droidcon academy

Accessibility in Android Jetpack Compose: User Interface Components

Quintin Balsdon

in In/qbalsdon

Section Overview

- Labels
- State
- Touch target size
- Focus
- Creating a custom component
- Espresso testing

droidcon academy

```
Icon(
    imageVector = ...,
    contentDescription = "An icon in Jetpack Compose"
)
```

Labels

- Known as content description, alternative text or alt text
- Built in for icons

```
modifier = Modifier
    ...
    .semantics {
        contentDescription = "A modifier with a content description"
    },
```

Labels

- Known as content description, alternative text or alt text
- Can be added to any component

```
with(composeTestRule...) {
    onNodeWithContentDescription("...")
    assert(hasText("..."))
    assert(hasTextExactly("..."))
}
```

Espresso testing: Labels

droidcon academy

```
modifier = Modifier
    ...
    .semantics { role = Role.Button }
```

Roles

- Identify the component in a consistent manner to the user (e.g. button, switch)
- Elements need both a name and role

```
fun SemanticsNodeInteraction.assertHasRole(role: Role): SemanticsNodeInteraction =
    assert(hasRole(role))

fun hasRole(role: Role): SemanticsMatcher = SemanticsMatcher.expectValue(
    SemanticsProperties.Role, role
)
```

Espresso testing: Roles

droidcon academy

```
.toggleable(
    role = Role.Checkbox,
    value = checked,
    onValueChange = { newValue -> ... }
)
```

State

- Indicate the value of a variable within the component

```
with(composeTestRule...) {
    assertIsToggleable()
    assertIsOff()
    ...
    assertIsOn()
}
```

Espresso testing: State

Touch Target Sizes

- Our fingers touch the screens a little bit off from our perspective
- Recommended minimum touch target size is 48dp x 48dp
  - WearOS: no exceptions
  - Other platforms: Some secondary components can be 32dp x 32dp

```
with(composeTestRule...) {
    assertHeightIsAtLeast(32.dp)
    assertWidthIsAtLeast(32.dp)
}
```

Espresso testing: Touch target size

- Also use the Accessibility Scanner

```
fun applyFocusColor(focusState: FocusState) {
    borderColor = if (focusState.isFocused) {
        focusColor
    } else {
        noColor
    }
}

Modifier
    .onFocusEvent(::applyFocusColor)
    .border(
        color = borderColor
    )
```

Focus State

- Use events and variables
- Keep in mind: Focused / Unfocused vs Selected / Unselected

droidcon academy

```
val (first, second, third) = remember { FocusRequester.createRefs() }


modifier = Modifier
    .focusRequester(first) // associate with a reference
    .focusProperties { next = second } // use the reference
```

Focus Order

- Create references (“anchors”)
- Assign reference
- Assign the properties

droidcon academy

Focus

- Try not to make significant changes using on focus, it can be jarring for users, and they might not want to perform an action
- Beware of circular references (keyboard traps)
- Give users what they expect
- Make sure focus highlight is visible

Custom component: Choice Chip

- Assume **FilterChip** doesn’t exist
- We need to create the following component
- Let’s start with tests!

droidcon academy

Section Summary

- Labelling components: Name and Role
- Component State
- Touch target size
- Focus
  - Highlighting
  - Ordering
- Custom components
- Espresso testing

Grouping Related Content

Up Next

droidcon academy