SwiftUI & Accessibility: Goodies and Gotchas

Kate Owens
iOS Engineer - Deque Systems

508

AUDIT

ARIA

ADA

WCAG

Agenda

- Brief overview of SwiftUI
- SwiftUI and accessibility
- Goodies & Gotchas

SwiftUI

A brief overview

What's the story with SwiftUI?

- Announced in 2019 at WWDC
- Declarative syntax
- No more constraints!
- Create beautiful views with much less code
- The code is very handsome and readable

```
struct MyView: View {
    var body: some View {
        Text("Hello, World!")
    }
}
```

SwiftUI and accessibility overview

Accessibility in SwiftUI

- Most elements are VoiceOver and other assistive technology (i.e. switch control, VoiceControl) focusable
- Most elements have a11y labels by default
- Not all custom controls are focusable and labeled by default
- Somewhat decent accessibility experience out of the box, but not great - it could be a lot better

Goodies & Gotchas

Things I learned from creating an accessible SwiftUI iOS app

Quick Note

Plantiverse - an Accessible SwiftUI App

- To demonstrate SwiftUI accessibility goodies and gotchas, I'll be using Plantiverse - a simple SwiftUI app I created to get familiar with accessibility in SwiftUI
- Plantiverse is public - feel free to clone it if you want to play around with the accessibility!
- Repo: https://github.com/kateowens12/Plantiverse

Goodies & Gotchas: A Brief Overview

What is a Goodie? What is a Gotcha?

- Goodie
  - makes it easier for a developer to make accessibility changes that improve the accessibility experience
- Gotcha
  - makes it easier for a developer to make an accessibility mistake, possibly without even realizing it. Gotchas are sneaky
- A Goodie can also become a Gotcha if it’s not used correctly

Goodies

Goodie #1: Default Basic Accessibility

- Accessibility Notifications handled automatically
- Most elements are automatically marked as accessibilityElements
- Default controls and elements (ex. Button, Toggle, TextField, Image):
    - Automatically focusable with an AT (Assistive Technology)
    - Assigned an action if applicable
    - Assigned a default accessibility label
- Custom controls:
    - Automatically given an accessibility label from the Text

Gotcha In Action: Default Basic Accessibility

Goodie #2: Accessibility Representation API

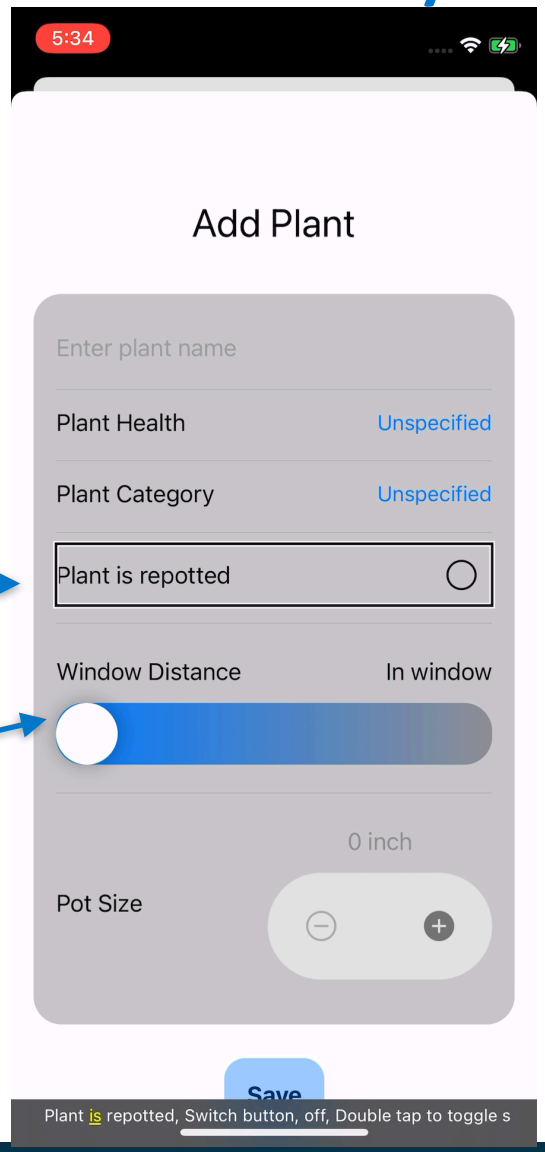
- Replaces accessibility elements in a view with another
  - e.g. replace a custom toggle's accessibility elements with a standard Toggle's accessibility elements
- Uses a non-interactive, hidden view specified in a closure to generate accessibility elements
- Accessibility Representation API makes it much simpler to make custom controls fully accessible

```
.accessibilityRepresentation {
    Toggle(isOn: $isRepotted) {
        Text("Plant is repotted")
    }
}
```

Goodie in Action: Accessibility Representation API

Without the accessibilityRepresentation API,
the default announcement for this custom Toggle is:
“Plant is repotted, Circle, Image”

Without the accessibilityRepresentation API,
the default announcement for this custom Slider is:
“Window Distance, In Window” and slider control is
Not focused with VoiceOver

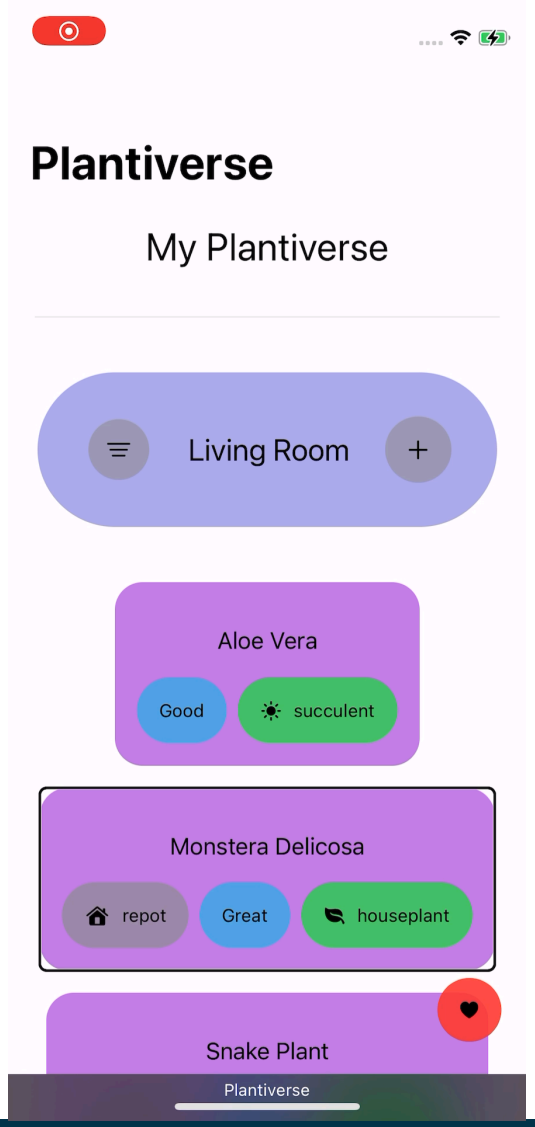


Goodie #3: Accessibility Custom Content

- Accessibility Custom Content API is a way to provide information to users in smaller, more specific portions when they need it
  - Ex. Plant tile contains the name, health, category, and tasks due, but a user might not want to know all of that data every single time an element is focused
- Views with somewhat complex data sets:
  - Can be overwhelming to navigate using an AT
  - May be providing more information than a user needs each time an element is focused with an AT

Goodie in Action: Accessibility Custom Content

If we don’t use the accessibilityCustomContent API,
The default announcement for this element is:
“Monstera Deliciosa, button, repot, great, houseplant”



Goodie #4: Accessibility Sort Priority

- Changes the sort order for accessibility elements
- Only use this when it makes sense for the user experience in terms of navigation and semantic views
  - I.e. Will the user have more context/a better experience if one element is focused before or after another?

Accessibility Sort Priority in Action

Exhibit A:

accessibilitySortPriority set for edit button in toolbar

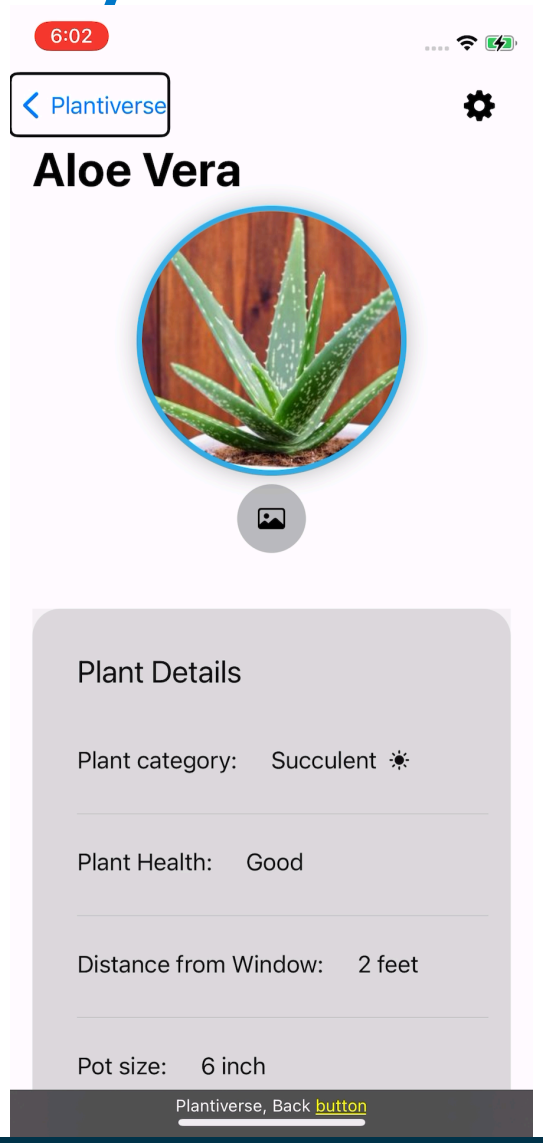
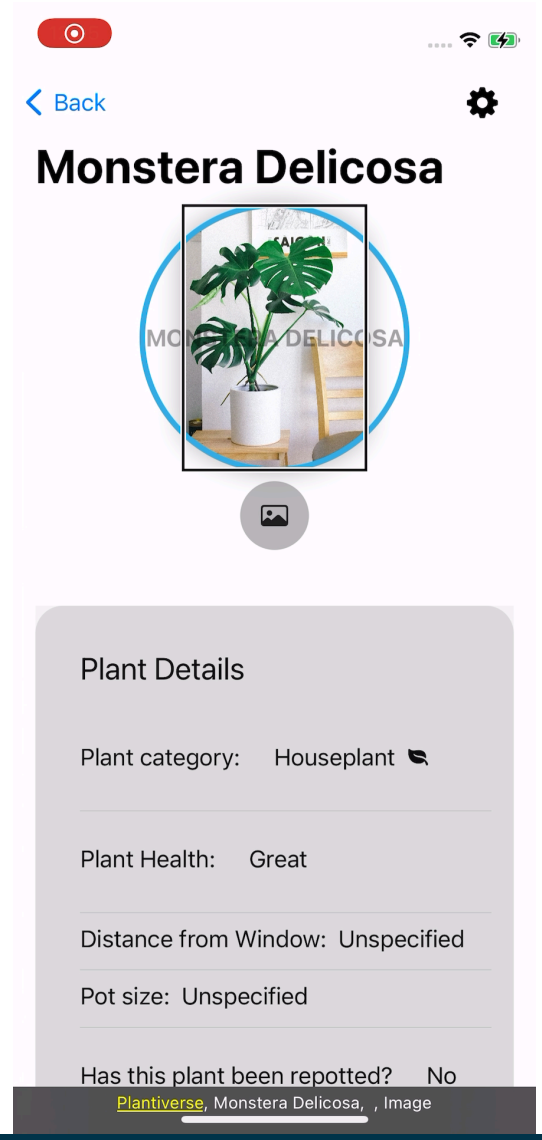


Exhibit B:

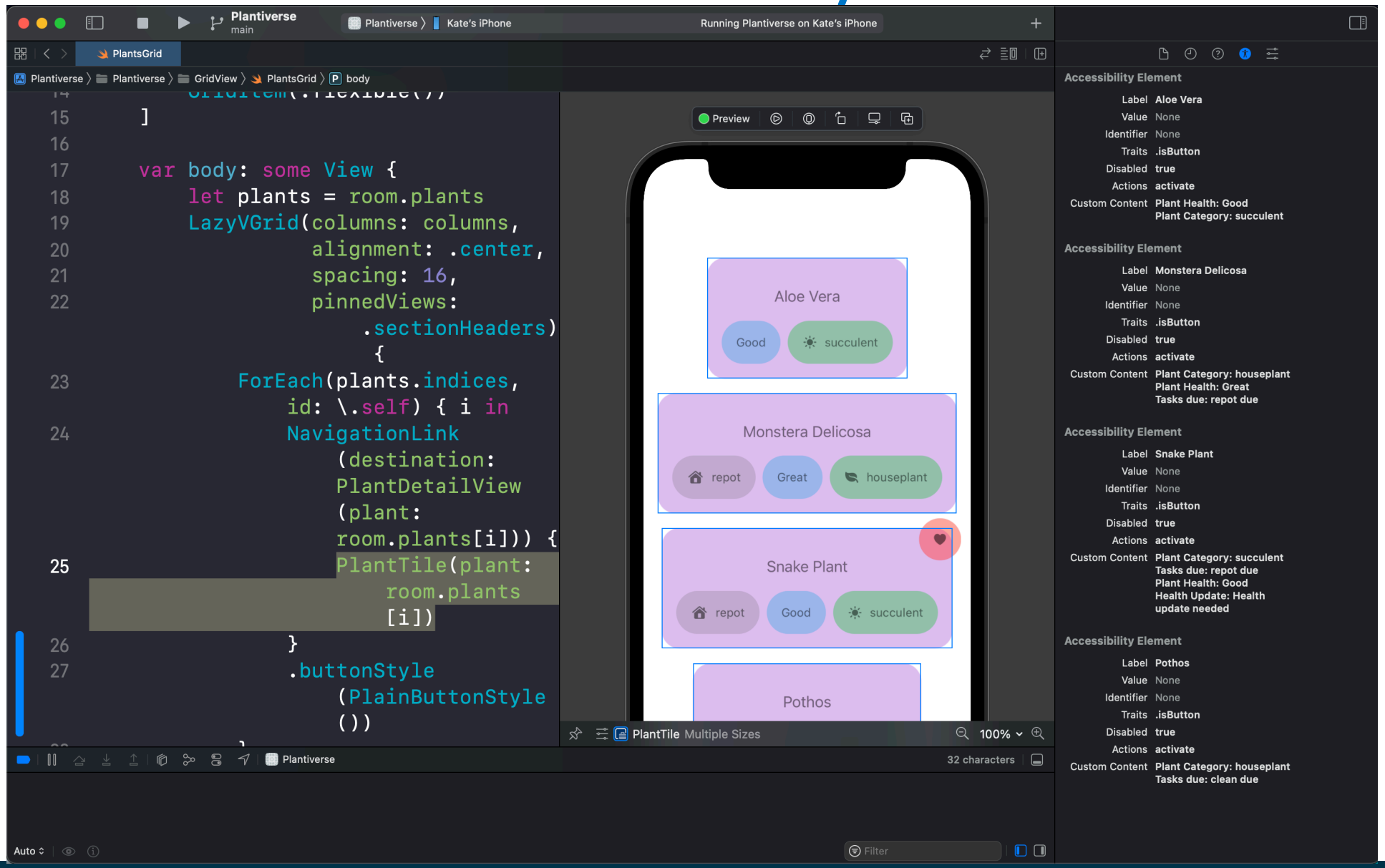
accessibilitySortPriority NOT set for edit button in toolbar



Goodie #5: Accessibility Preview

- Allows you to preview the accessibility properties of a view right from Xcode - no running the app required!
  - accessibilityLabel
  - accessibilityValue
  - accessibilitySortOrder
- Can save time - you can have an idea of what the a11y properties are without needing to run the app

Goodie in Action: Accessibility Preview



Goodies

- Default basic accessibility
- Accessibility Representation API
- Accessibility Custom Content
- Accessibility Sort Priority
- Accessibility Preview

Gotchas

Gotcha #1: Default Basic Accessibility

- Default accessibilityLabel doesn’t always make sense without additional information or context
- The order in which elements are focused with an AT doesn’t always result in a logical, easily navigable experience
- Some elements make more sense when they are grouped together/contained in a certain way
- Certain types of custom controls do not always have the same default basic accessibility capabilities that others do
  - i.e. one type of custom control may be activated with an AT by default, while another type may not
- Decent out of the box basic accessibility != great accessibility experience for users

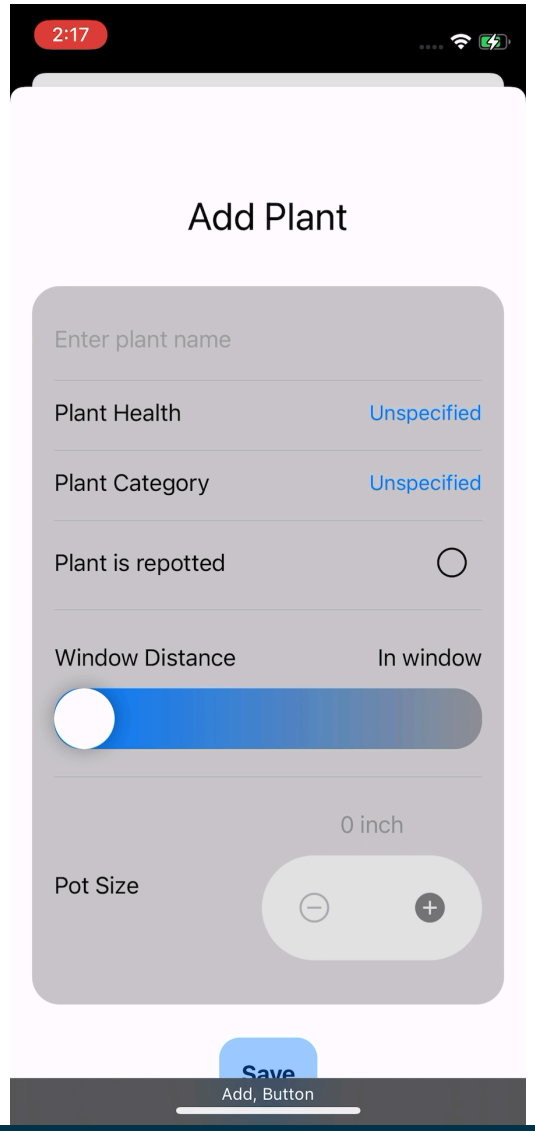
Gotcha In Action: Default Basic Accessibility

A lot of repeated information

Elements in containing views are not grouped together

Custom toggle announcement doesn't make it clear that it's a button

Custom slider control cannot be focused with VoiceOver



Gotcha #2: Accessibility Sort Priority

- Changes the order in which accessibility elements are focused with an AT
- Use this with EXTREME caution and lots of thought and care
  - Which elements and in what order would a user want to navigate through them?
  - Which order makes the most sense?
  - Which sort order provides all the necessary context?
- If used incorrectly .accessibilitySortPriority can make an app very confusing and/or inefficient for a user using an assistive technology to navigate
- Don’t choose the sort priority based on where you WANT users to swipe to first - choose based on what would make the most semantic sense to the user

Gotcha #2: Accessibility Sort Priority continued

- Only use this modifier if it will make your app easier and more logical to navigate
  - Never use this modifier to increase prominence or consumption rate of something that's revenue generating - accessibility sort priority is here to make your app more navigable, not to manipulate the user experience to increase revenue or eyeballs on something
  - Ex. In a paid app scenario - don't change the sort priority to focus on a 'pay' button first if it doesn't make logical sense for navigation

Gotcha #2: Accessibility Sort Priority continued

- If a view contains multiple instances of a view that's frequently focused on with an AT, consider using the accessibilityElement .contain modifier to avoid a potentially repetitive, cumbersome navigation experience

Gotcha In Action: Accessibility Sort Priority continued

Exhibit A:

The room heading may seem like a good candidate for accessibilitySortPriority to focus the room name before the buttons on either side.

However, it's not! This makes a cumbersome experience when navigating to another room in our Plantiverse - we must swipe through each room heading, each element within that heading, then each plant tile within that room.

That is A LOT of swiping!!

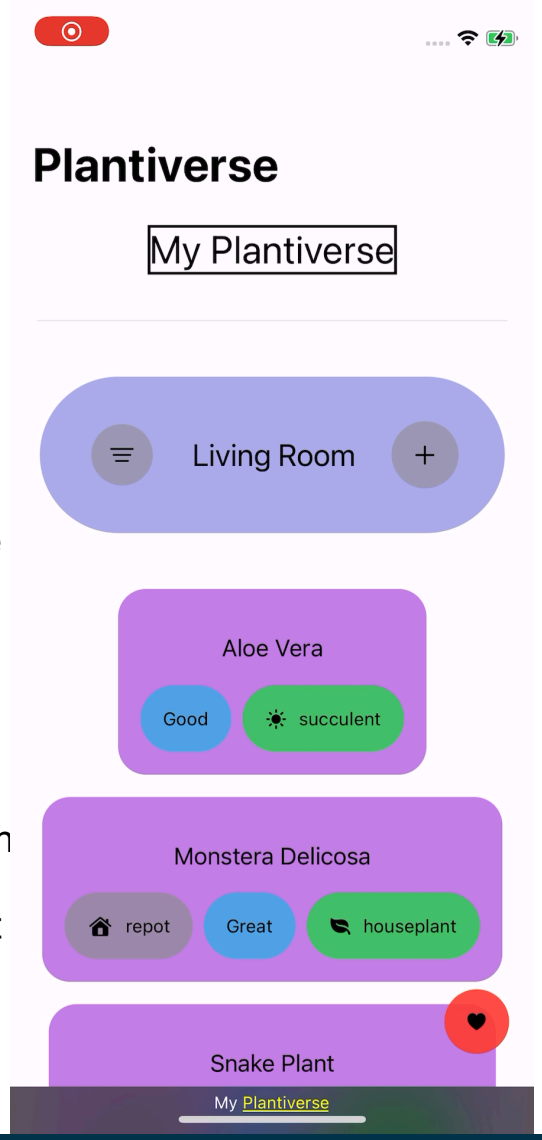
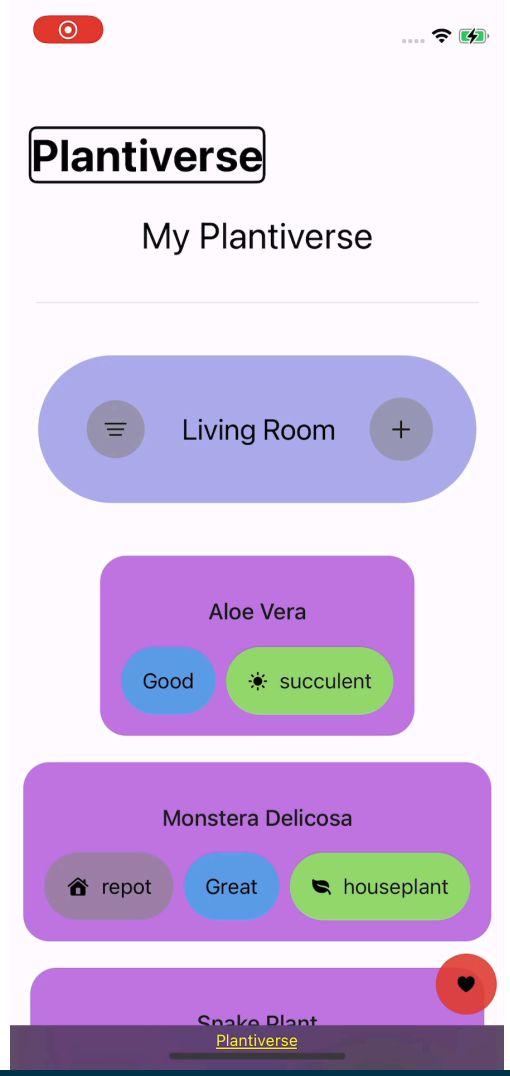


Exhibit B:

If we use the .accessibilityChildren(.contain) behavior instead, we can swipe through each room heading as a singular element, rather than each room heading and then each of the elements within the heading



Gotcha #3: SF Symbol Images

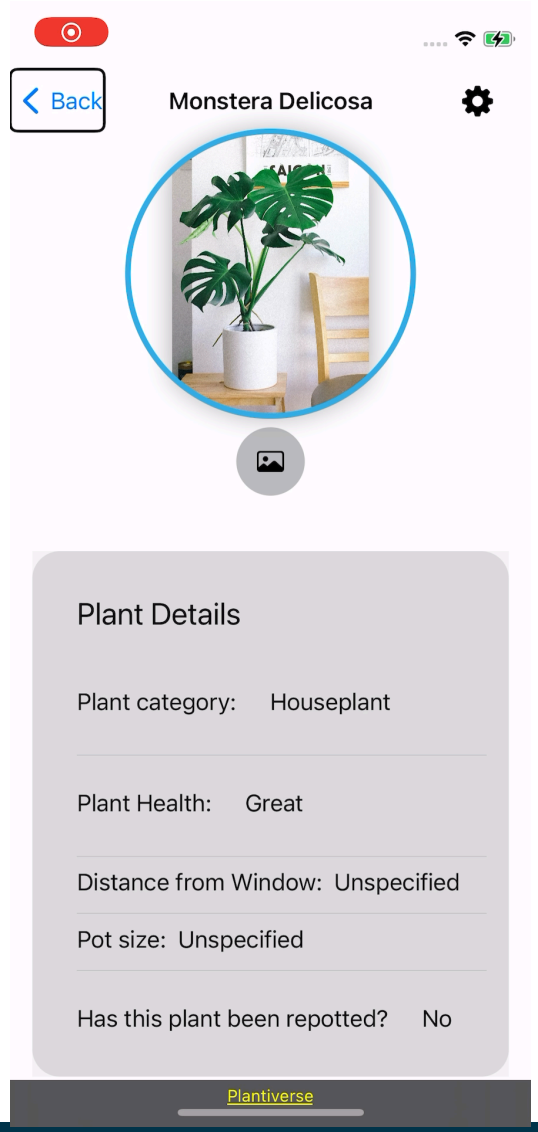
- Image element's systemName initializer creates an image using an SFSymbol

```
var body: some View {
    Image(systemName: "gearshape.fill")
}
```

- The systemName is used as the accessibilityLabel for the element unless otherwise specified
- Use an a11y label modifier on the Image to remedy this

```
var body: some View {
    Image(systemName: "gearshape.fill")
        .accessibilityLabel("Edit Plant")
}
```

Gotcha In Action: SFSymbol Images

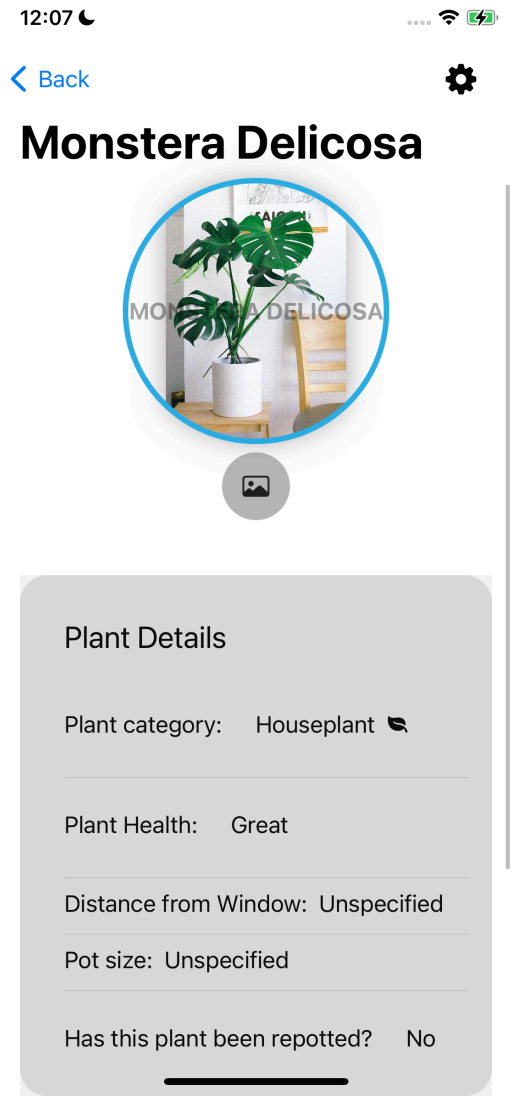


Gotcha #4: Text Based Image Overlays

- When using an image overlay that contains Text, be certain that the color contrast ratio is passing
  - Adding an accessibility label in this scenario will not resolve a color contrast failure - sighted, low-vision users who don’t use a screen reader will not be able to consume this information

Gotcha in Action: Text Based Image Overlays

The color of this text overlay does not contrast enough!



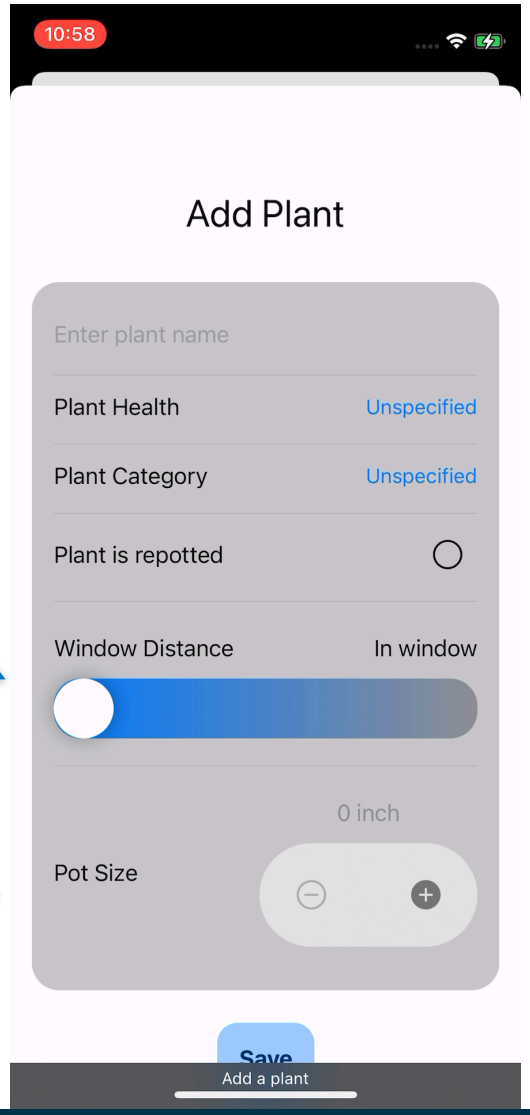
Gotcha #5: Pobody's Nerfect

- There's no substitution for experiencing your app's accessibility first hand context is also very important - the human element of creating an accessible swiftUI app will always be crucial regardless of how mature, stable, robust the language is
- NEVER operate on the assumption that everything is working flawlessly
- SwiftUI and some of the a11y APIs we've covered are relatively young - therefore there are still some bugs present
  - Ex. Currently there is a bug in accessibility representation API when an adjustable control is used

Gotcha in Action: Pobody's Nerfect

Custom Slider should be announced as: "Window Distance, In Window, Adjustable, swipe up or down with one finger to adjust the value'. Slider position and value label should update with each swipe.

Custom Stepper should be announced as: "Pot Size, 0 inch, Adjustable, swipe up or down with one finger to adjust the value'. Value label should update with each swipe.



Correct behavior does not happen until VoiceOver is turned off and back on

Gotchas

- Default basic accessibility
- Accessibility sort priority
- SF symbol images
- Text based image overlays
- Pobody’s nerfect

Turning the Gotchas into Goodies

How can I fix issues from Gotchas?

- Use Accessibility preview in Xcode
- Always experience your app with an AT
- axe DevTools XCUI with SwiftUI support is now available!
- axe DevTools for iOS - UIKit
- Deque can help!
  - Manual assessments and audits
  - Accessibility consulting
- Check out the example app from this talk on gitHub for accessibility examples

Summary

Some Key Takeaways

Summary

- SwiftUI provides a somewhat decent accessibility experience out of the box
- Default elements and controls are focusable with an Assistive Technology
    - A default accessibility label is provided
    - Actions provided automatically so users can activate controls
- Not all custom controls are focusable by default
- Goodies:
    - Default basic accessibility
    - Accessibility Representation API
    - Accessibility Sort Priority
    - Accessibility Custom Content
    - Accessibility Preview
- Gotchas:
    - Default basic accessibility
    - Accessibility Sort Priority
    - SFSymbol Images
    - Text Based Image Overlays
    - Podbody's Nerfect

Any Questions?

 @dequesystems

 https://github.com/kateowens12/Plantiverse

 deque-systems-inc

 dequesystemsinc

 Kate.owens@deque.com