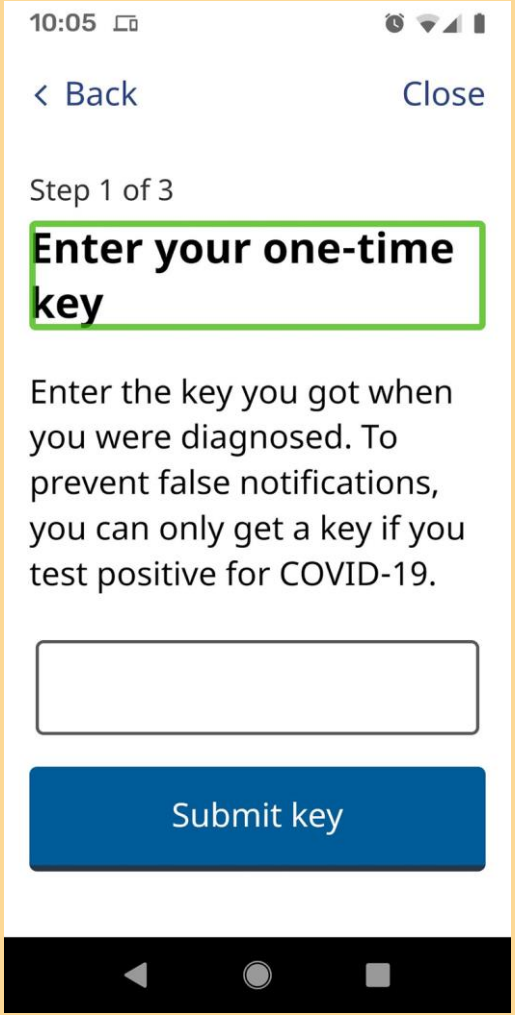
gatsbyjs.com/blog/2019-07-11-user-testing-accessible-client-routing

Shift focus to a heading

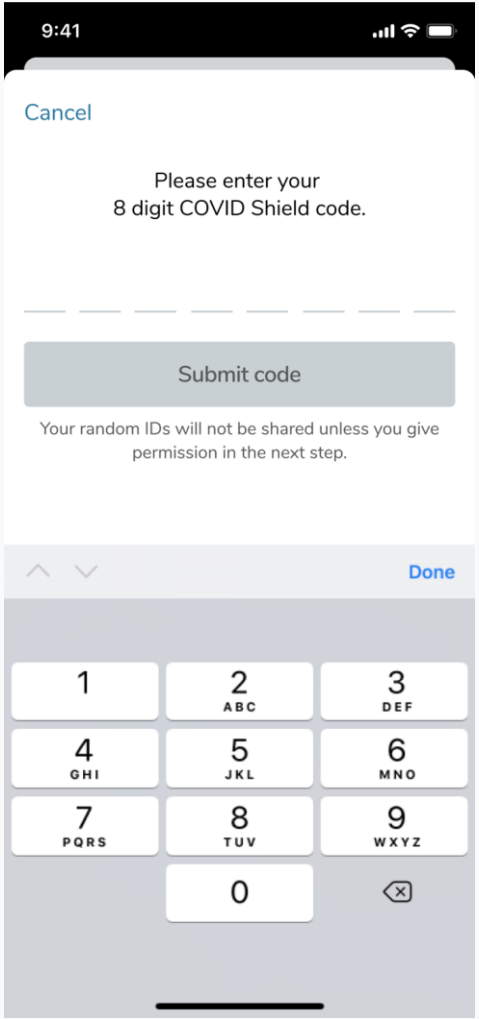
Focusing on a heading

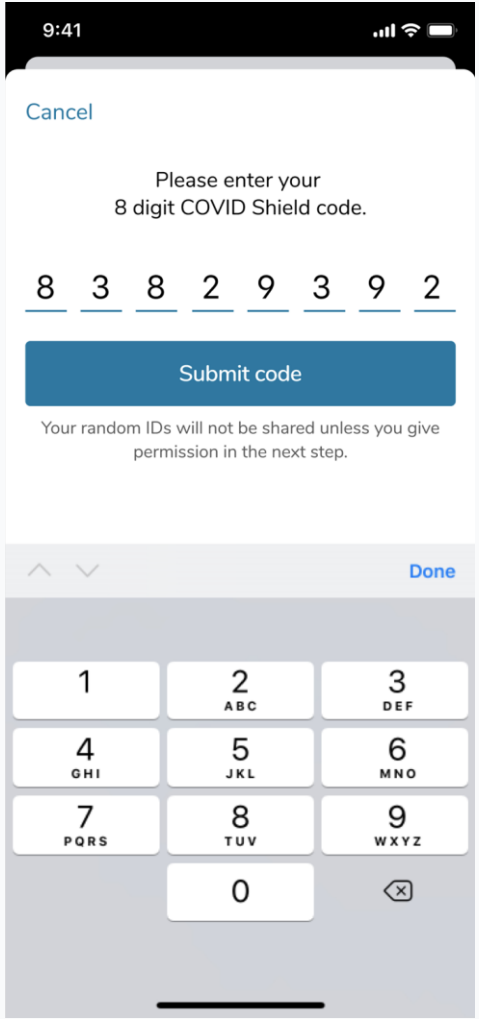
- Covid Alert uses a custom `accessibilityAutoFocus` prop to set heading focus
- This is **not** part of Facebook's API
- Might be other options available

```
<Text
  …
  accessibilityRole="header"
  accessibilityAutoFocus
>
  Share your random IDs
</Text>
```



Hiding things?


9:41
Cancel
Please enter your
8 digit COVID Shield code.
Submit code
Your random IDs will not be shared unless you give permission in the next step.
Done
1
2 ABC
3 DEF
4 GHI
5 JKL
6 MNO
7 PQRS
8 TUV
9 WXYZ
0


9:41
Cancel
Please enter your
8 digit COVID Shield code.
8 3 8 2 9 3 9 2
Submit code
Your random IDs will not be shared unless you give permission in the next step.
Done
1
2 ABC
3 DEF
4 GHI
5 JKL
6 MNO
7 PQRS
8 TUV
9 WXYZ
0

Cancel

Please enter your
8 digit COVID Shield code.

Submit code

Your random IDs will not be shared unless you give permission in the next step.

Done

Cancel

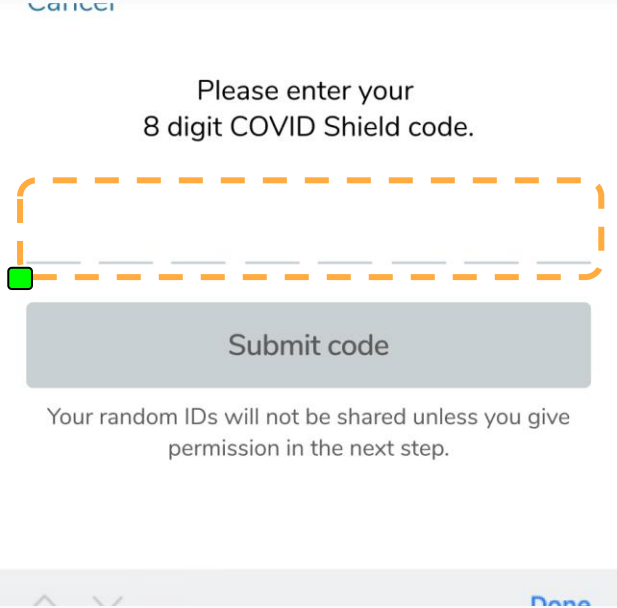
Please enter your
8 digit COVID Shield code.

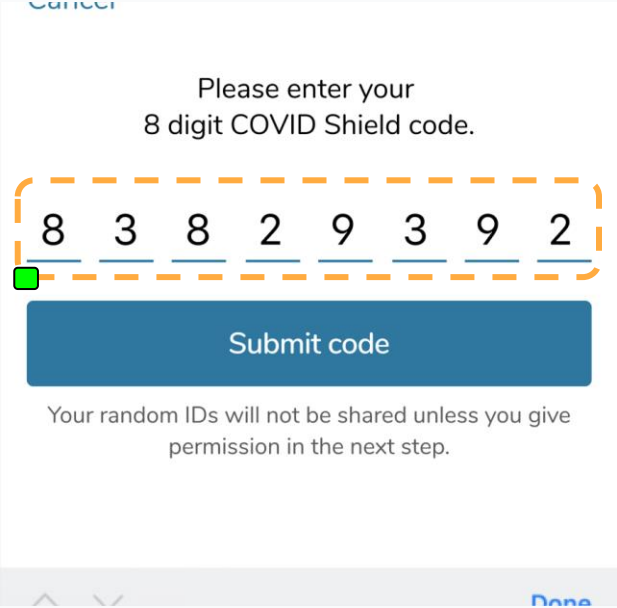
8 3 8 2 9 3 9 2

Submit code

Your random IDs will not be shared unless you give permission in the next step.

Done

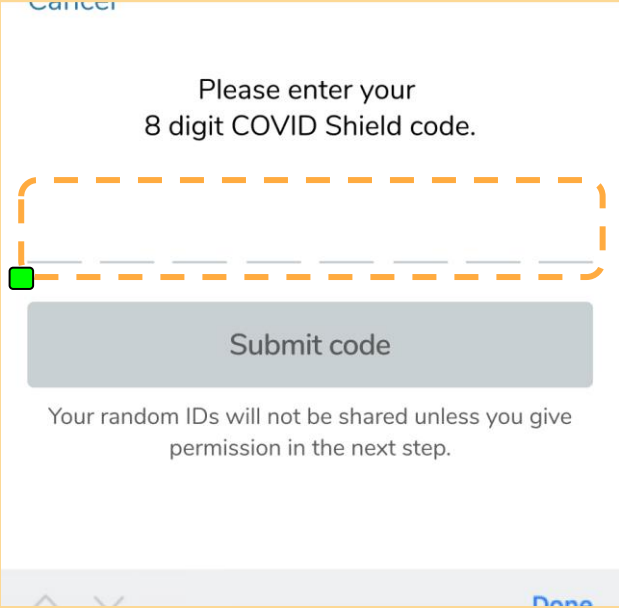

Cancel
Please enter your
8 digit COVID Shield code.
Submit code
Your random IDs will not be shared unless you give permission in the next step.
Done


Cancel
Please enter your
8 digit COVID Shield code.
8 3 8 2 9 3 9 2
Submit code
Your random IDs will not be shared unless you give permission in the next step.
Done

Accessibility issues…

1. `TextInput` missing role and name
2. `TouchableWithoutFeedback` missing role and name
3. `0` pixel input did not display a visible focus indicator
4. `TouchableWithoutFeedback` created extra focusable tab-stop

```
<TextInput
  value={value}
  ref={inputRef}
  ...
/>
<TouchableWithoutFeedback
onPress={giveFocus}>
  // ...
</TouchableWithoutFeedback>
```



Accessibility recommendations…

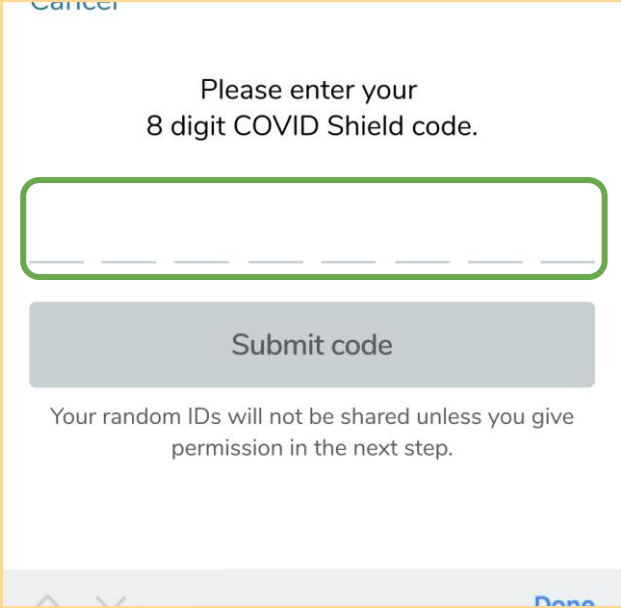
1. Add `accessibilityLabel` to `TextInput`

2. Adjust styles for screen reader discoverability/role

3. Hide the clickable control via `accessibilityElementsHidden` (for iOS) and `importantForAccessibility` (for Android) props

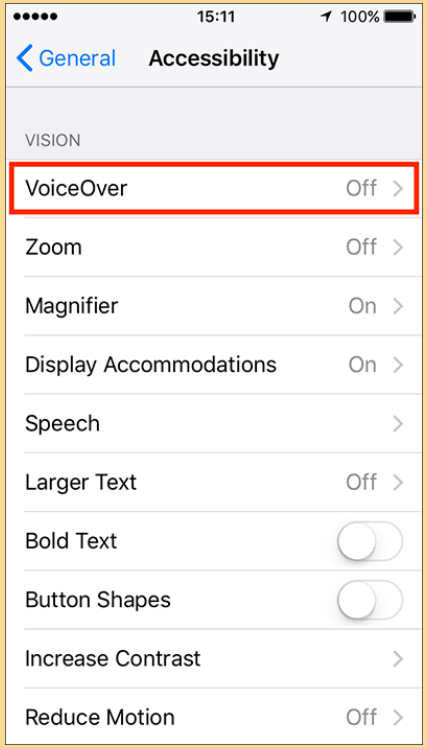
   This is similar to HTML's `aria-hidden="true"` + `tabindex="-1"`.

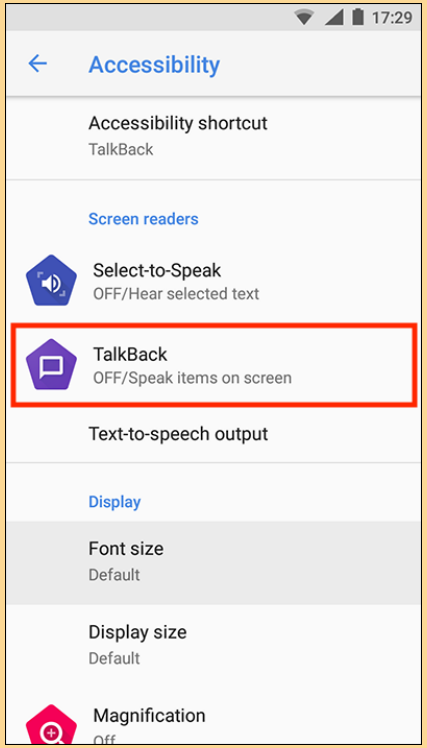
```
<TextInput
  accessibilityLabel="Covid Shield Code"
  ...
/>
<TouchableWithoutFeedback
  accessibilityElementsHidden={true}
  importantForAccessibility="no-hide-descendants"
  ...
>
  // ...
</TouchableWithoutFeedback>
```



Before:
🗣 "…" 🗣

After:
🗣 "Covid Shield Code, edit box"





Accessibility is more than screen readers…

- ☑ Font size
- ☑ Color inversion
- ☑ Reduce motion
- … and more!

Thank you! ☺

shopify