To make the world's information universally accessible and useful

Android for Everyone
Accessibility in Mobile

• • •

sierraOBryan

SIERRA OBRYAN (SHE/HER)

@_sierraOBryan

If you would prefer to follow along with a copy of the slides

As Google Slides : bit.ly/codemash-doc

As a PDF (on Github) : bit.ly/codemash-pdf

Make Tech Inclusive.
Make Inclusive Tech.

Kim Arnett @kaydacode

It's a team effort.

More users!

Risk Management

WHY?

Better for Everyone

The right thing to do

Accessibility is not only a "nice" thing to do, your app quality increases as you make it more accessible.

Victoria Gonda @ttgonda

Types of Disabilities

- Motor Impairments
  - May use a hardware device - Accessibility Switch - to control the app or accessibility menu
- Cognitive Impairments
  - May use Action Blocks to set up routines
- Visual Impairments
  - May use increased text size, Braille keyboard, or TalkBack
- Deaf and Hard of Hearing
  - May use Closed Captioning, Live Transcribe or Live Captioning

If you open your accessibility settings, you'll find even more options that folks might use on their device

Permanent | Temporary | Situational

- Touch: One arm | Arm injury | New parent
- See: Blind | Cataract | Distracted driver
- Hear: Deaf | Ear infection | Bartender
- Speak: Non-verbal | Laryngitis | Heavy accent

Learn more

Inclusive
A Microsoft Design Toolkit

Sierra

I don't want to touch my phone when I'm trying to "cook"

I forget to blink when I'm wearing contacts

I like to stay up too late watching TV while everyone else is sleeping

I am a mumbler

@_sierraobryan

What are the Web Content Accessibility Guidelines and why do they matter for mobile?

Learn more

We're going to talk about both **traditional Android UI** and Development and **Jetpack Compose**.
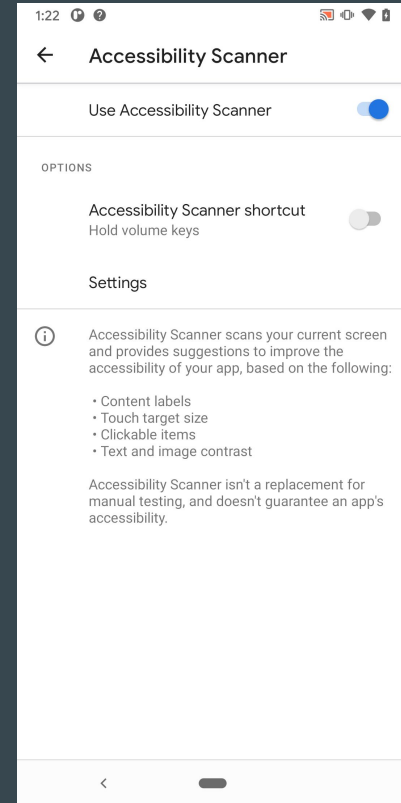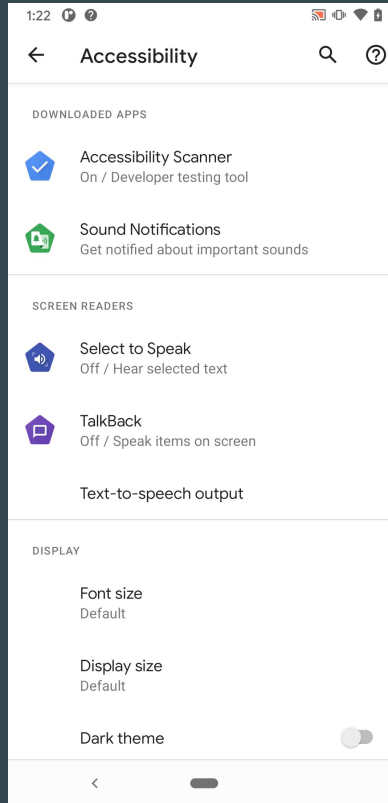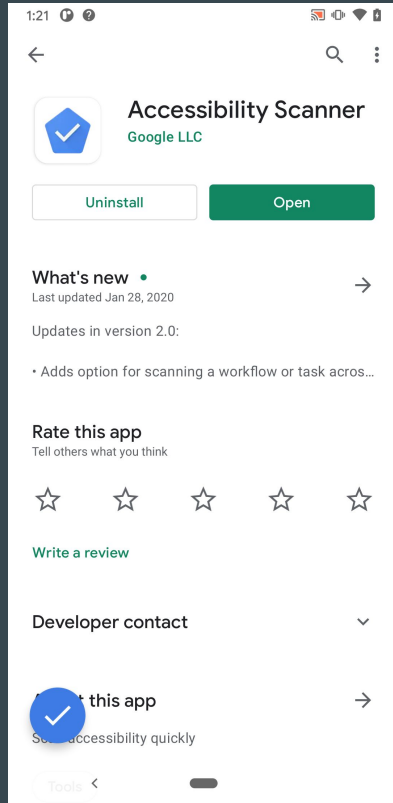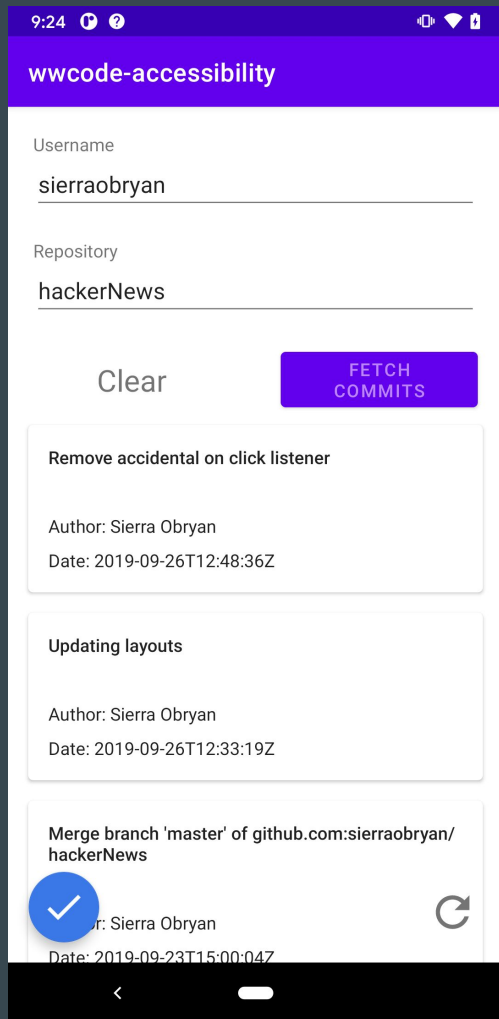
Consistency

How to test your app for accessibility [Learn More](#)

- User and Manual testing
- Analysis Tools
  - Accessibility Scanner
  - UI Automator Viewer
  - Lint
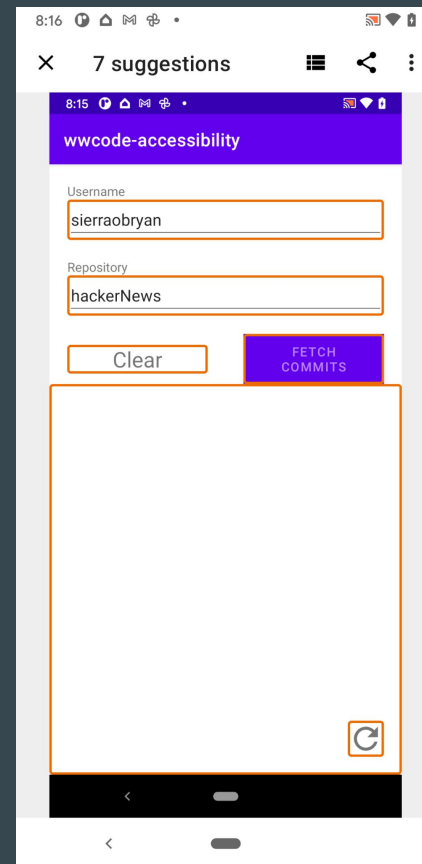- Automated Testing
  - Espresso and Robolectric
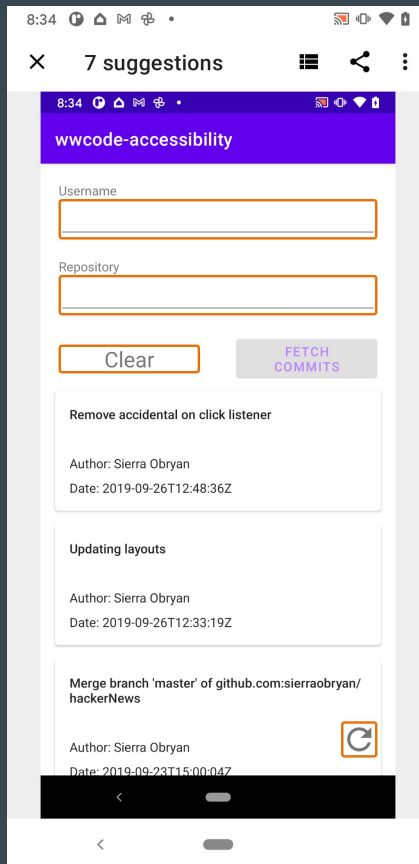
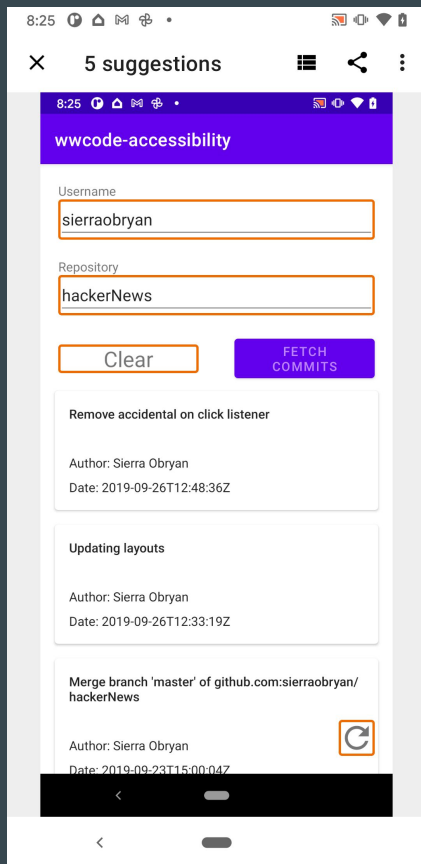# Using the Accessibility Scanner [Get the app](#)
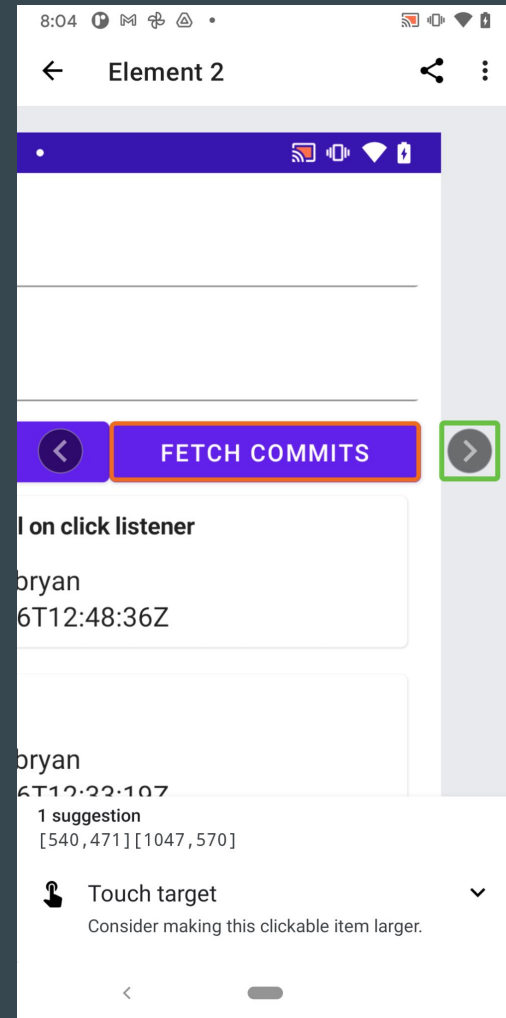
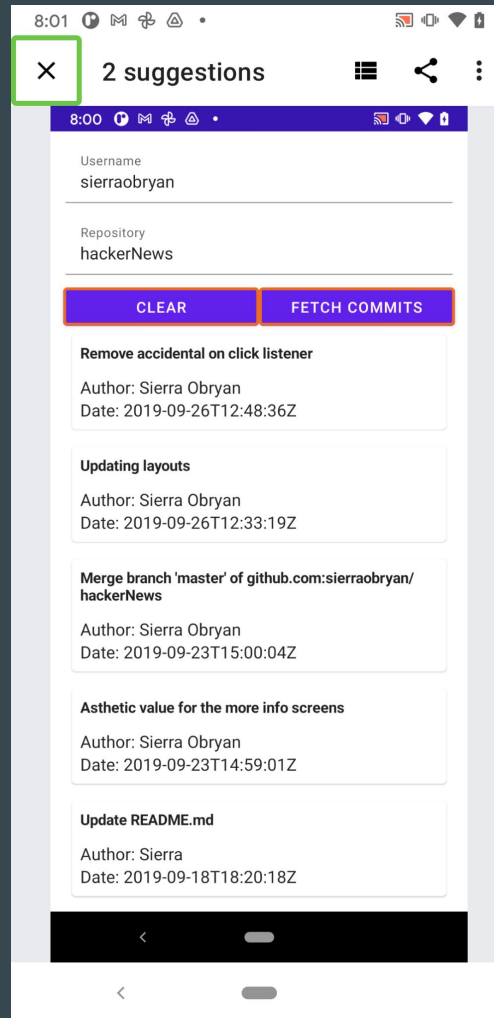Let's see how it works for a simple app

Accessibility Scanner is a good tool but it is NOT perfect - it can only check what's on the screen

Does the scanner work for apps written with Compose?

Yes!

Let's jump in and talk about solving some of are accessibility suggestions!

Touch Targets [Learn More](#)

The recommended size for each interactive UI element's focusable area, or **touch target size**, is at least 48px by 48px.

This does **not** mean the UI element's visible area must be at least 48px by 48px.

This can be achieved with padding and height / width or minHeight / minWidth.

paddingTop + height/minHeight + paddingBottom ≥ 48dp
paddingStart + width/minWidth + paddingEnd ≥ 48dp

PS - what does minHeight do? Just adds padding to meet the required height

Why do I care about touch target size?

Have you ever played an free mobile game...
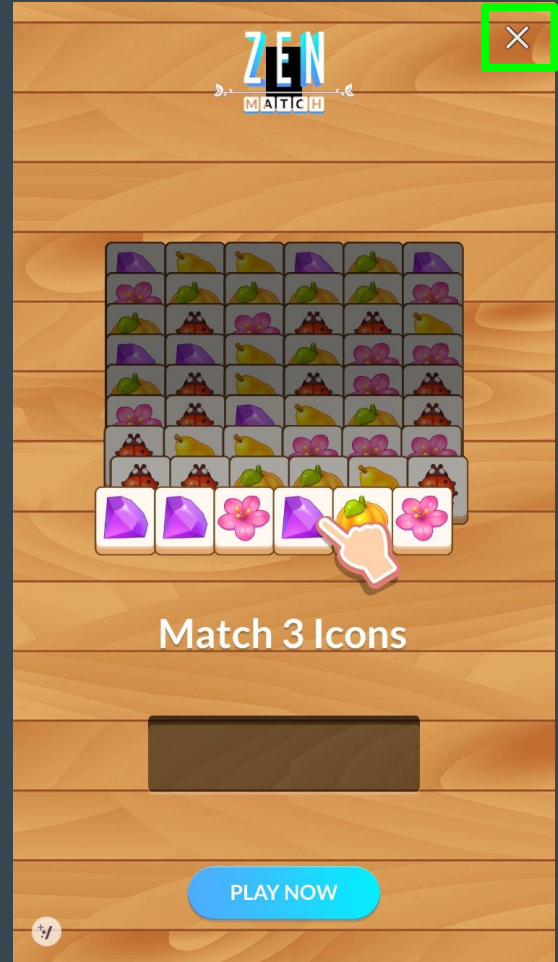
How do I decide when to set a size or set a min?

How do we do this in XML?

```
<TextView/EditText
    android:id="@+id/element_id"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:minHeight="48dp"
    ...
/>
```

For dynamic text, consider using minHeight.

Remember :
Localization == Dynamic

Combine width + height and padding to keep the same UI but increase touch target

```
<ImageView
    android:layout_width="40dp"
    android:layout_height="40dp"
    ...
/>
```

```
<ImageView
    android:layout_width="48dp"
    android:layout_height="48dp"
    android:padding="8dp"
    ...
/>
```

```
TextField(
    value = value,
    onValueChange = onValueChanged,
    modifier = modifier
        .fillMaxWidth()
        .heightIn(min = 48.dp)
        .clip(shape = shapes.medium),
)
```

Use heightIn to set min (and max!)
height and width

```
Icon(
    imageVector = icon,
    contentDescription = contentDescription,
    modifier = Modifier
        .clickable { ... }
        .size(40.dp)
)
```

What about
Jetpack Compose?

```
Icon(
    imageVector = icon,
    contentDescription = contentDescription,
    modifier = Modifier
        .clickable { ... }
        .padding(8.dp)
        .size(40.dp)
)
```

Use modifiers to set size
and padding

Color Contrast [Learn more](#)

AA Compliance requires at least a requires **4.5 : 1** for regular text and **3 : 1** for large text

Slides: 5.88 : 1

7.62 : 1   **3.92 : 1**   **2.38 : 1**

How do I meet these requirements and stick to my theme?

Use [Material Color Palettes](#)!

How do I check my colors? There are lots of [tools](#) online!

WCAG 3 will use a new color contrast method called the Advanced Perceptual Contrast Algorithm.

Learn More

Labels (and hints) <u>Learn more</u>

The basics of adding labels is that each UI element in your app should include a description that describes the element's purpose.

We do this using `contentDescription` although we get some for free - we do not need to provide a description for `TextView` because Android Accessibility automatically announces the text as the description.

Some best practices: Use the right UI element, don't include the UI element in the description, descriptions should be unique, skip over decorative effects

Localized, Concise, descriptive

Why do I care about content descriptions?

You might be already using them without thinking about it!

There's a lot to think about with Labels

Does it need a label? It depends but generally...

TextView: No

EditText: Include a hint

Text Button: No

ImageButton: Yes

ImageView: Maybe

Is it a decorator?

Can it be skipped?

Do they make more sense together?

Should it be grouped?

Does it include numbers or abbreviations?

How will it be read?

Use the right UI Element

If you're adding an onClick action to one of these...   **Consider using these...**

I want to add an onClick to a TextView → Use a Button

I want to add an onClick to a ImageView → Use an ImageButton

I want to make a Text Composable clickable → Use a Button Composable

I want to make a Icon Composable clickable → Use a IconButton Composable

EditText [Learn more](#)

```xml
<TextView
    android:id="@+id/name"
    android:text="@string/username"
    android:labelFor="@id/username_input"
    ...
    />

<EditText
    android:id="@+id/username_input"
    android:layout_width="0dp"
    android:minHeight="48dp"
    android:layout_height="wrap_content"
    ...
    />
```

```xml
<TextView
    android:id="@+id/repo"
    android:text="@string/repository"
    ...
    />

<EditText
    android:id="@+id/repo_input"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:minHeight="48dp"
    android:hint="@string/repository"
    ...
    />
```

Pairs of elements where one describes the other

Hint is example of valid input and available to the screen reader

Grouping Labels [Learn more](#)

- Out of the box, TalkBack will read each RecyclerView as one continuous label and include the position in the list

```
<androidx.constraintlayout.widget.ConstraintLayout
    android:id="@+id/container a"
    android:layout width="match parent"
    android:layout height="wrap content"
    android:screenReaderFocusable="true" (3)
>

</androidx.constraintlayout.widget.ConstraintLayout>

<androidx.constraintlayout.widget.ConstraintLayout
    android:id="@+id/container b"
    android:layout width="match parent"
    android:layout height="wrap content"
    android:screenReaderFocusable="true" (3)
>

</androidx.constraintlayout.widget.ConstraintLayout>
```

1. If I add screenReaderFocusable to one of the UI elements, it will read it separately
2. If this were not a recyclerView, I can use view attributes to make it read as natural group - this can make finding information on your screen more efficient
3. Sometimes you may want to read the container itself

What about Jetpack Compose?

For some composables,
we have access to the
content description in
function parameters

```kotlin
@Composable
fun Image(
    painter: Painter,
    contentDescription: String?,
    modifier: Modifier = Modifier,
    alignment: Alignment = Alignment.Center,
    contentScale: ContentScale = ContentScale.Fit,
    alpha: Float = DefaultAlpha,
    colorFilter: ColorFilter? = null
)
```

@_sierraobryan

Now let's look at our TextField

```
TextField(
    ...
    placeholder = { Text(text = "Placeholder") },
    ...
)
```

IThe placeholder text is like a hint - it will show in our Textfield when there is no input.

```
TextField(
    ...
    label = { Text(text = "Label") },
    ...
)
```

A label will act like both like the placeholder and as a label above the TextField

Semantics add context to your UI elements.

Content Description
State Description
Progress Bar Range Info
Heading
Pane Title
Disabled
Live Region
Focused
Invisible to User
Scroll Axis Range
Pop up
Dialog
Role
Password
Selectable Group
Custom Actions
onClick
onLongClick

@_sierraobryan

We can also combine our views

In this example, our all three Texts will be read as a group

```
Column(
    modifier = Modifier.weight(1f)
        .semantics(mergeDescendants = true) {}
) {
    Text(text = commit.commitMessage)
    Text(text = "Author: ${commit.author}")
    Text(text = "Date: ${commit.date}")
}
```

In this example, our Column, Text, and TextField will be each be read separately.

```
Column(modifier = Modifier
    .fillMaxWidth()
    .semantics {
        contentDescription = "Container"
    }
) {
    Text(text = "Text")
    TextField(label = { Text(text = "Label") })
}
```

There are a lot of tools to help

And so there are many UI attributes to help build an accessible UI (learn more)

```
labelFor
screenReaderFocusable
accessibilityHeading
```

```
All of our
Semantics Properties!
```

A TtsSpan is a special type of span that can pass in metadata to give contextual information about the string. This information can help Text to Speech correctly pronounce a text element

123456789   TYPE_DIGIT

"1 hundred 23 million 4 hundred 56 thousand 7 hundred 89" vs
"1 2 3 4 5 6 7 8 9 "
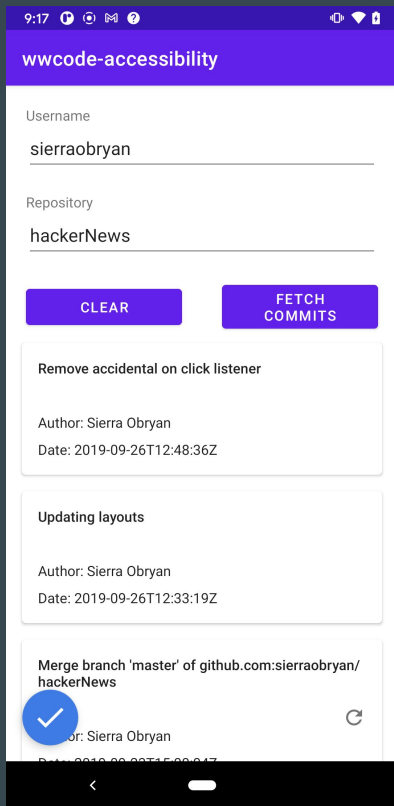
What else?

- Different Displays
- Custom Views and Accessibility APIs
    - Swipe to delete
    - Actions
    - Spans with link
    - Custom Alerts
- Other Media

Think about (different) Displays [Learn more](#)

Normal

Large Text

Dark Mode

Swipe to delete

```
ViewCompat.addAccessibilityAction(mailCell, R.string.remove, (v,b) ->
    mailCell.delete());
```

Spans with links

```
ViewCompat.enableAccessibleClickableSpanSupport(hamsterInfoView);
```

Custom view

```
ViewCompat.setAccessibilityPaneTitle(alertView, R.string.alertTitle);
```

```
Column(
    modifier = Modifier
        .semantics { paneTitle = "Alert Title" }
) { ... }
```

Accessibility-friendly title for a screen's pane

Adding a custom action

```
Column(
    modifier = Modifier
        .semantics {
            customActions = listOf(
                CustomAccessibilityAction(
                    label = "Delete",
                    action = { delete(); true }
                )
            )
        }
) { ... }
```

Other Media

(remember that cat video?)

UI Testing!

What's next?

The Web Content Accessibility Guidelines v3 are set to be published in 2023

what does that mean for mobile?

Draft 1 was published in late Jan 2021 and continues to have updates

where do I learn more?

Android Accessibility by Tutorials! by Victoria Gonda

I'm just getting started with Accessibility + Compose

what should I do next?

Jetpack Compose: Accessibility and the new Code Lab

To make the world's information universally accessible and useful

Thank you!

...

Where to find me?
🐦 @_sierraObryan
sierraobryan.dev