Jetpack Compose

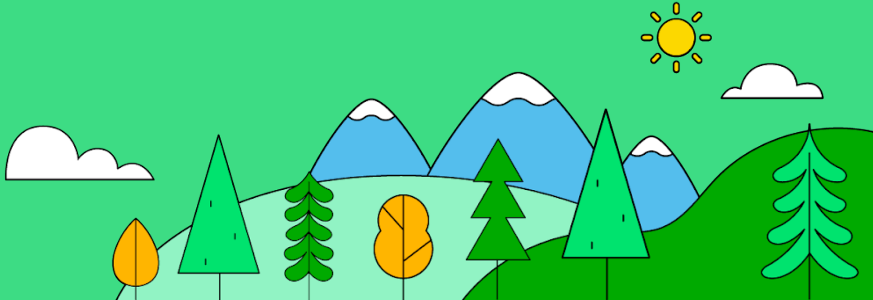
para Android Developers

android

Expositor

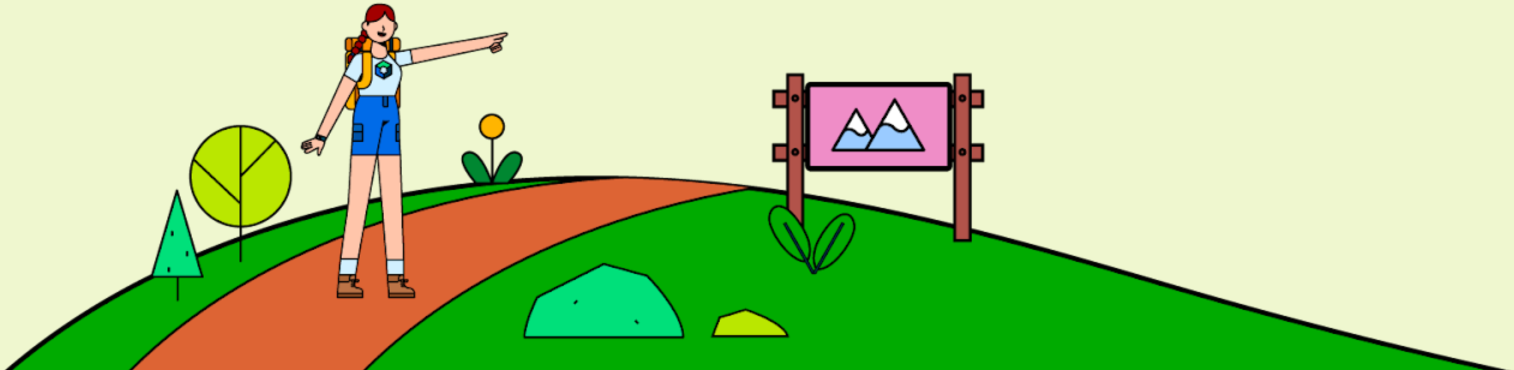
Bruno Aybar
Shopify, Senior Mobile Developer

Twitter: @brunoaybarg
Github: @Bruno125



android

Experienced Track



Requisitos:

Conocimientos básicos de

- Android
- Kotlin

android

Jetpack Compose para Android Developers

Este curso contiene 5 secciones:

1	**Compose Essentials**	Tus primeros pasos con Jetpack Compose. Entenderás lo que significa que Compose sea un toolkit de UI declarativo, y cómo usarlo para construir UIs increíbles.	(5 horas)
2	**Layouts, theming, and animation**		(2 horas)
3	**Architecture and state**		(3 horas)

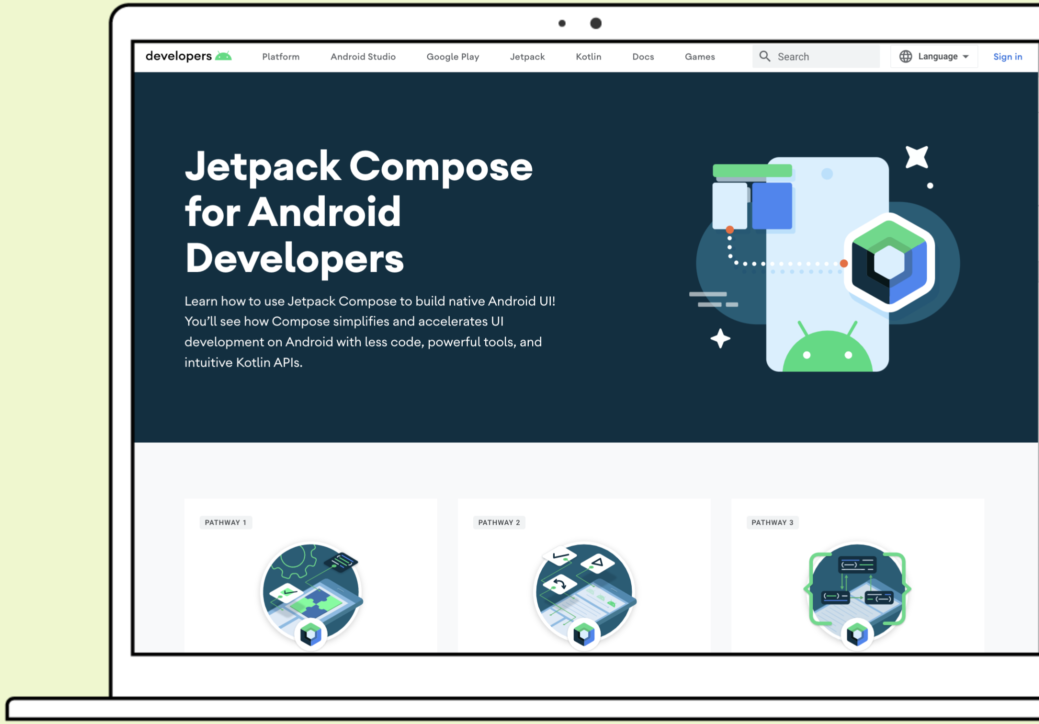
android

Jetpack Compose para Android Developers

4	**Accessibility, testing, and performance**	(2 horas)
5	**Form factors**	(3 horas)

android

Jetpack Compose for Android Developers Course



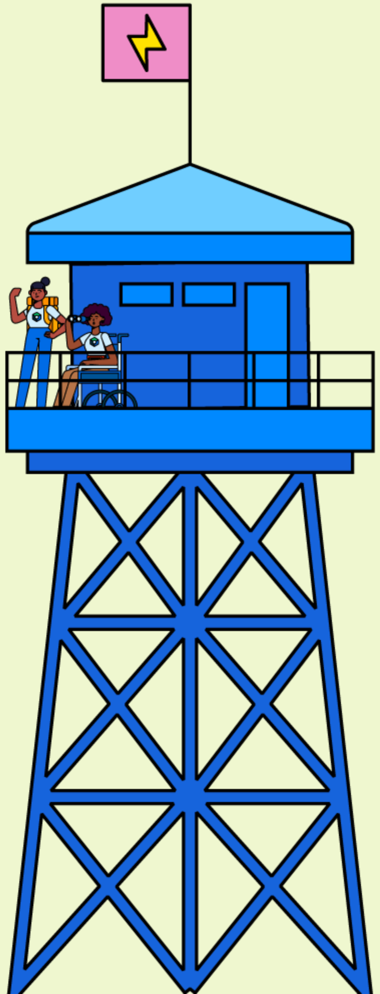
Jetpack Compose for Android Developers

Learn how to use Jetpack Compose to build native Android UI! You'll see how Compose simplifies and accelerates UI development on Android with less code, powerful tools, and intuitive Kotlin APIs.

android

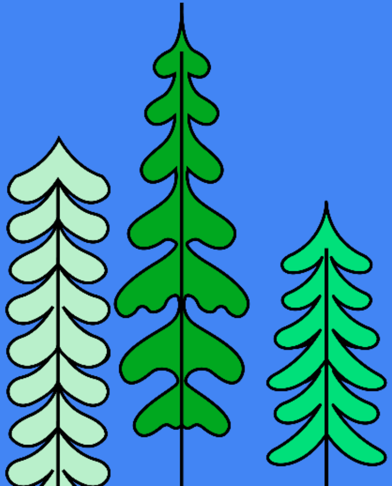
Agenda

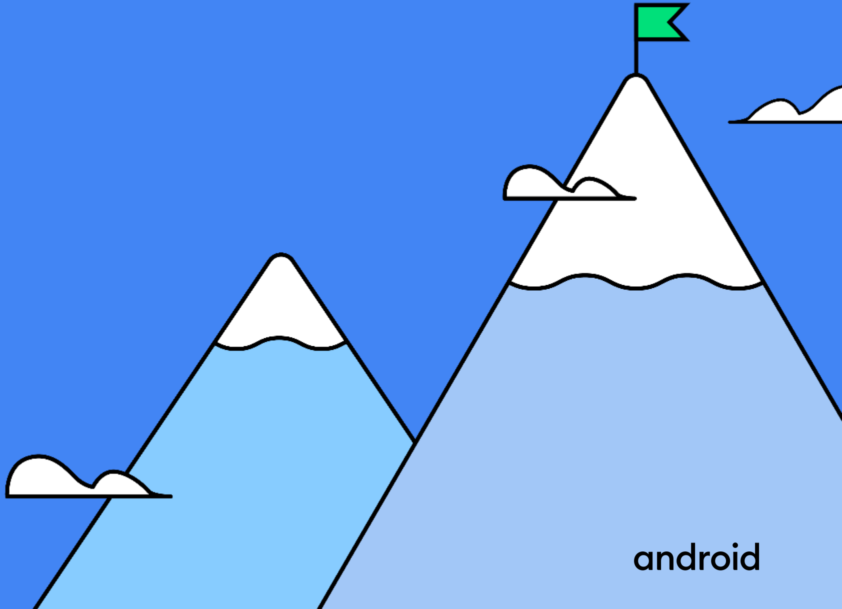
- Pensando en “Compose"
- Funciones ***Composable***
- Compose toolkit
- Tooling (Android Studio)



android

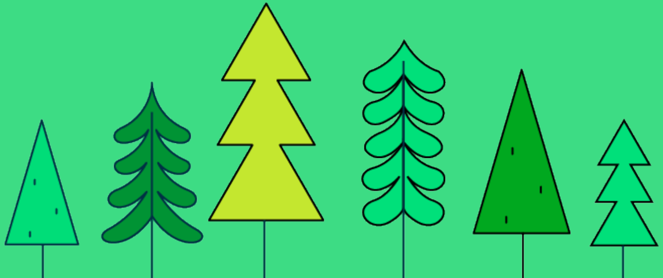
Live demo!





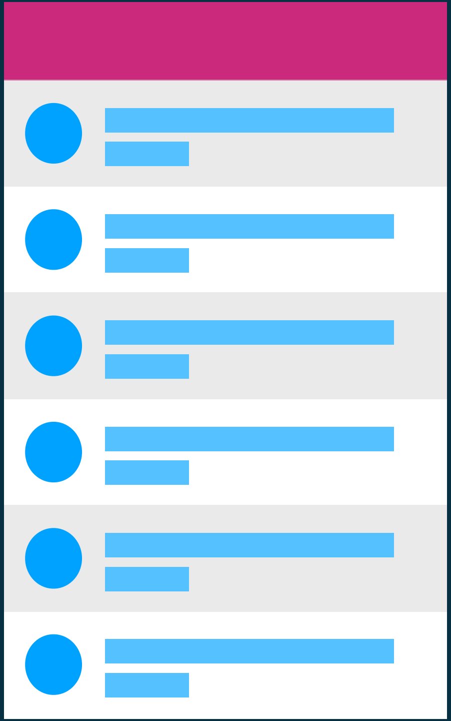
android

Pensando en Compose

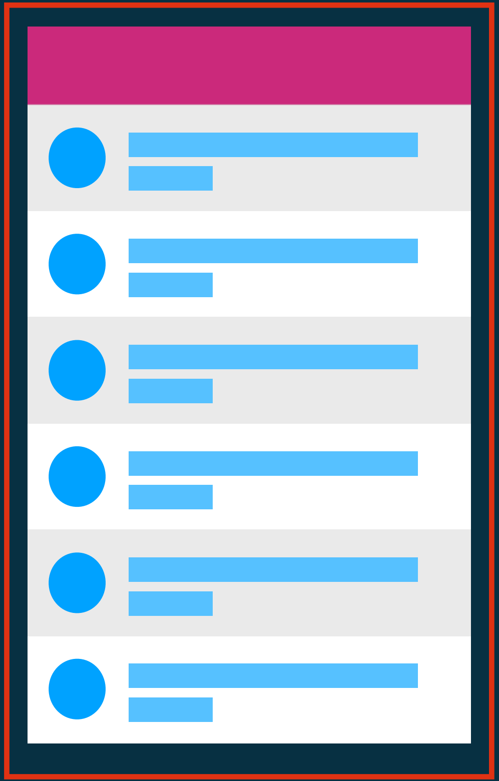


android

Cómo es hoy en día

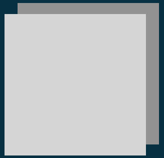


Cómo es hoy en día

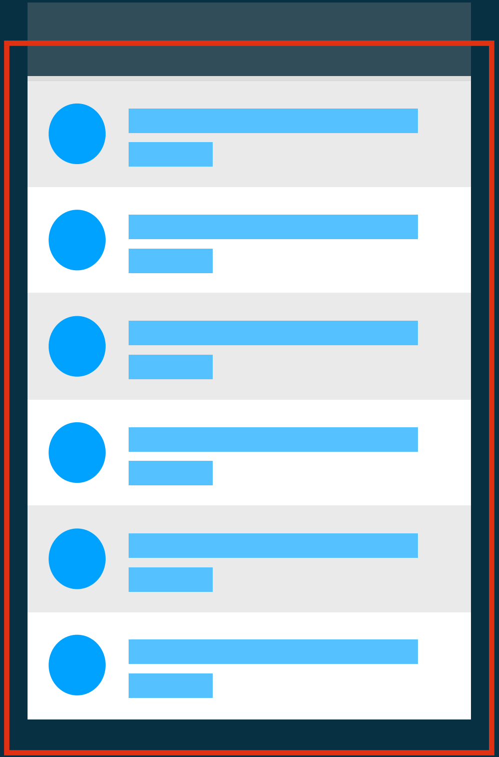


MainActivity.kt

activity_main.xml



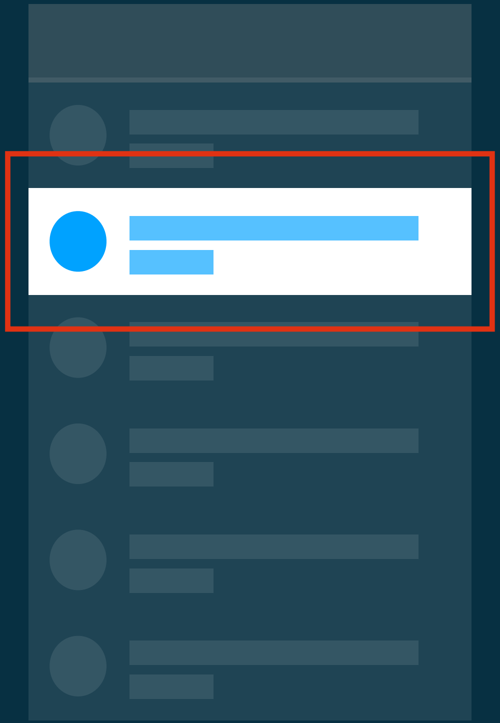
Cómo es hoy en día



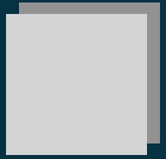
ItemsFragment.kt

fragment_items.xml

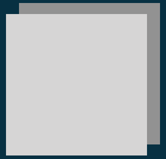
Cómo es hoy en día



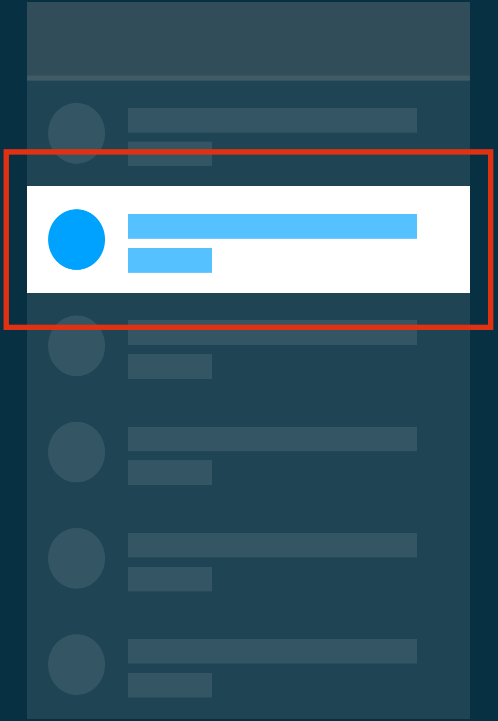
ItemsAdapter.kt



single_item.xml



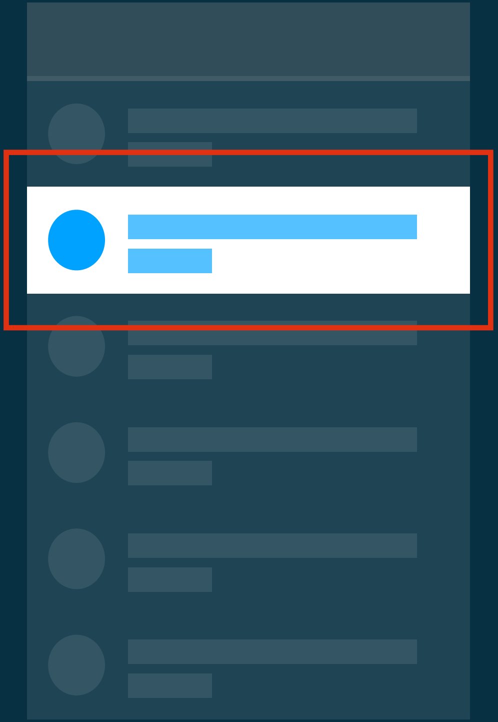
Cómo es hoy en día



ItemsAdapter.kt

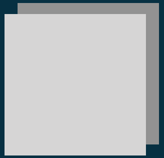
```
class ItemsAdapter: RecyclerView.Adapter {

  fun onCreateViewHolder () { … }

  fun onBindViewHolder () { … }

  fun getItemCount () { … }

  class ItemsViewHolder: RecyclerView.VH() {

     .....

  }

}
```

Cómo es hoy en día

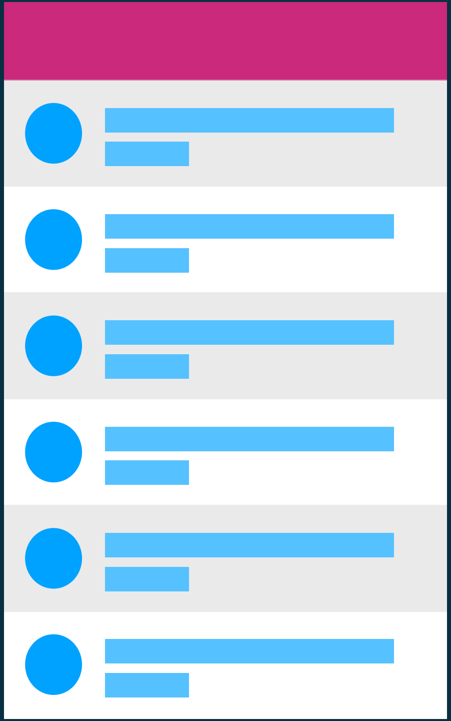


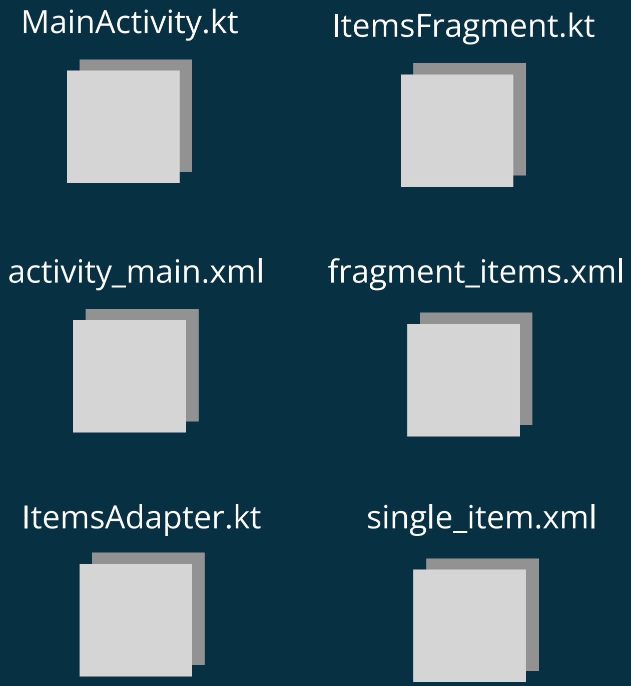
ItemsAdapter.kt

single_item.xml



Cómo es hoy en día

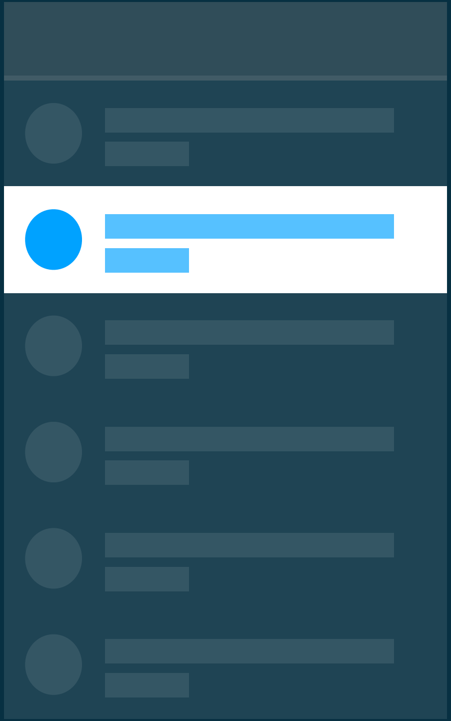




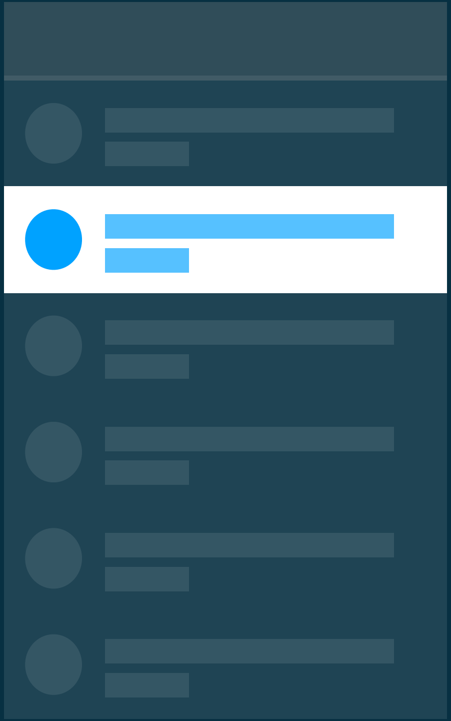
Construye la interfaz describiendo el qué, no el cómo.

android

Usando Compose

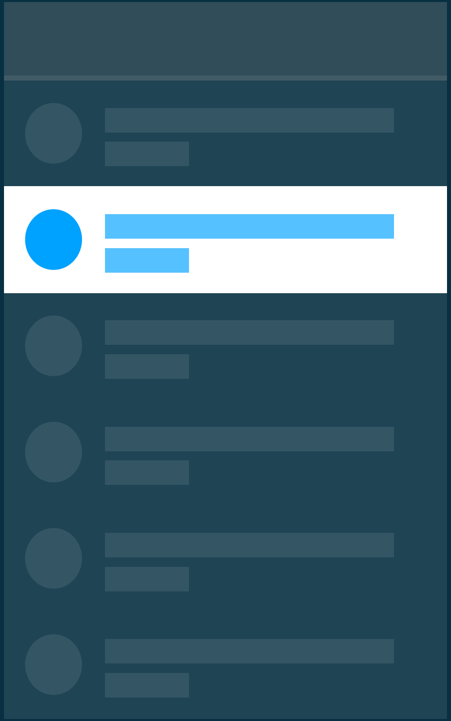


Usando Compose



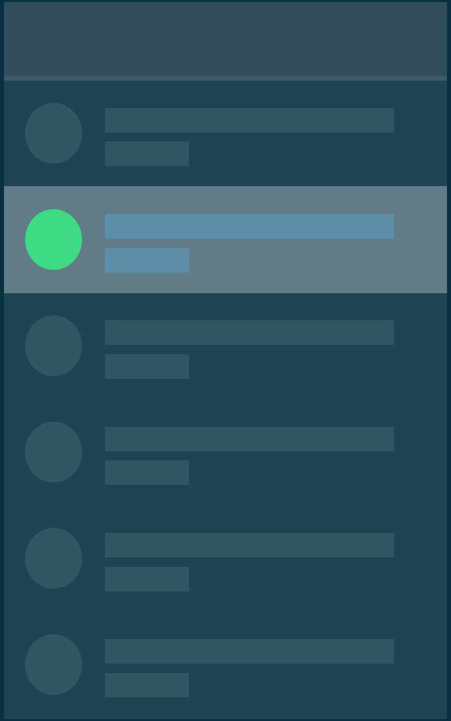
```
@Composable
fun ItemRow(item: Item)
```

Usando Compose



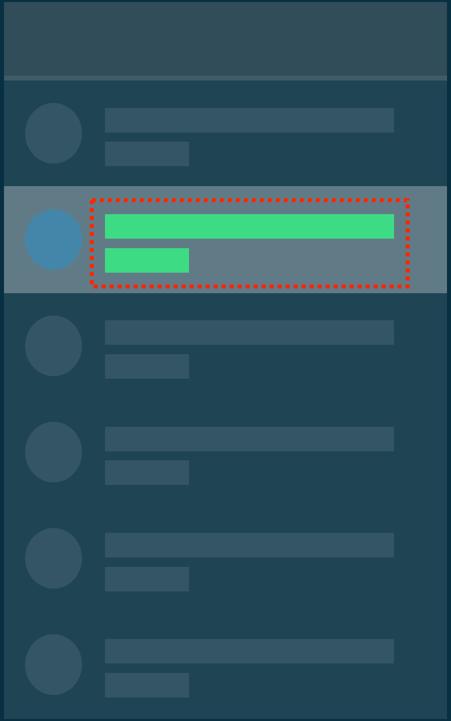
```
@Composable
fun ItemRow(item: Item) {
    Row {



    }
}
```

Usando Compose



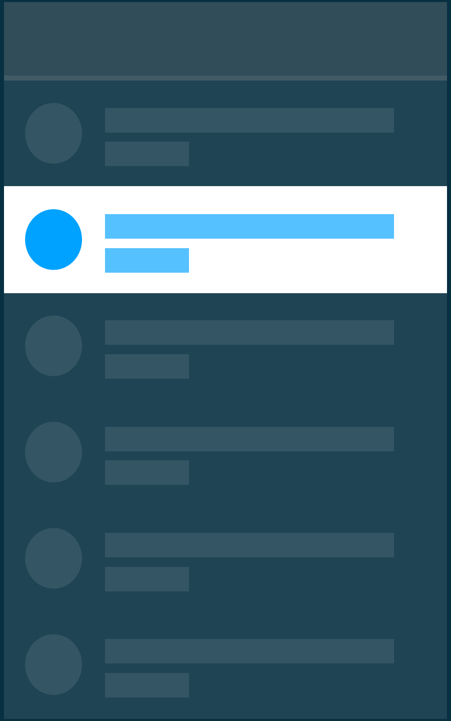
```
@Composable
fun ItemRow(item: Item) {
    Row {
        Image(imageResource(item.image))




    }
}
```

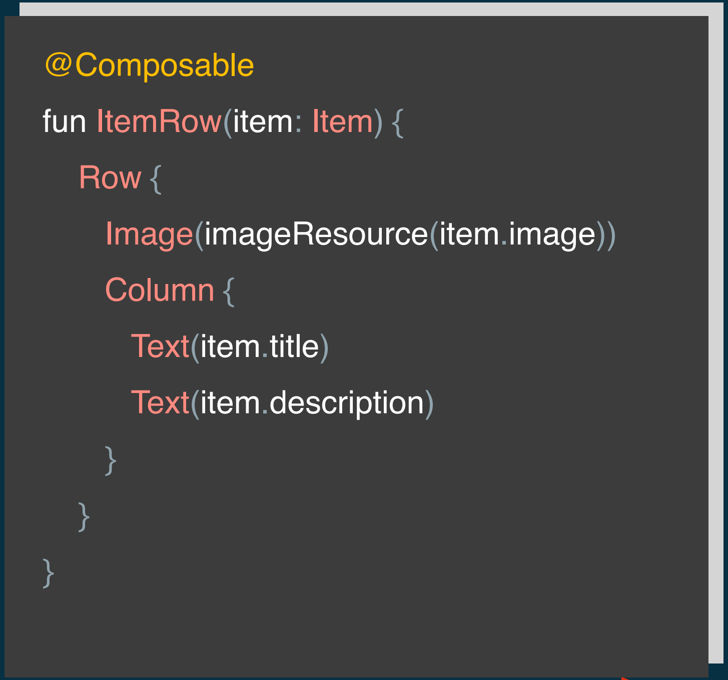
Usando Compose



```
@Composable
fun ItemRow(item: Item) {
    Row {
       Image(imageResource(item.image))
       Column {
          Text(item.title)
          Text(item.description)
       }
    }
}
```

Usando Compose





```
@Composable
fun ItemRow(item: Item) {
  Row {
    Image(imageResource(item.image))
    Column {
      Text(item.title)
      Text(item.description)
    }
  }
}
```

¿Qué podemos ver aquí?

1. Solución 100% orientada a **Kotlin**

```kotlin
@Composable
fun ItemRow(item: Item) {
    Row {
       Image(imageResource(item.image))
       Column {
          Text(item.title)
          Text(item.description)
       }
    }
}
```

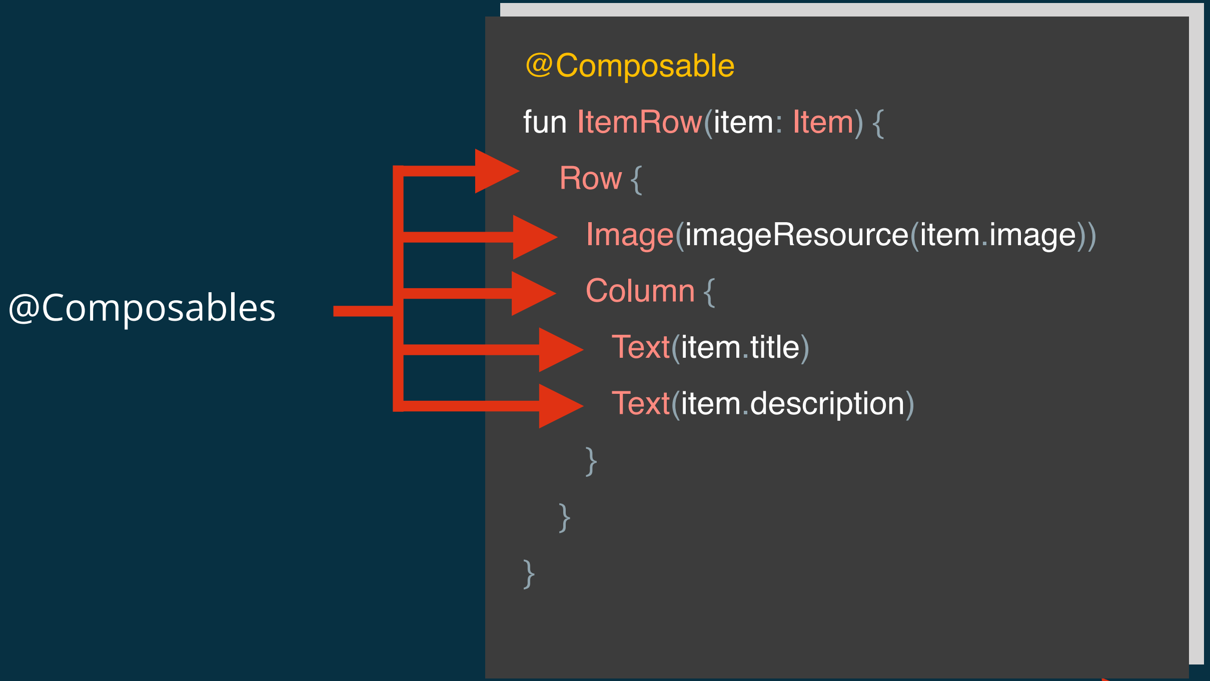
¿Qué podemos ver aquí?

1. Solución 100% orientada a **Kotlin**

2. UI **declarativa,** basada en @Composables

```kotlin
@Composable
fun ItemRow(item: Item) {
    Row {
        Image(imageResource(item.image))
        Column {
            Text(item.title)
            Text(item.description)
        }
    }
}
```

¿Qué podemos ver aquí?

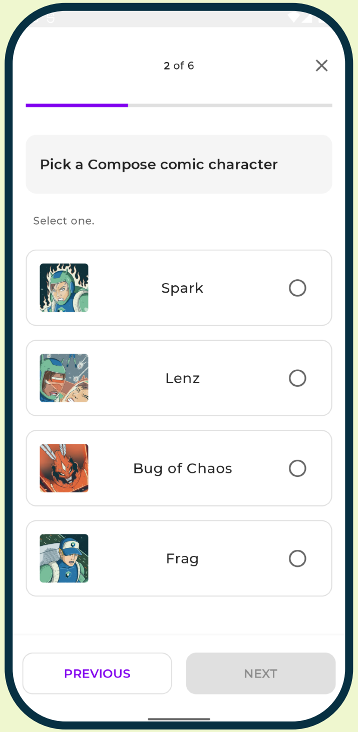


```
@Composable
fun ItemRow(item: Item) {
   Row {
     Image(imageResource(item.image))
     Column {
        Text(item.title)
        Text(item.description)
     }
   }
}
```

¿Qué podemos ver aquí?

1. Solución 100% orientada a **Kotlin**
2. UI **declarativa,** basada en @Composables
3. **Composición** en lugar de Herencia

```
@Composable
fun ItemRow(item: Item) {
    Row {
       Image(imageResource(item.image))
       Column {
          Text(item.title)
          Text(item.description)
       }
    }
}
```



2 of 6

Pick a Compose comic character

Select one.

Spark

Lenz

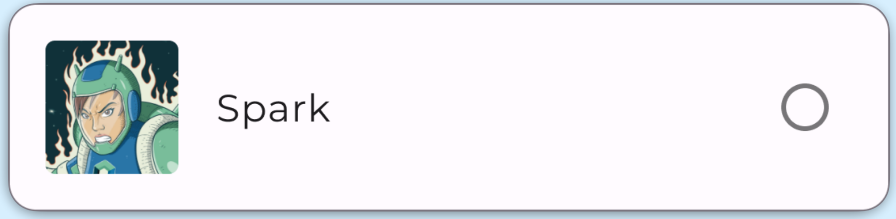
Bug of Chaos

Frag

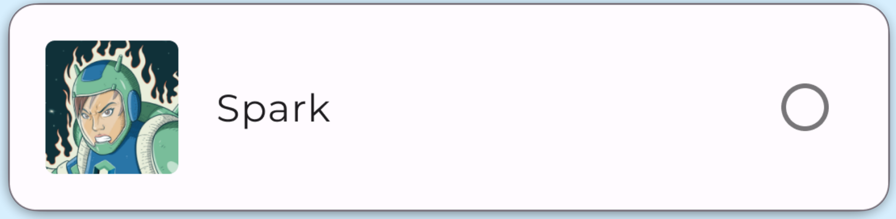
jetsurvey

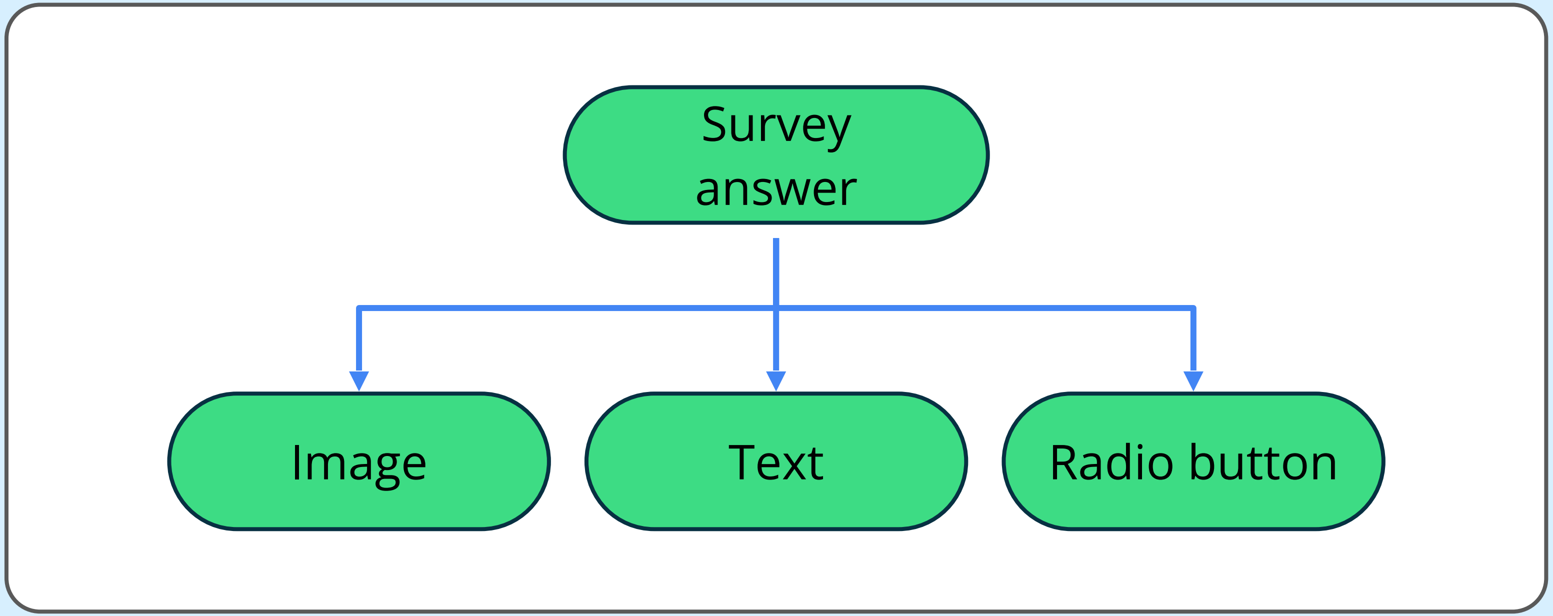
goo.gle/compose-samples

android

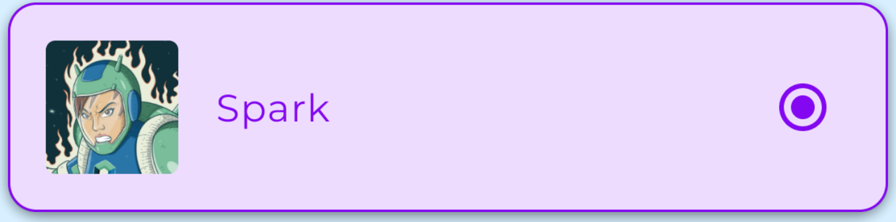

Spark

android



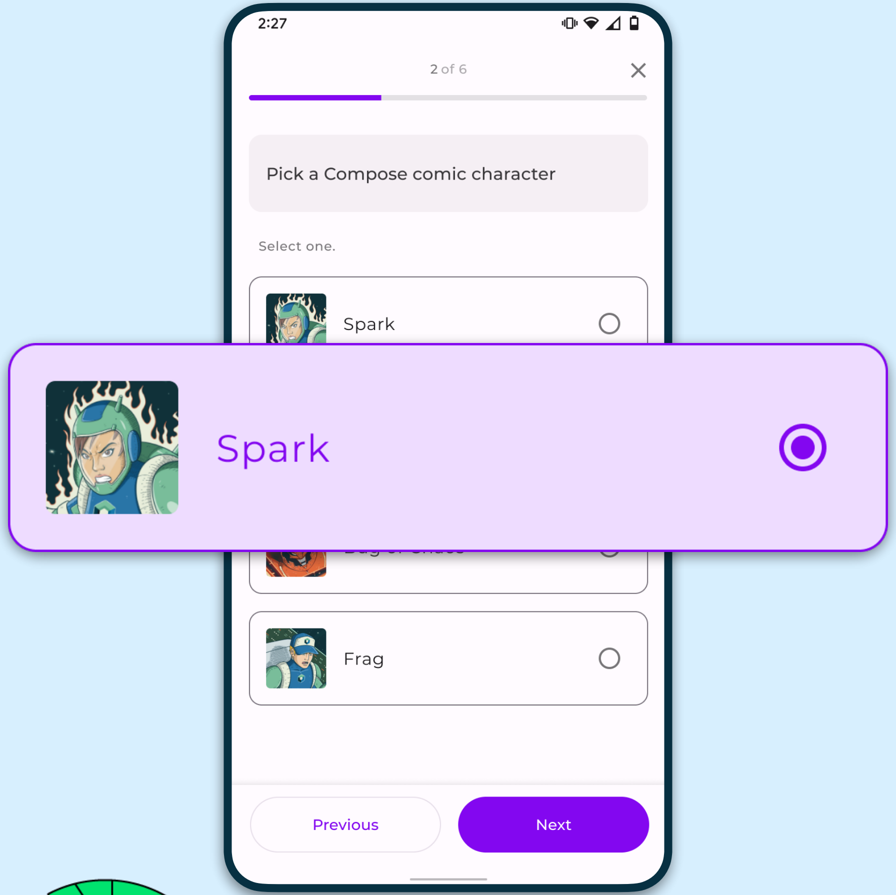


android



Spark ●

android

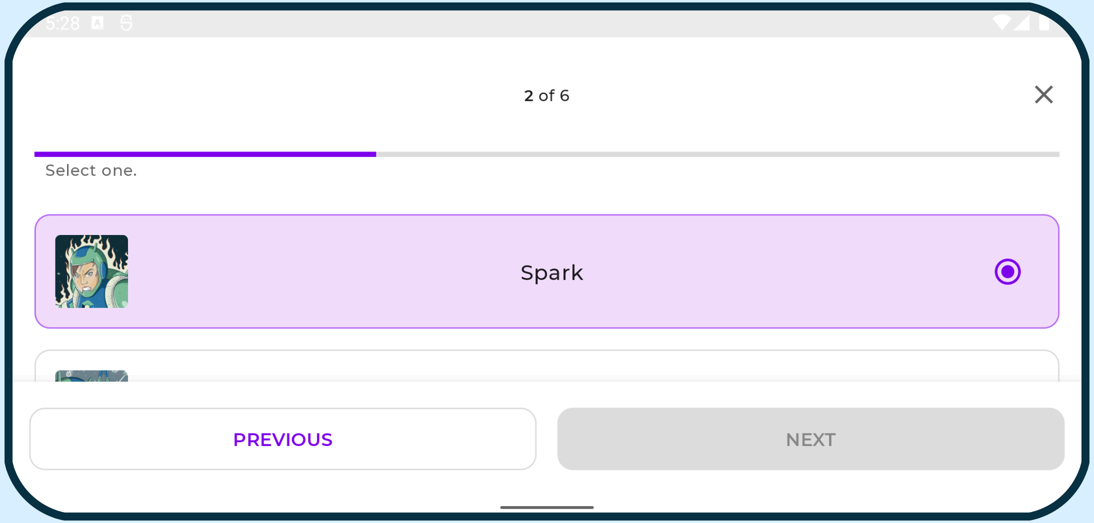

2:27

2 of 6

Pick a Compose comic character

Select one.

android


5:28

2 of 6

Select one.

Spark [✓]

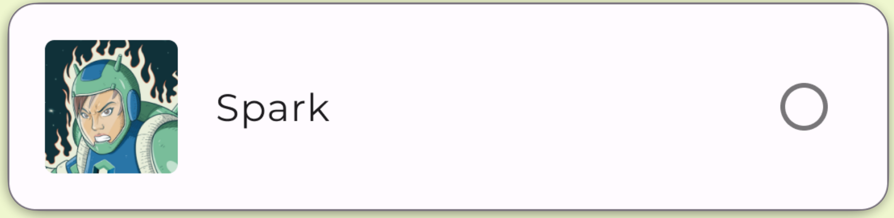
PREVIOUS

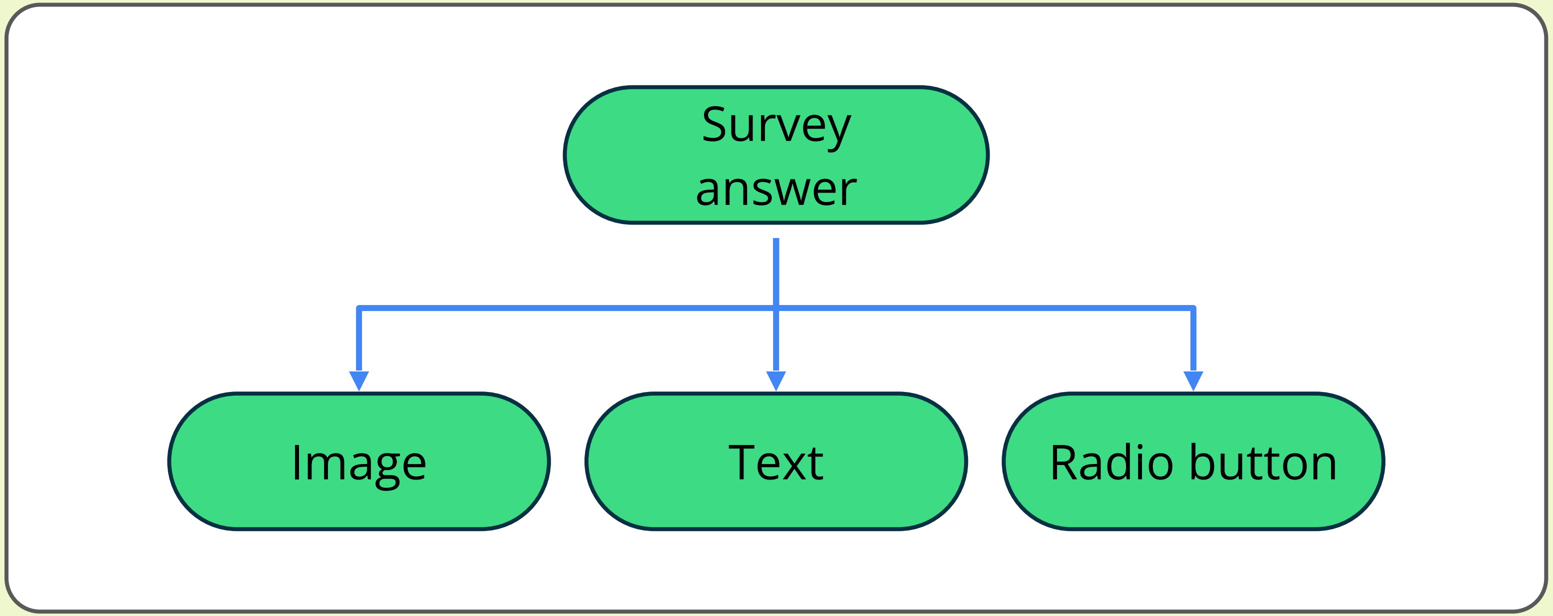
NEXT

android

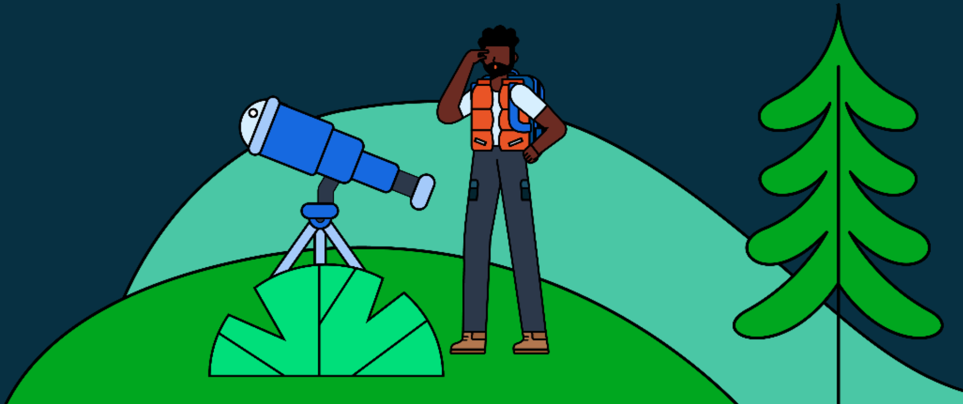
jetpack compose

android


Spark


Survey answer
Image
Text
Radio button

```kotlin
// SurveyAnswer.kt


@Composable
fun SurveyAnswer(answer: Answer) {
    Row {
        Image(answer.image)
        Text(answer.text)
        RadioButton(selected = false, onClick = { /* … */ })
    }
}
```

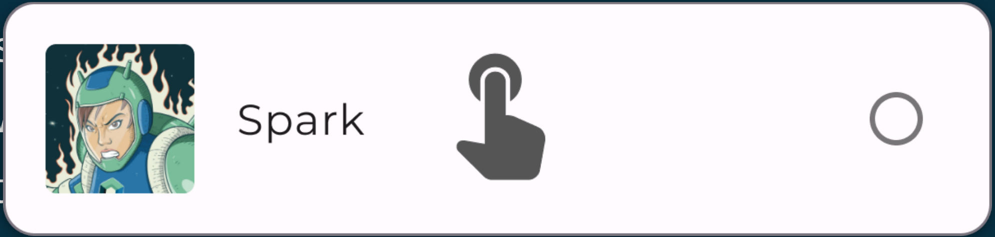


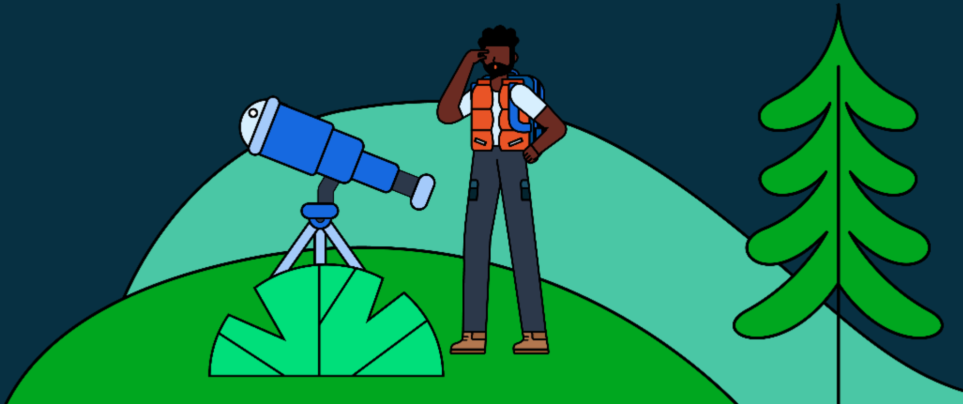
android

```kotlin
// SurveyAnswer.kt

@Composable
fun SurveyAnswer(answer: Answer) {
    Row {
        Image(answer.image)
        Text(answer.text)
        RadioButton(selected = false, onClick = { /* … */ })
    }
}
```

android

```
// SurveyAnswer.kt


@Composable
fun SurveyAnswer(answer: Answer) {
    Row {
        Image(ans
        Text(answ
        RadioButt
    }
}
```

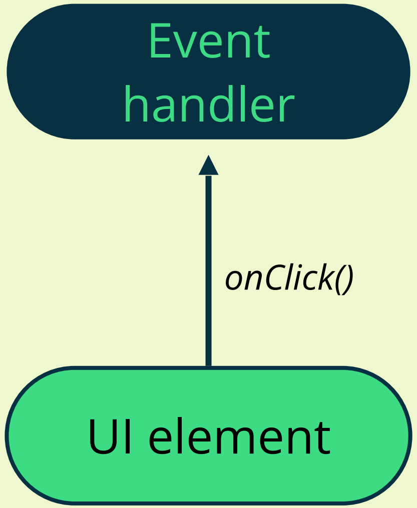


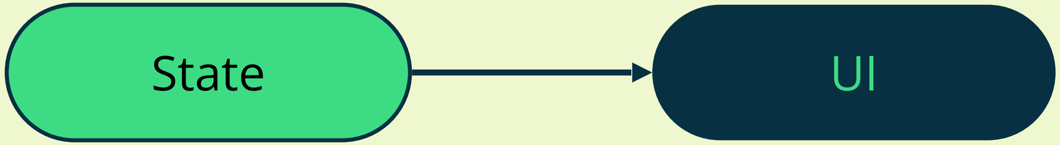


android

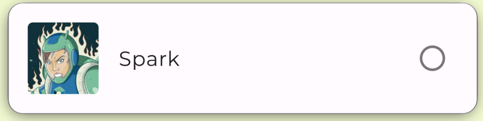
Construye la interfaz describiendo el qué, no el cómo.

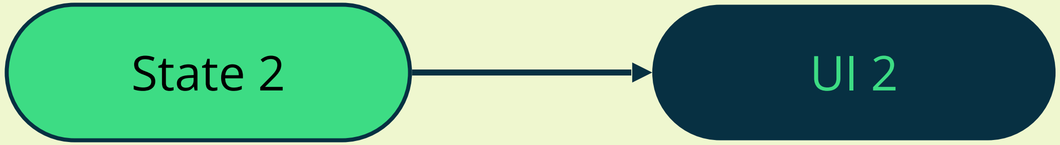
android


Event handler
onClick()
UI element

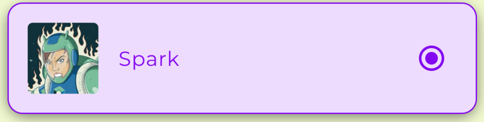


selected = false





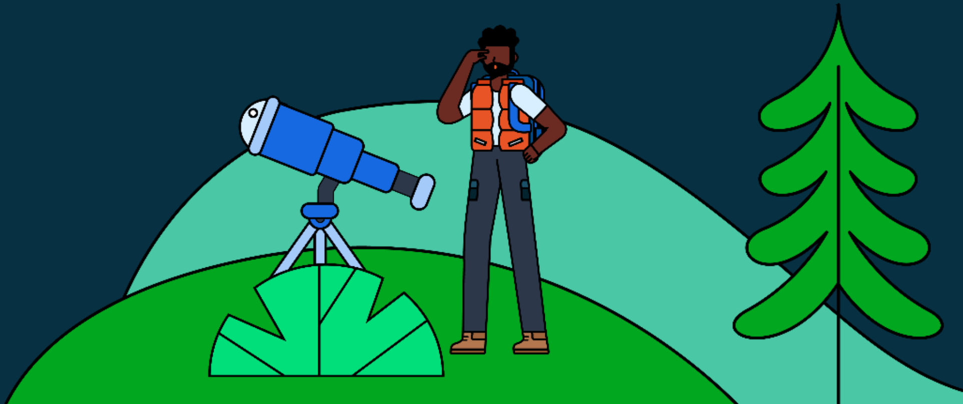
selected = true



android

```
// SurveyAnswer.kt

@Composable
fun SurveyAnswer(answer: Answer) {
 Row {
   / * ... */
   var selected: Boolean = // ...
   RadioButton(selected, onClick = { /* ... */ })
 }
}
```

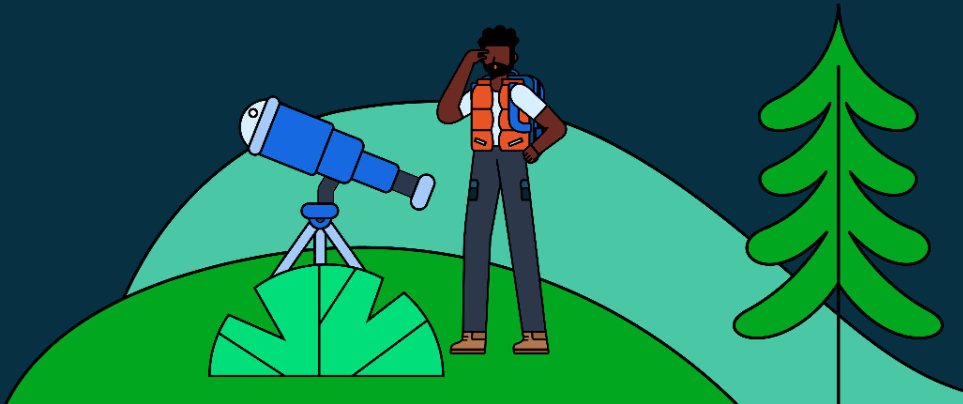
Los Estados controlan la UI



android

```
// SurveyAnswer.kt


@Composable
fun SurveyAnswer(answer: Answer) {
  Row {
    / * ... */
    var selected: Boolean = // ...
    RadioButton(selected, onClick = {
        selected = !selected
    })
  }
}
```

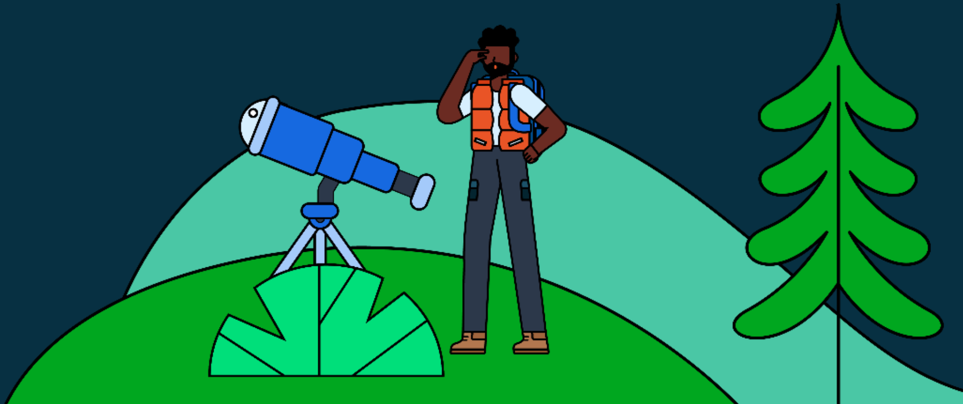
Los Eventos controlan el estado



android

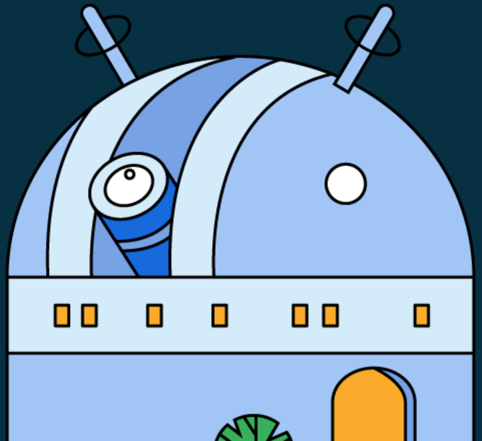
// SurveyAnswer.kt

```
@Composable
fun SurveyAnswer(answer: Answer) {
  Row {
    / * ... */
    var selected: Boolean = // ...
    RadioButton(selected, onClick = {
        selected = !selected
    })
  }
}
```

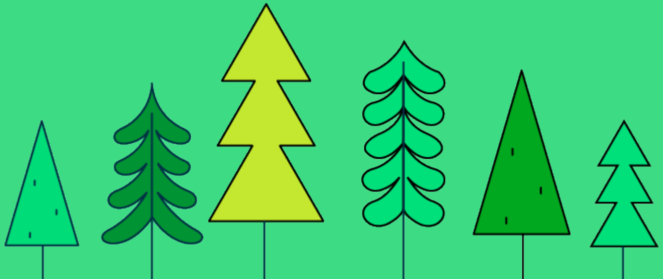


android

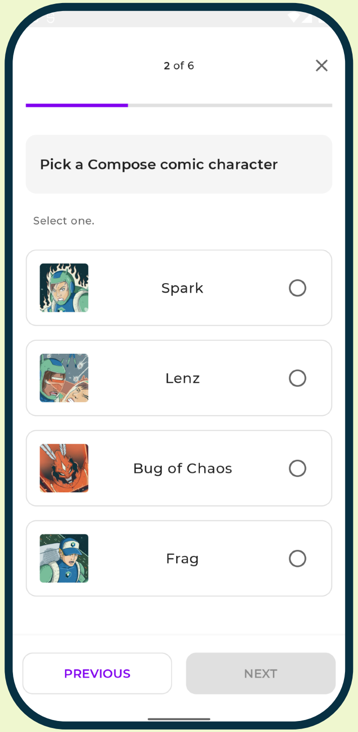
1. Describe el qué, no el cómo
2. Elementos de UI = Funciones
3. Los Estados controlan la UI
4. Los Eventos controlan el State



Composable functions



android



2 of 6

Pick a Compose comic character

Select one.

Spark ○

Lenz ○

Bug of Chaos ○

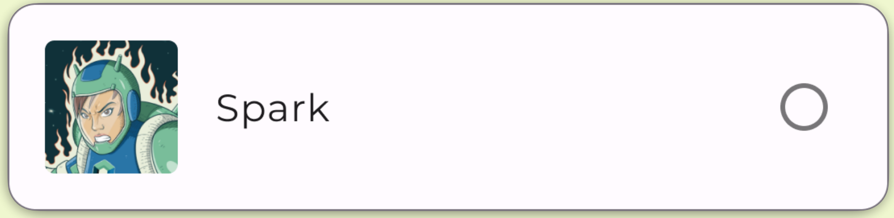
Frag ○

PREVIOUS NEXT

jetsurvey

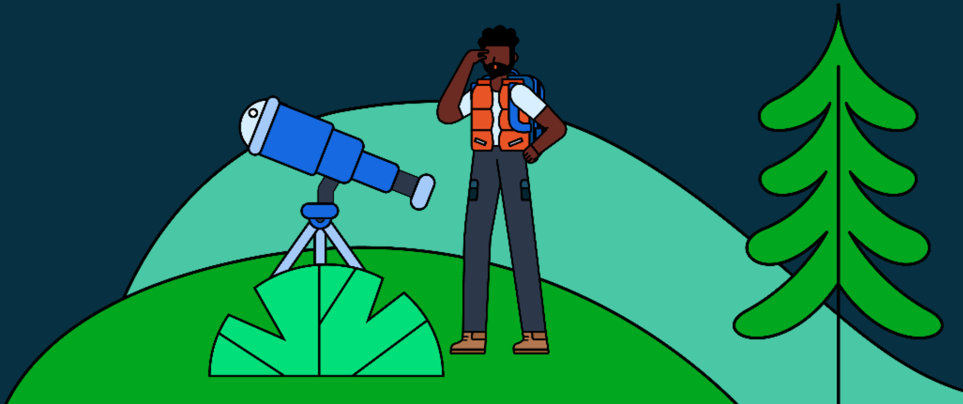
goo.gle/compose-samples

android



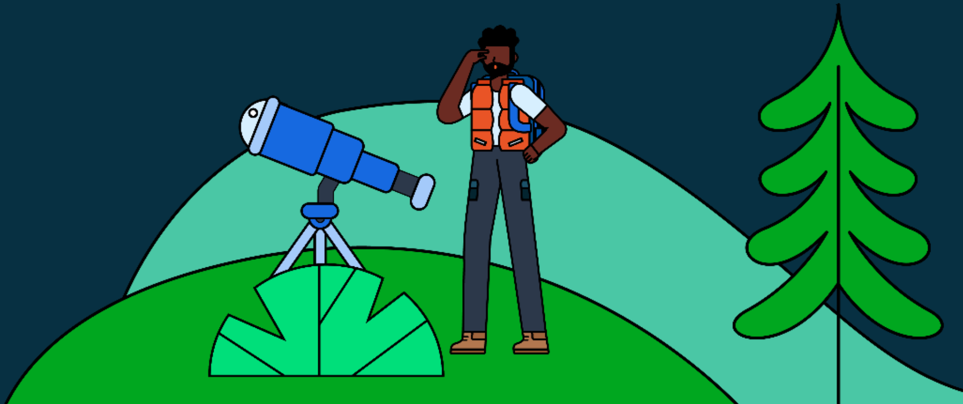
android

```
// SurveyAnswer.kt

@Composable
fun SurveyAnswer(answer: Answer) {
    Row {
        Image(answer.image)
        Text(answer.text)
        RadioButton(false, onClick = { /* … */ })
    }
}
```



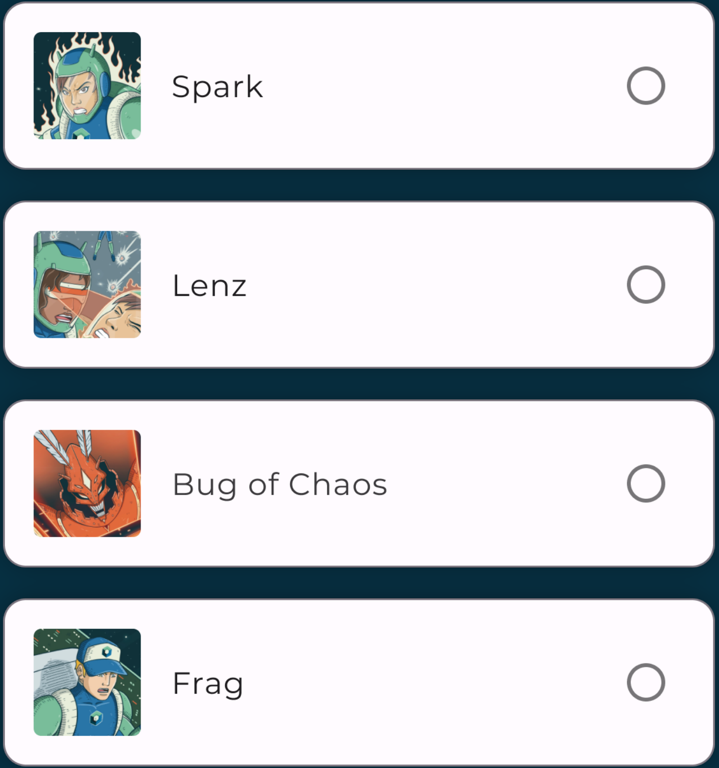
android

```
// SurveyAnswer.kt

@Composable
fun SurveyAnswer(answer: Answer) { /* … */ }


@Composable
fun SingleChoiceQuestion(answers: List<Answer>) {
  Column {
    answers.forEach { answer ->
      SurveyAnswer(answer = answer)
    }
  }
}
```

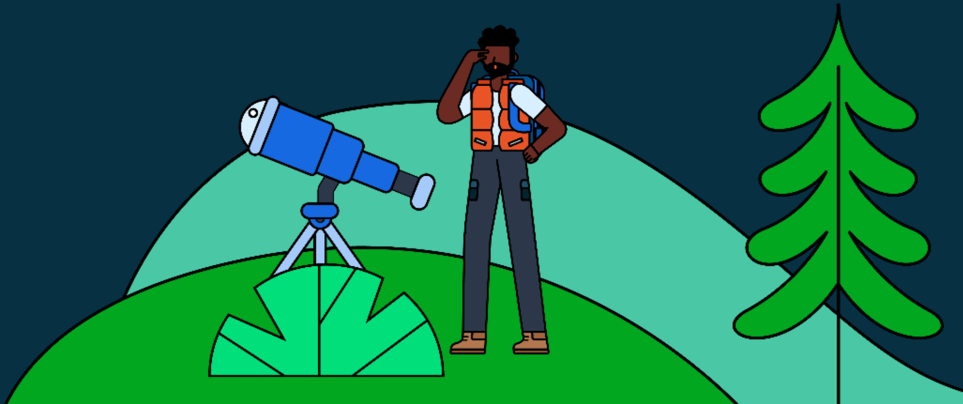


android

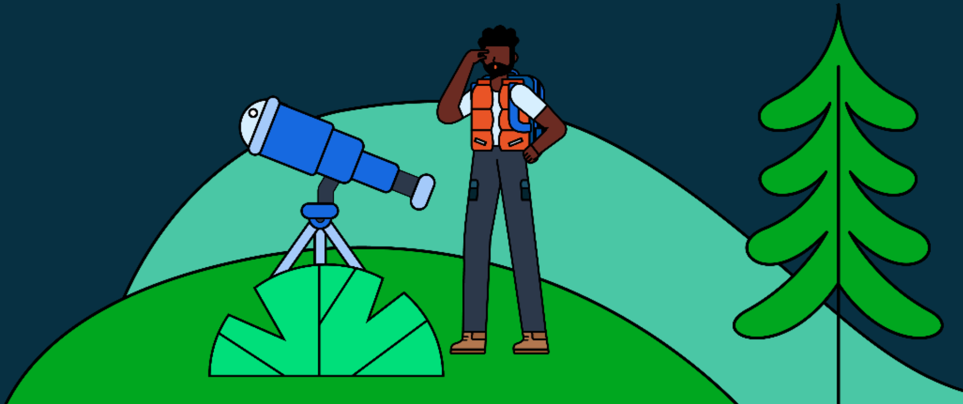
// SingleChoiceQuestion.kt



- Spark ○
- Lenz ○
- Bug of Chaos ○
- Frag ○



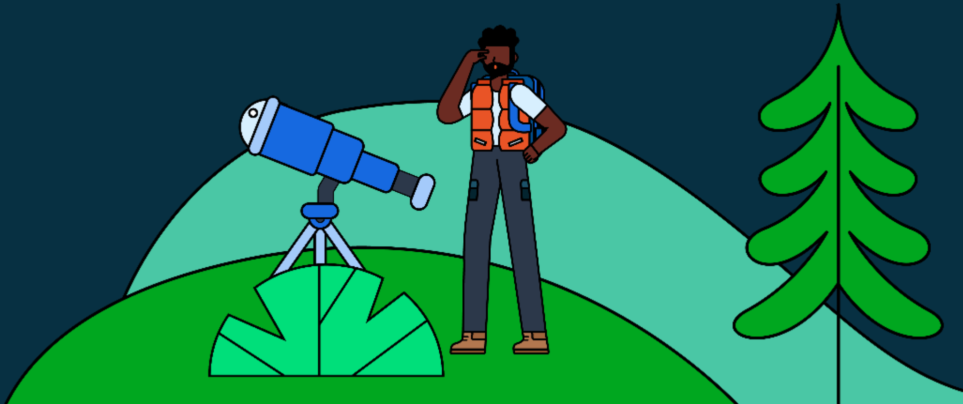
```
// SingleChoiceQuestion.kt

@Composable
fun SingleChoiceQuestion(answers: List<Answer>) {
  Column {
    answers.forEach { answer ->
      SurveyAnswer(answer = answer)
    }
  }
}
```



android

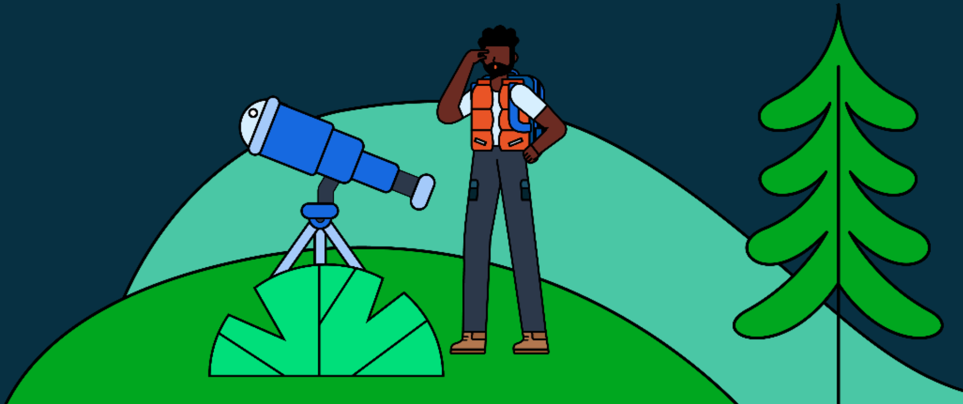
// SingleChoiceQuestion.kt

```
@Composable
fun SingleChoiceQuestion(answers: List<Answer>) {
  Column {
    answers.forEach { answer ->
      // No debemos hacer esto!!
      val answer = SurveyAnswer(answer = answer)
    }
  }
}
```



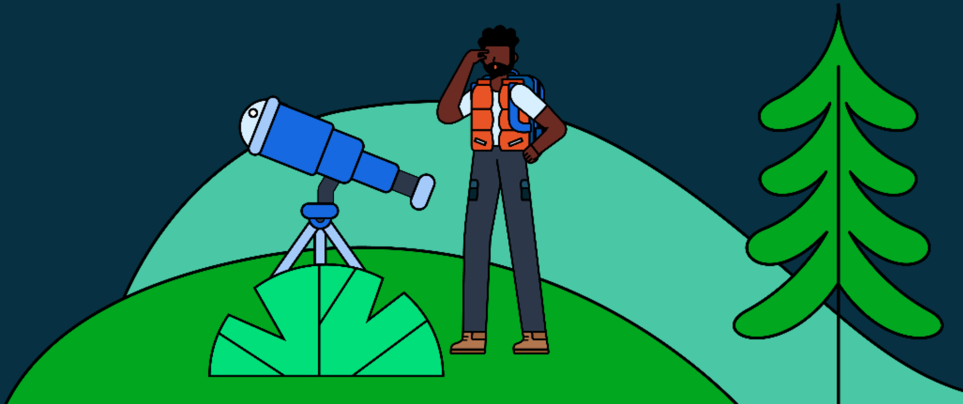
android

```
// SingleChoiceQuestion.kt

@Composable
fun SingleChoiceQuestion(answers: List<Answer>) {
  Column {
    if (answers.isEmpty()) {
      Text("No hay opciones!")
    } else {
      answers.forEach { answer ->
        SurveyAnswer(answer = answer)
      }
    }
  }
}
```



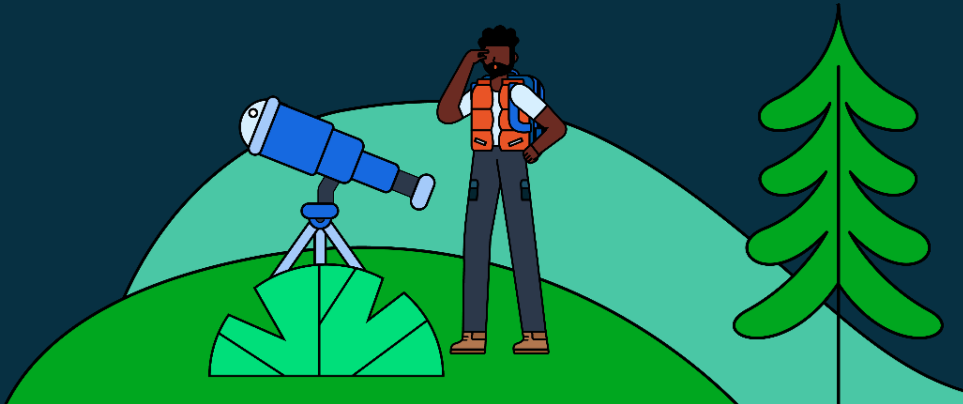
android

```
// SingleChoiceQuestion.kt

@Composable
fun SingleChoiceQuestion(answers: List<Answer>) {
  // Rápido y sin efectos secundarios
  Column {
    if (answers.isEmpty()) { /* ... */ }
    else { /* ... */ }
  }
}
```

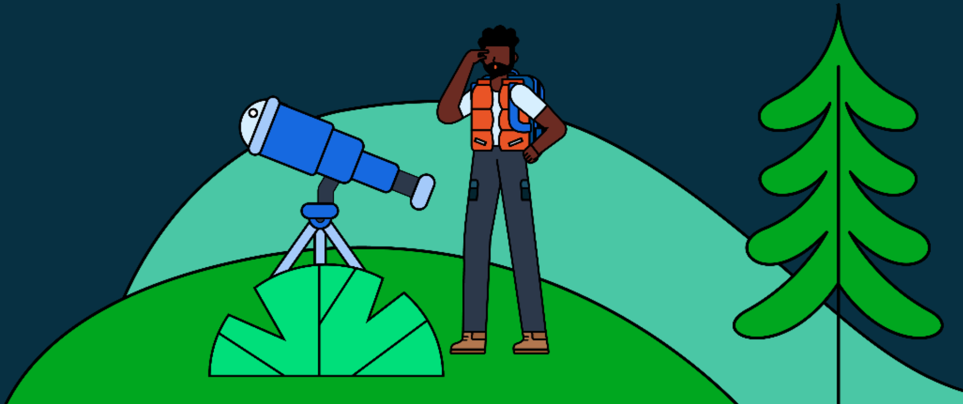


android

```
// SingleChoiceQuestion.kt

@Composable
fun SingleChoiceQuestion(answers: List<Answer>) {
  // No debería haber efectos secundarios
  SurveyApp.didShowSingleChoiceQuestion = true
  Column {
    if (answers.isEmpty()) { /* ... */ }
    else { /* ... */ }
  }
}
```

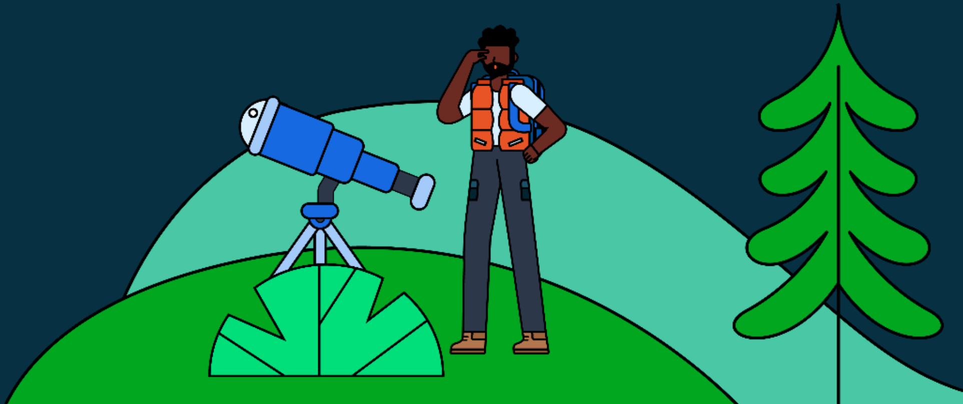


```kotlin
// SingleChoiceQuestion.kt

@Composable
fun SingleChoiceQuestion(answers: List<Answer>) {
  Column {
    answers.forEach { answer ->
      SurveyAnswer(answer = answer)
    }
  }
}
```



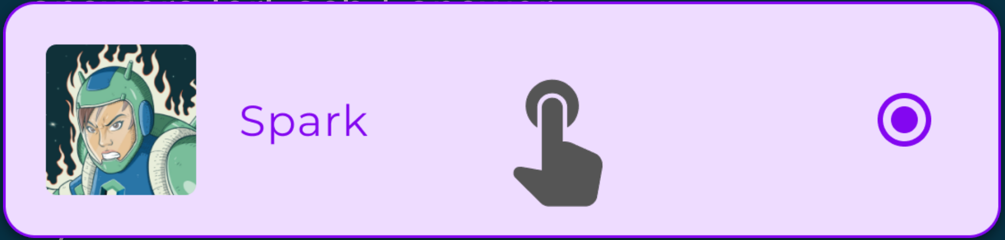
android

```
// SingleChoiceQuestion.kt

@Composable
fun SingleChoiceQuestion(answers: List<Answer>) {
  answers.forEach { answer ->
    SurveyAnswer(
      answer = answer,
      isSelected = false,
    )
  }
}
```

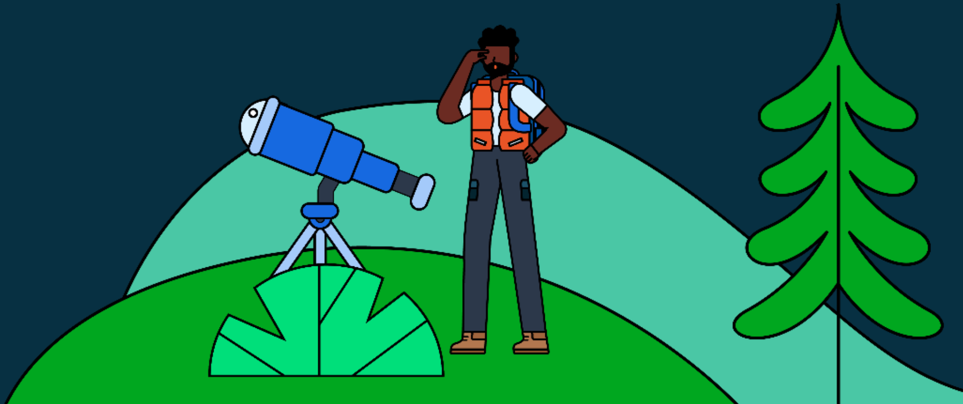


android

```
// SingleChoiceQuestion.kt

@Composable
fun SingleChoiceQuestion(answers: List<Answer>) {
```

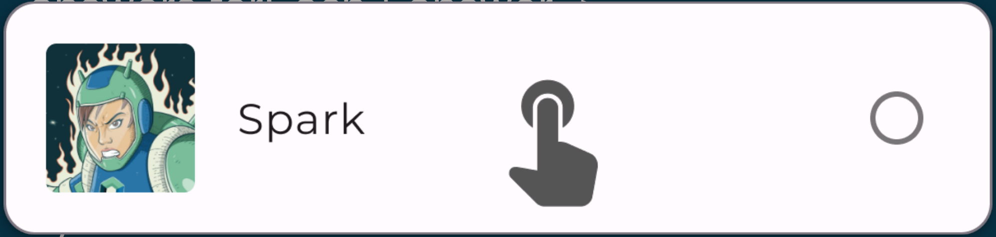


```
  }
}
```

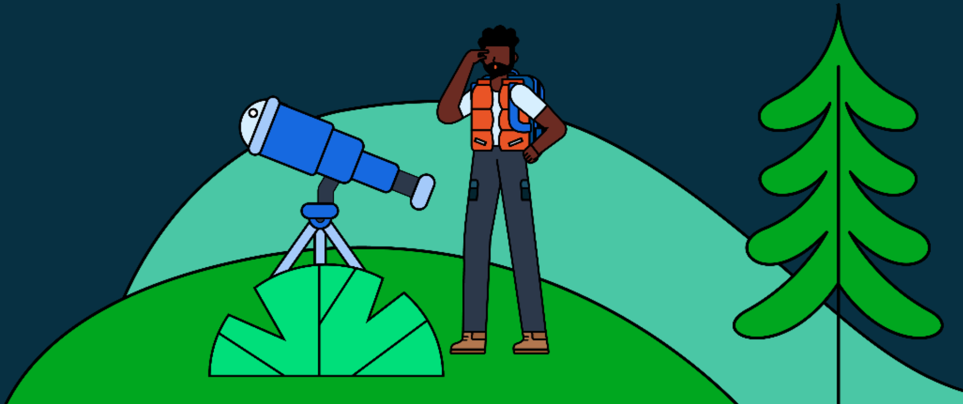


android

```
// SingleChoiceQuestion.kt

@Composable
fun SingleChoiceQuestion(answers: List<Answer>) {
  answers.forEach { answer ->
```

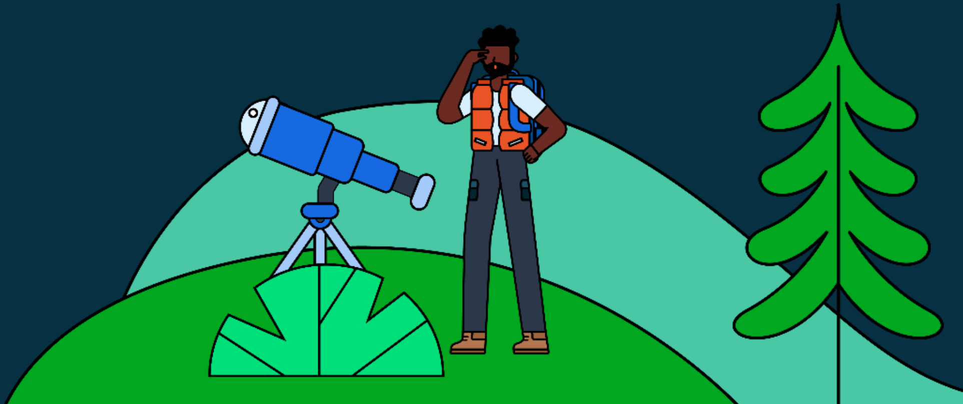


```
    )
  }
}
```



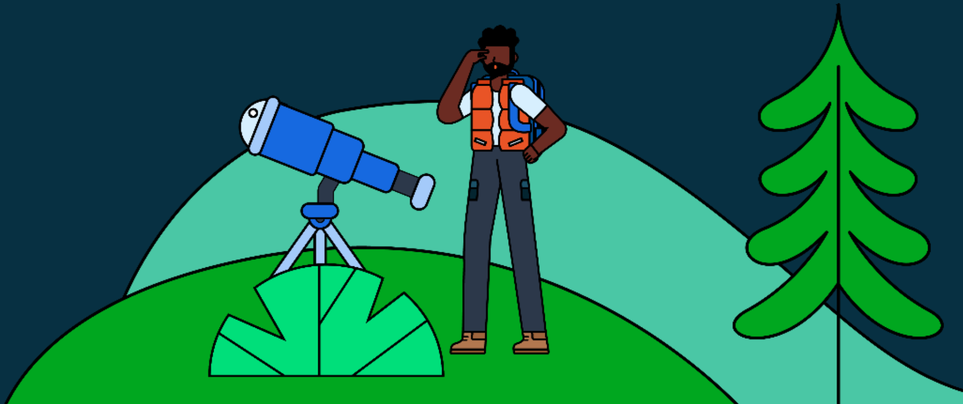
android

```kotlin
// SingleChoiceQuestion.kt

@Composable
fun SingleChoiceQuestion(answers: List<Answer>) {
  var selectedAnswer: Answer? = null
  answers.forEach { answer ->
    SurveyAnswer(
      answer = answer,
      isSelected = false,
    )
  }
}
```



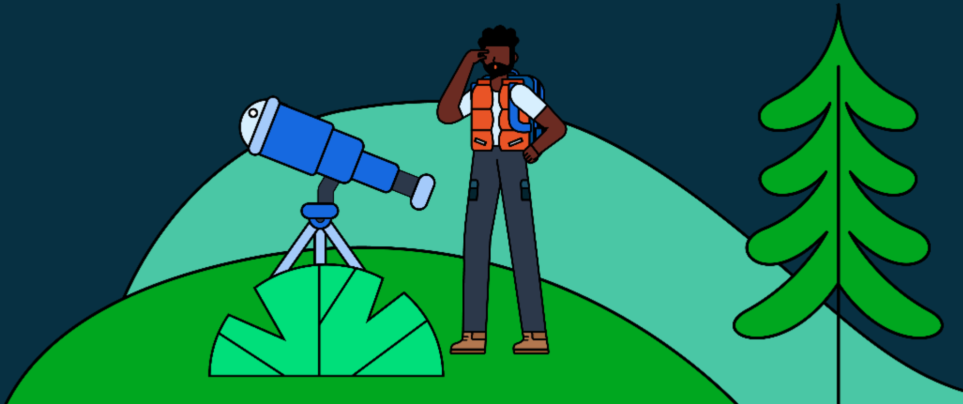
android

```
// SingleChoiceQuestion.kt

@Composable
fun SingleChoiceQuestion(answers: List<Answer>) {
  var selectedAnswer: Answer? = null
  answers.forEach { answer ->
    SurveyAnswer(
      answer = answer,
      isSelected = false,
    )
  }
}
```



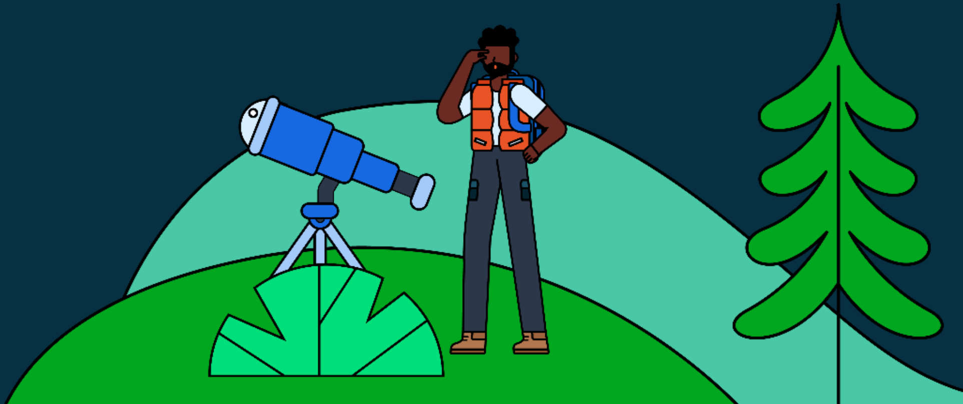
android

```kotlin
// SingleChoiceQuestion.kt

@Composable
fun SingleChoiceQuestion(answers: List<Answer>) {
  var selectedAnswer: MutableState<Answer?> =
    mutableStateOf(null)
  answers.forEach { answer ->
    SurveyAnswer(
      answer = answer,
      isSelected = false,
    )
  }
}
```



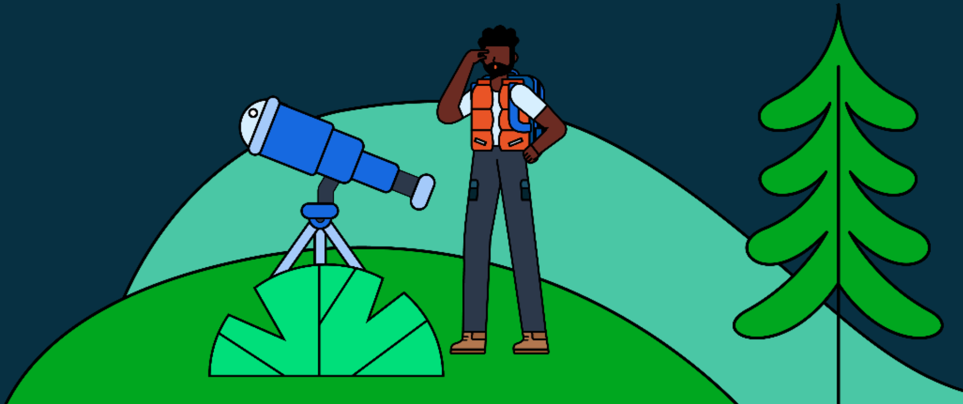
android

```
// SingleChoiceQuestion.kt

@Composable
fun SingleChoiceQuestion(answers: List<Answer>) {
  var selectedAnswer: MutableState<Answer?> =
    mutableStateOf(null)
  answers.forEach { answer ->
    SurveyAnswer(
      answer = answer,
      isSelected = false,
    )
  }
}
```



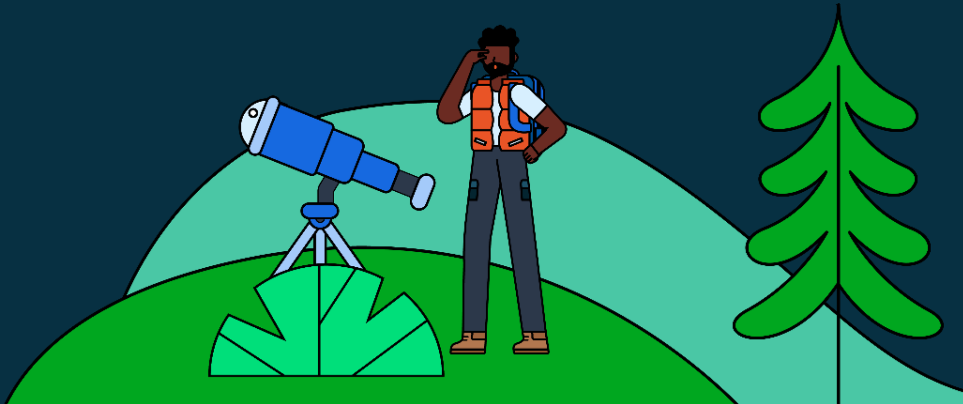
android

```
// SingleChoiceQuestion.kt

@Composable
fun SingleChoiceQuestion(answers: List<Answer>) {
  var selectedAnswer: MutableState<Answer?> =
    mutableStateOf(null)
  answers.forEach { answer ->
    SurveyAnswer(
      answer = answer,
      isSelected = (selectedAnswer.state == answer),
    )
  }
}
```



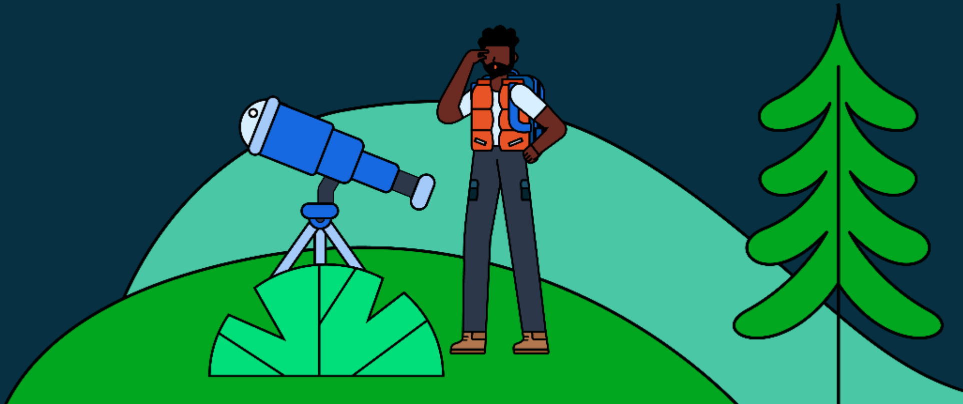
android

```
// SingleChoiceQuestion.kt

@Composable
fun SingleChoiceQuestion(answers: List<Answer>) {
  var selectedAnswer: MutableState<Answer?> =
    remember { mutableStateOf(null) }
  answers.forEach { answer ->
    SurveyAnswer(
      answer = answer,
      isSelected = (selectedAnswer.state == answer),
    )
  }
}
```



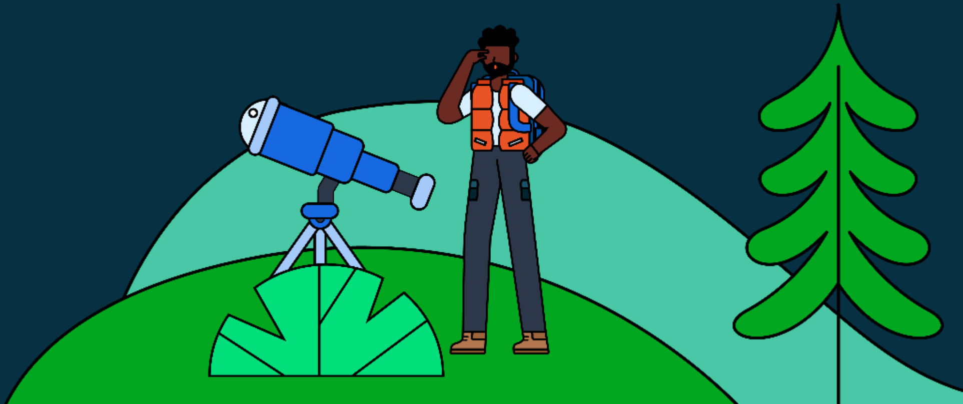
android

```kotlin
// SingleChoiceQuestion.kt

@Composable
fun SingleChoiceQuestion(answers: List<Answer>) {
  var selectedAnswer: MutableState<Answer?> =
    rememberSaveable { mutableStateOf(null) }
  answers.forEach { answer ->
    SurveyAnswer(
      answer = answer,
      isSelected = (selectedAnswer.state == answer),
    )
  }
}
```



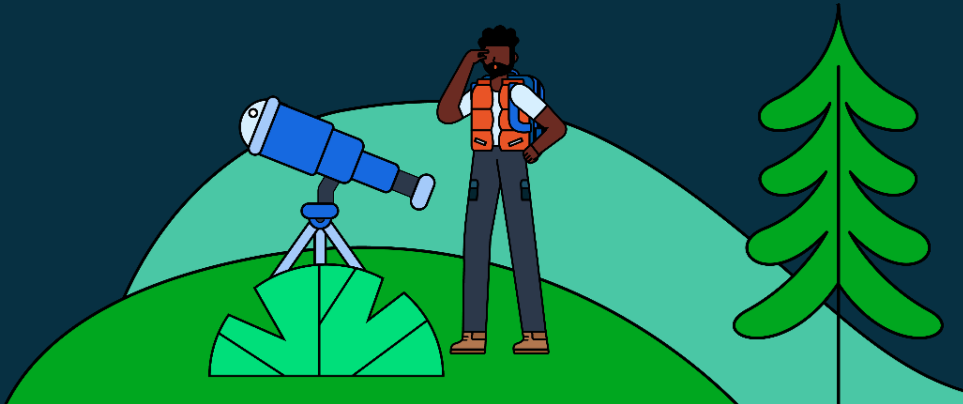
android

```
// SingleChoiceQuestion.kt

@Composable
fun SingleChoiceQuestion(answers: List<Answer>) {
  var selectedAnswer: Answer? by
    rememberSaveable { mutableStateOf(null) }
  answers.forEach { answer ->
    SurveyAnswer(
      answer = answer,
      isSelected = (selectedAnswer.state == answer),
    )
  }
}
```



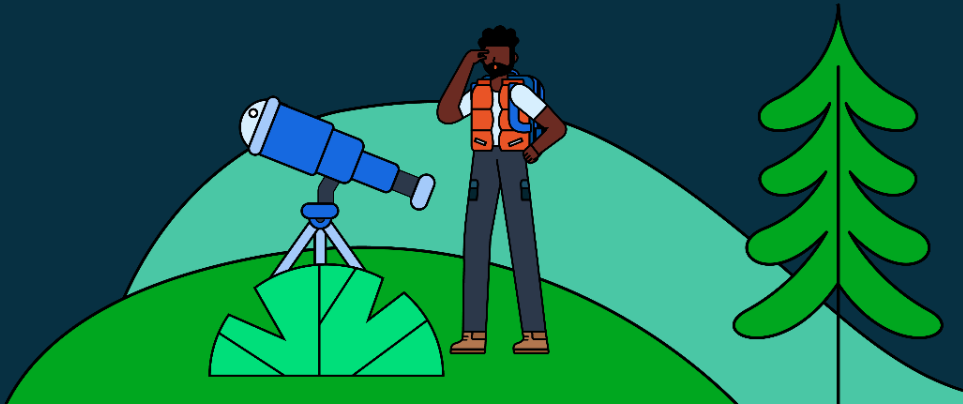
android

```kotlin
// SingleChoiceQuestion.kt

@Composable
fun SingleChoiceQuestion(answers: List<Answer>) {
  var selectedAnswer: Answer? by
    rememberSaveable { mutableStateOf(null) }
  answers.forEach { answer ->
    SurveyAnswer(
      answer = answer,
      isSelected = (selectedAnswer == answer),
    )
  }
}
```



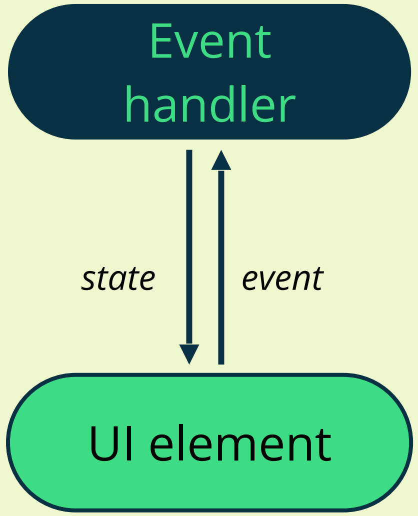
android

```kotlin
// SingleChoiceQuestion.kt

@Composable
fun SingleChoiceQuestion(answers: List<Answer>) {
  var selectedAnswer: Answer? by
    rememberSaveable { mutableStateOf(null) }
  answers.forEach { answer ->
    SurveyAnswer(
      answer = answer,
      isSelected = (selectedAnswer == answer),
      onAnswerSelected = { answer -> selectedAnswer = answer }
    )
  }
}
```



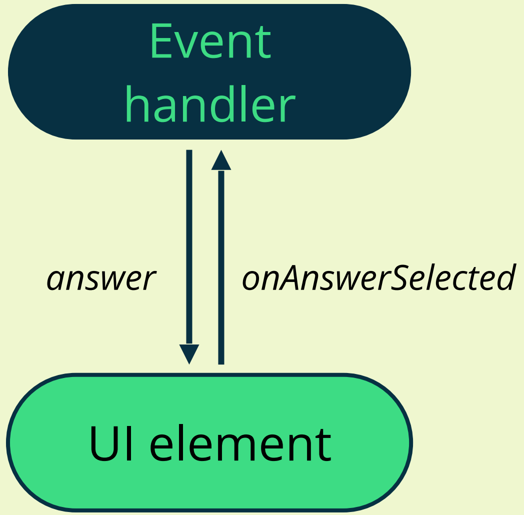
android

Events change State



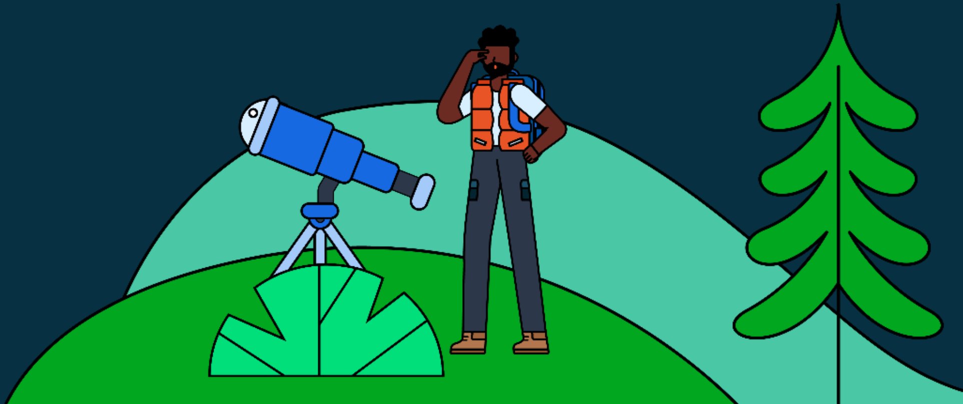
android

Events change State



android

```kotlin
// SingleChoiceQuestion.kt

@Composable
fun SingleChoiceQuestion(answers: List<Answer>) {
  val selectedAnswer: Answer? by
    rememberSaveable { mutableStateOf<Answer>(null) }
  answers.forEach { answer ->
    SurveyAnswer(
      answer = answer,
      isSelected = (selectedAnswer == answer),
      onAnswerSelected = { answer -> selectedAnswer = answer }
    )
  }
}
```

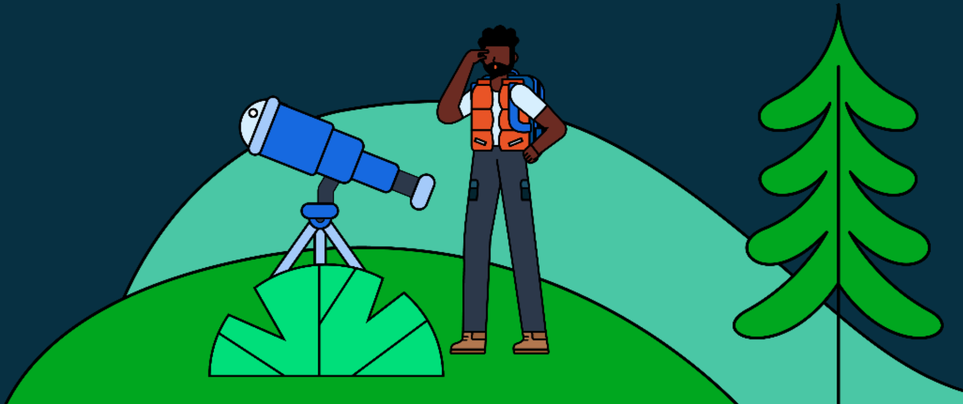


android

Las funciones *Composable* se pueden ejecutar en cualquier orden.

android

```kotlin
// ButtonRow.kt

@Composable
fun ButtonRow() {
  MyFancyNavigation {
    StartScreen()
    MiddleScreen()
    EndScreen()
  }
}
```

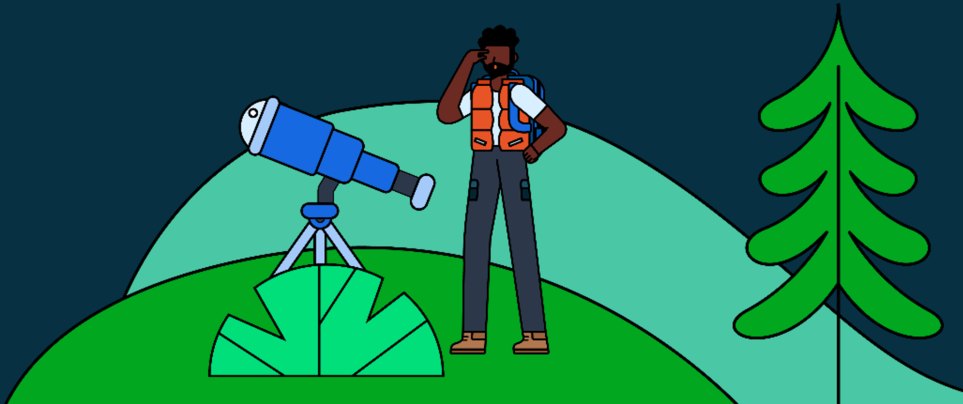


android

Las funciones *Composable* pueden correr en paralelo.

android

```kotlin
// ListComposable.kt

@Composable
fun ListComposable(myList: List<String>) {
  Row(horizontalArrangement = Arrangement.SpaceBetween) {
    Column {
      for (item in myList) {
        Text("Item: $item")
      }
    }
    Text("Count: ${myList.size}")
  }
}
```



android

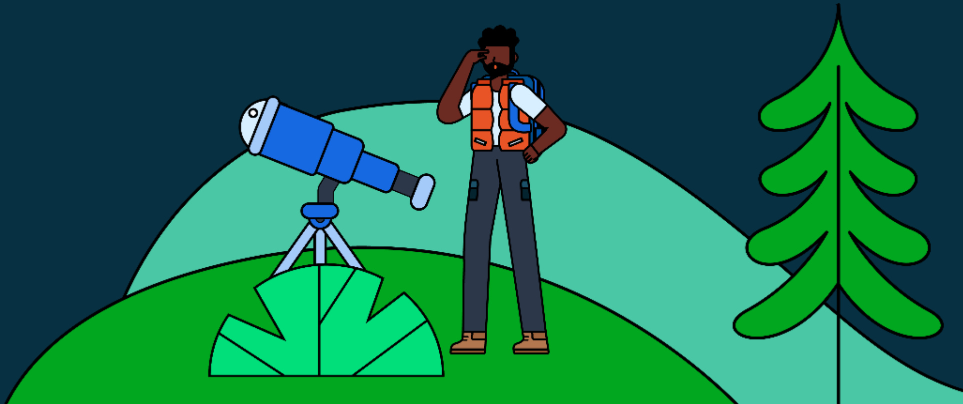
```
// ListComposable.kt

@Composable
fun ListWithBug(myList: List<String>) {
    var items = 0
    Row(horizontalArrangement = Arrangement.SpaceBetween) {
      Column {
        for (item in myList) {
          Text("Item: $item”)
          items++ // Evitar! Efecto secundario de la recomposición de la columna
        }
      }
      Text("Count: $items")
    }
}
```

android

La Recomposición evita la mayor cantidad de pasos posible.

android

```kotlin
// GreenScreen.kt

@Composable
fun GreetingScreen(name: String) {
  Column {
    Header()
    Greeting(name = name)
    Footer()
  }
}
```



android

La Recomposición es optimista.

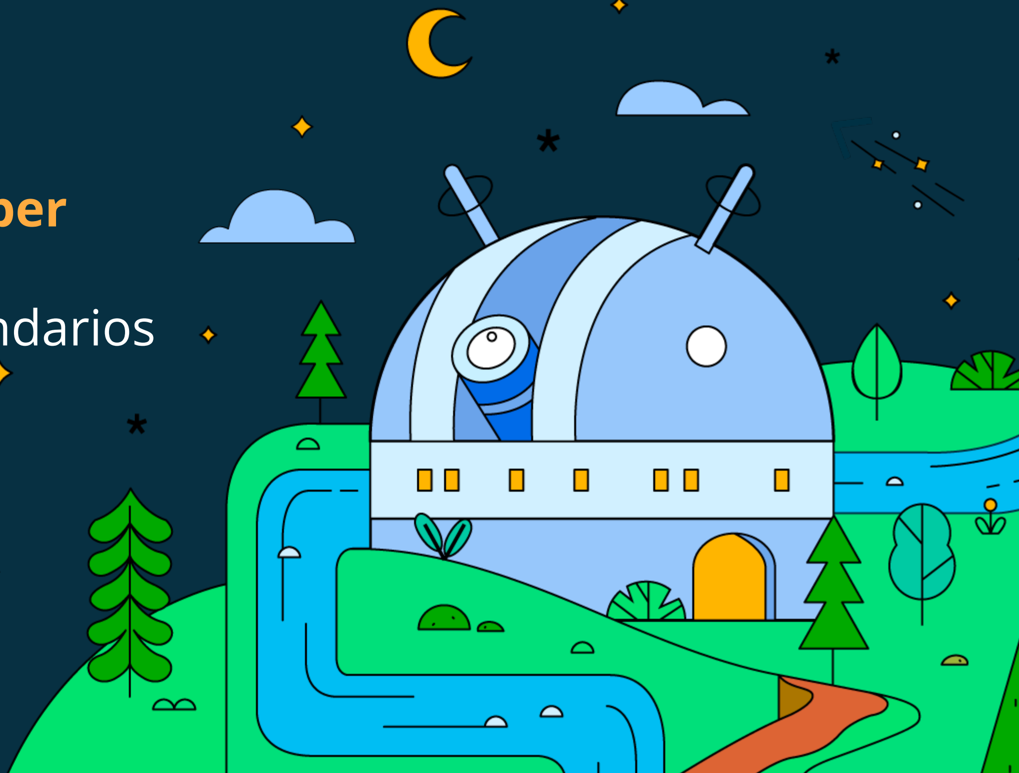
android

Las funciones Composable pueden ejecutarse de forma seguida.

android

¿Cómo usarlos?

1. Se usa la anotación @Composable
2. Aceptan **parámetros**
3. Usa **MutableState** y **remember**
4. No deben tener efectos secundarios



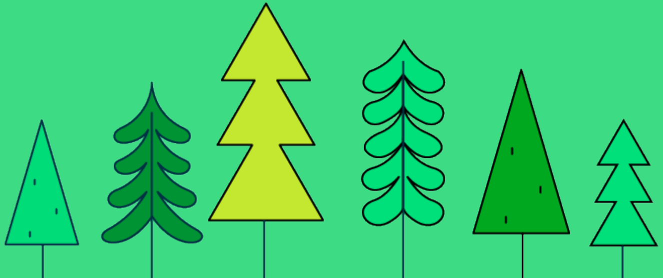
android

¿Cómo se ejecutan? Pueden...

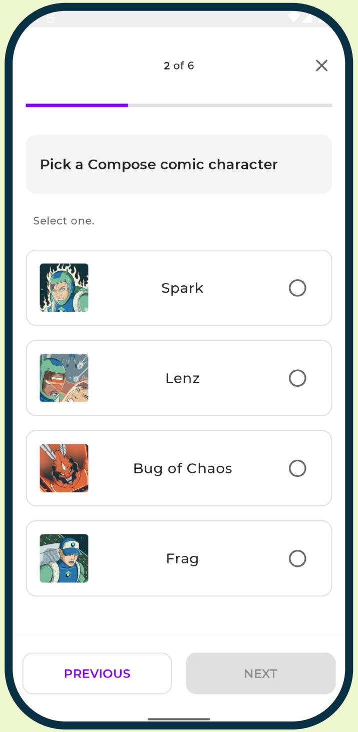
1. Ejecutarse en cualquier orden
2. Correr en paralelo
3. Ser “salteadas" / "ignoradas"
4. Correr frecuentemente

android

Compose Toolkit



android



2 of 6

Pick a Compose comic character

Select one.

Spark

Lenz

Bug of Chaos

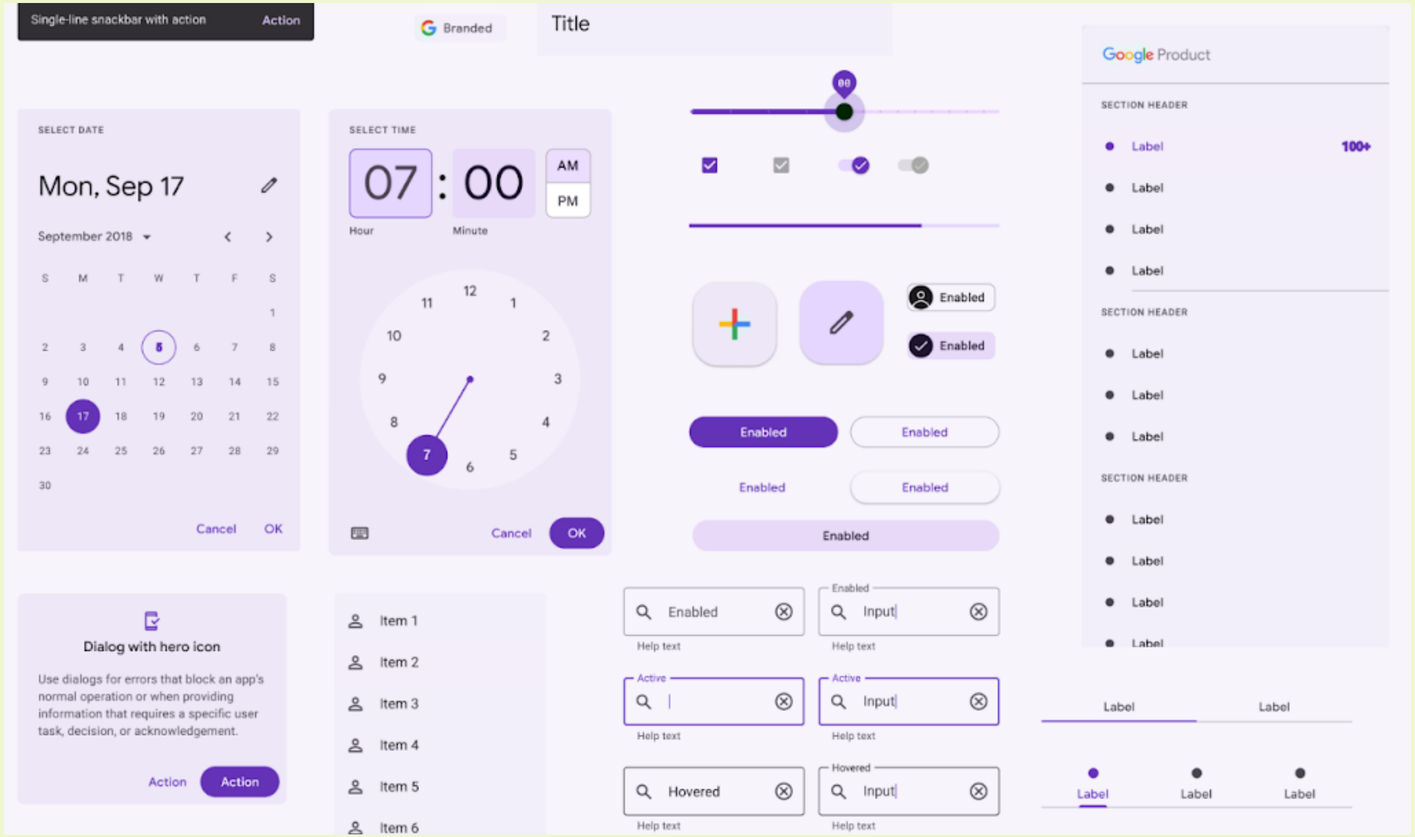
Frag

PREVIOUS NEXT

jetsurvey

goo.gle/compose-samples

android

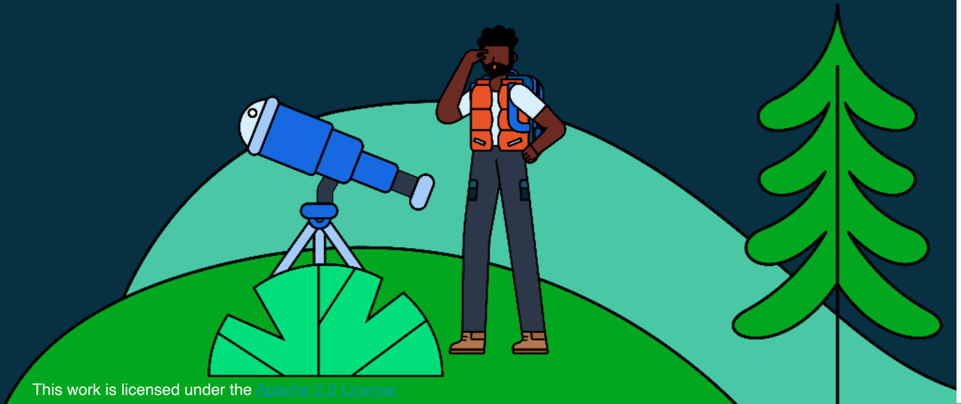

Single-line snackbar with action
Action
Branded
Title
Google Product
SECTION HEADER
Label
100+
Label
Label
Label
SECTION HEADER
Label
Label
Label
SECTION HEADER
Label
Label
Label
Label
SELECT DATE
Mon, Sep 17
September 2018
S M T W T F S
Cancel
OK
SELECT TIME
07 : 00
AM
PM
Hour
Minute
Cancel
OK
Enabled
Enabled
Enabled
Enabled
Enabled
Enabled
Enabled
Dialog with hero icon
Use dialogs for errors that block an app's normal operation or when providing information that requires a specific user task, decision, or acknowledgement.
Action
Action
Item 1
Item 2
Item 3
Item 4
Item 5
Item 6
Enabled
Input
Help text
Active
Hovered
Label
Label
Label
Label
Label

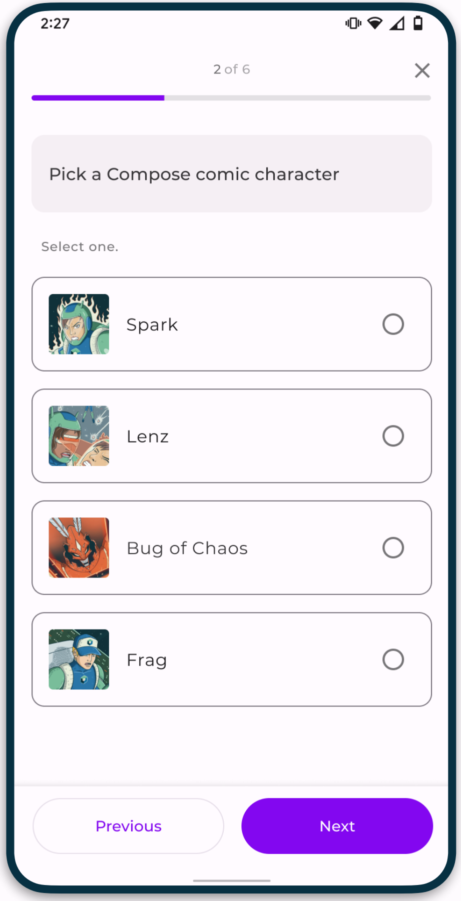
android

Material You

android

```
MaterialTheme(
    colorScheme = MyAppsColorScheme,
    typography = MyAppsTypography,
    shapes = MyAppsShapes
) {
    // Content goes here
}
```




2:27

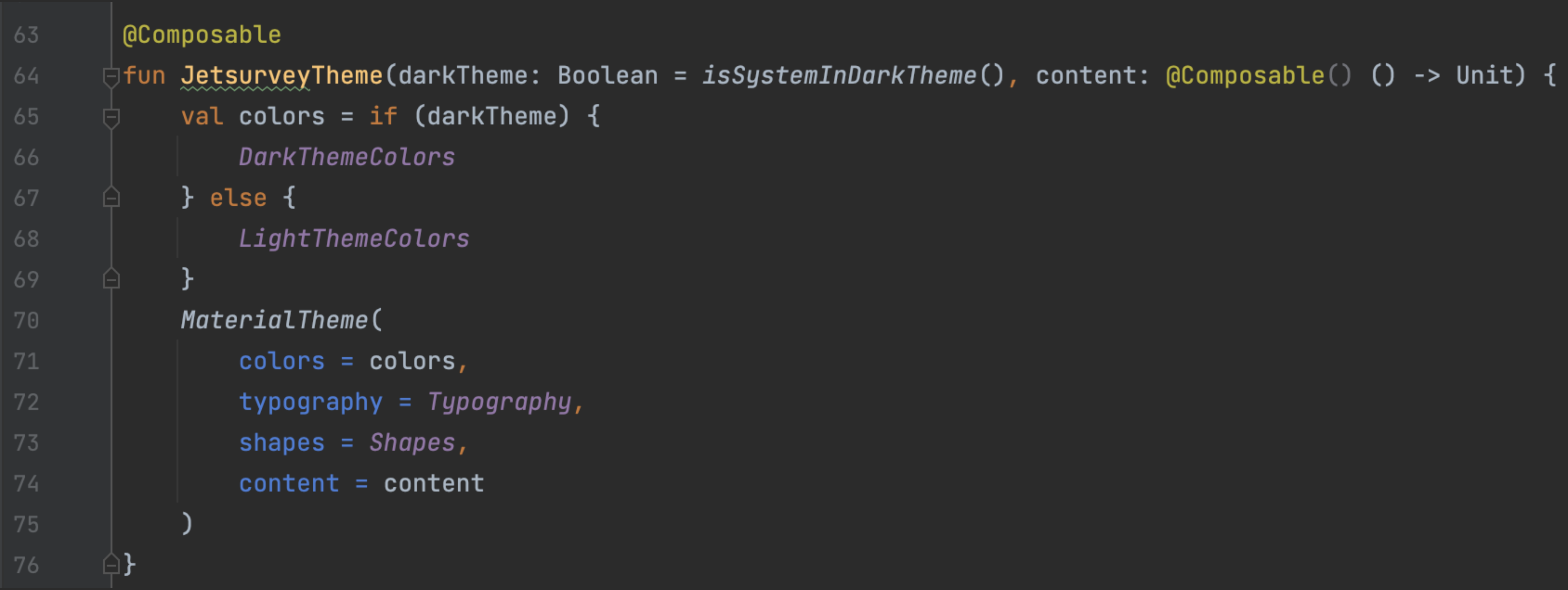
2 of 6

Pick a Compose comic character

Select one.

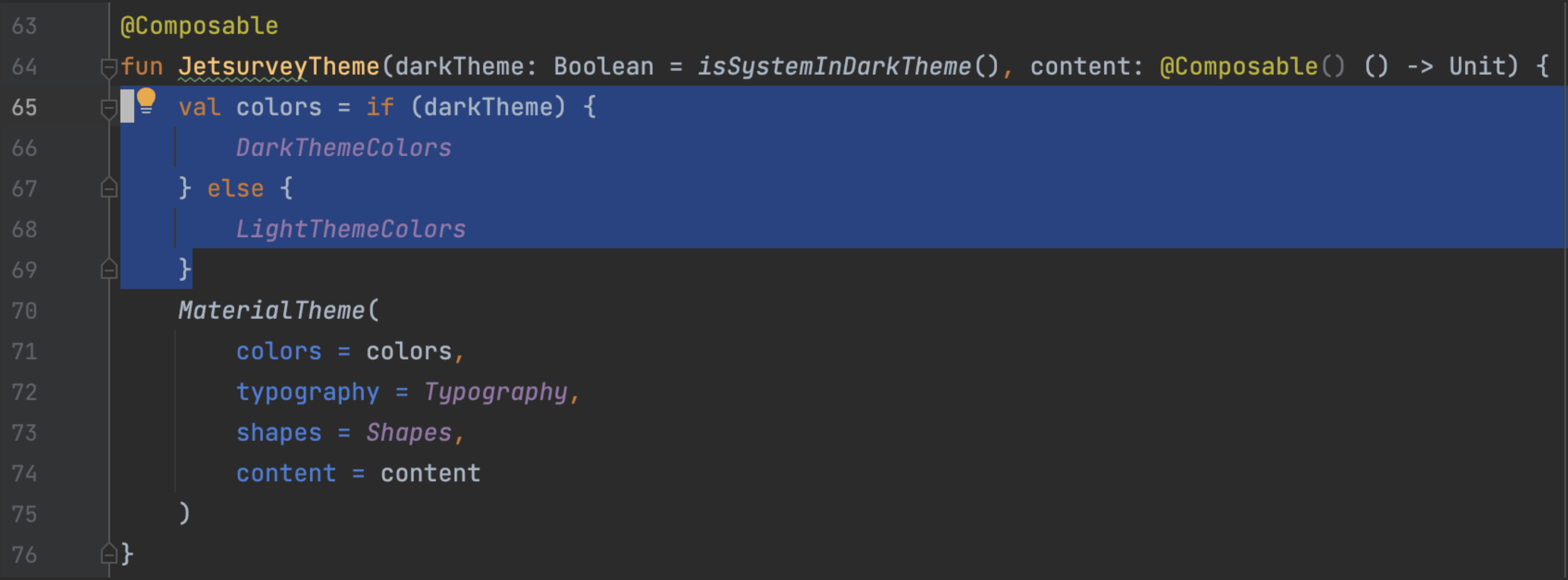
- Spark
- Lenz
- Bug of Chaos
- Frag

android



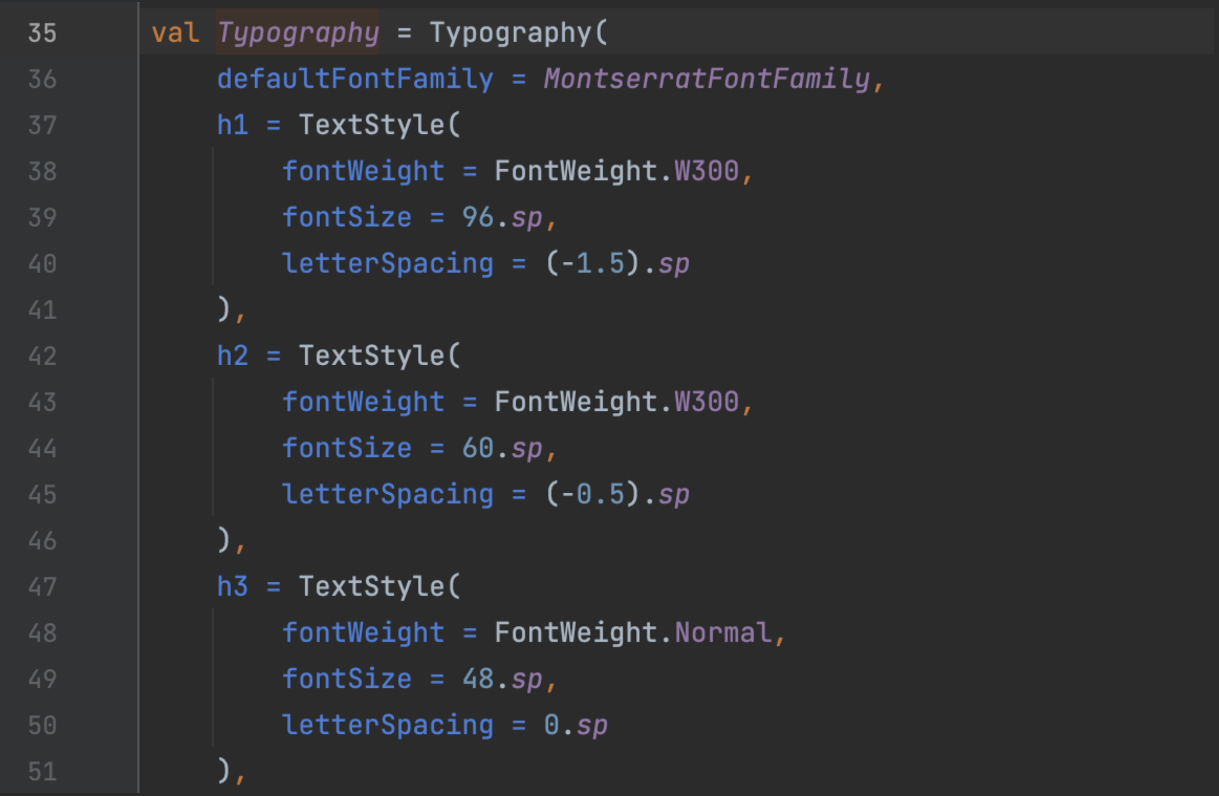
```kotlin
@Composable
fun JetsurveyTheme(darkTheme: Boolean = isSystemInDarkTheme(), content: @Composable() () -> Unit) {
    val colors = if (darkTheme) {
        DarkThemeColors
    } else {
        LightThemeColors
    }
    MaterialTheme(
        colors = colors,
        typography = Typography,
        shapes = Shapes,
        content = content
    )
}
```

android



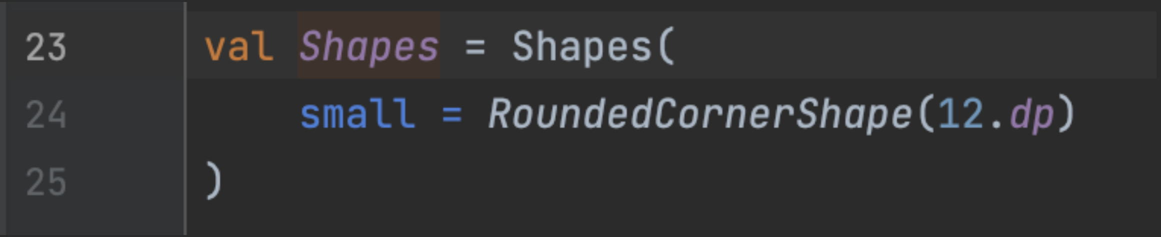
```
@Composable
fun JetsurveyTheme(darkTheme: Boolean = isSystemInDarkTheme(), content: @Composable() () -> Unit) {
    val colors = if (darkTheme) {
        DarkThemeColors
    } else {
        LightThemeColors
    }
    MaterialTheme(
        colors = colors,
        typography = Typography,
        shapes = Shapes,
        content = content
    )
}
```

android



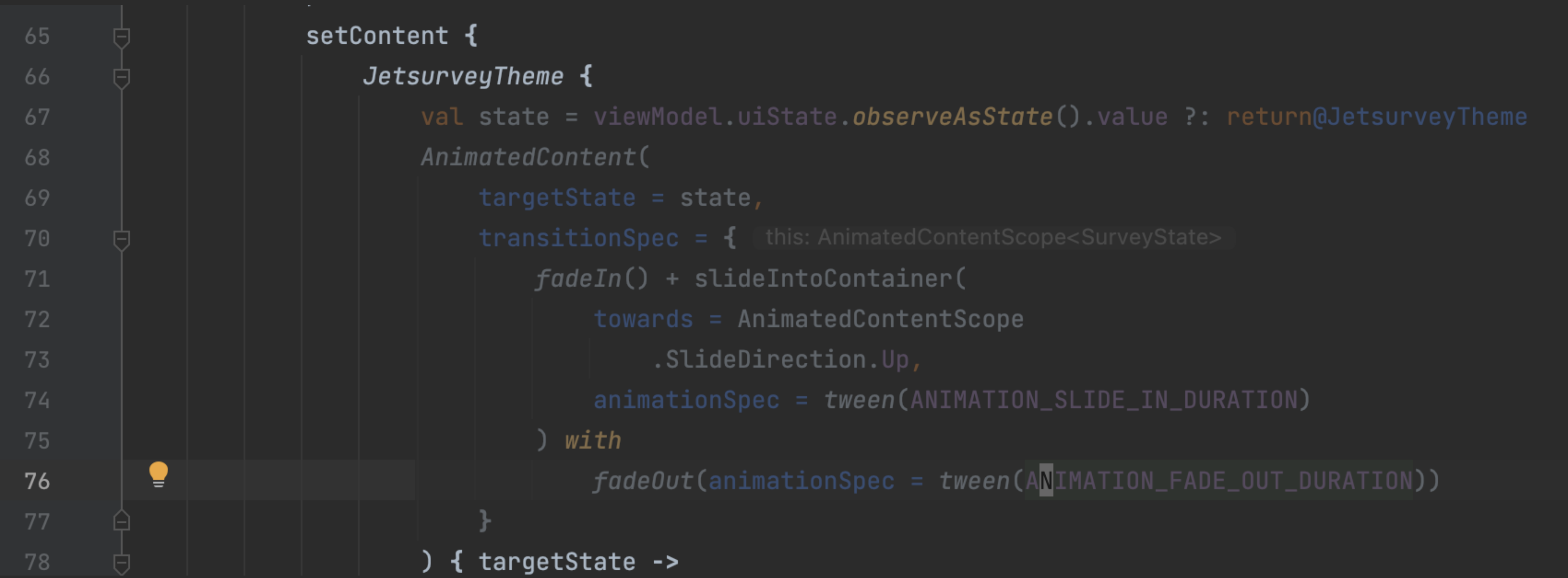
```kotlin
val Typography = Typography(
    defaultFontFamily = MontserratFontFamily,
    h1 = TextStyle(
        fontWeight = FontWeight.W300,
        fontSize = 96.sp,
        letterSpacing = (-1.5).sp
    ),
    h2 = TextStyle(
        fontWeight = FontWeight.W300,
        fontSize = 60.sp,
        letterSpacing = (-0.5).sp
    ),
    h3 = TextStyle(
        fontWeight = FontWeight.Normal,
        fontSize = 48.sp,
        letterSpacing = 0.sp
    ),
```

android



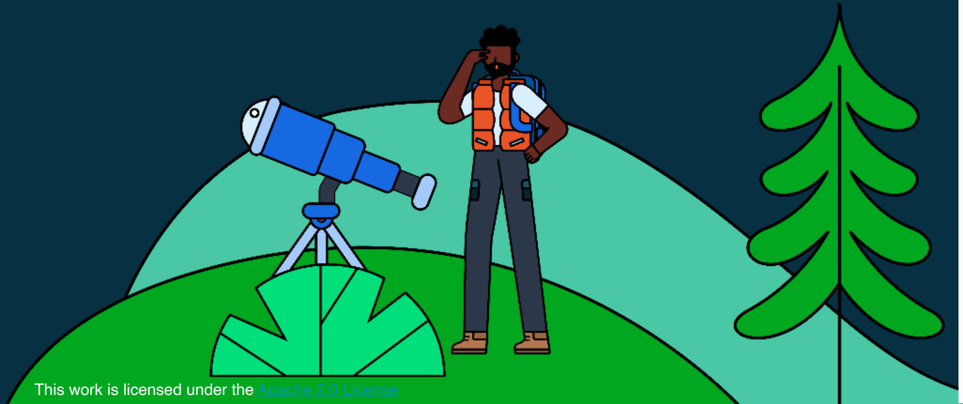
```
val Shapes = Shapes(
        small = RoundedCornerShape(12.dp)
)
```

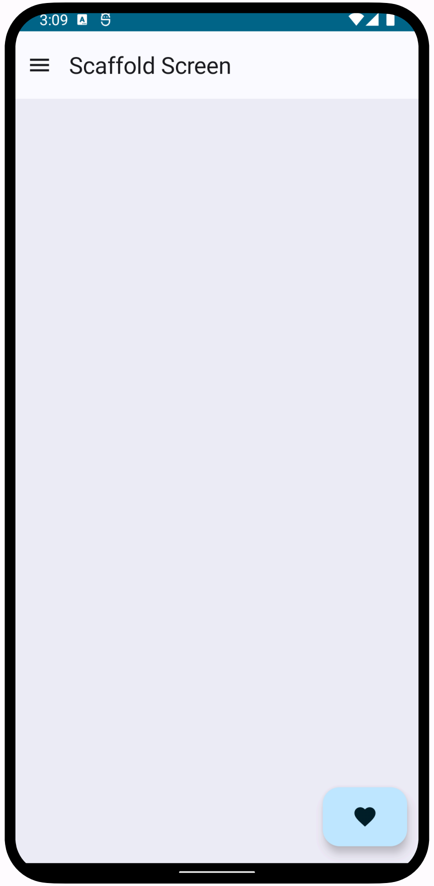
android



```kotlin
setContent {
    JetsurveyTheme {
        val state = viewModel.uiState.observeAsState().value ?: return@JetsurveyTheme
        AnimatedContent(
            targetState = state,
            transitionSpec = {  this: AnimatedContentScope<SurveyState>
                fadeIn() + slideIntoContainer(
                    towards = AnimatedContentScope
                        .SlideDirection.Up,
                    animationSpec = tween(ANIMATION_SLIDE_IN_DURATION)
                ) with
                    fadeOut(animationSpec = tween(ANIMATION_FADE_OUT_DURATION))
            }
        ) { targetState ->
```

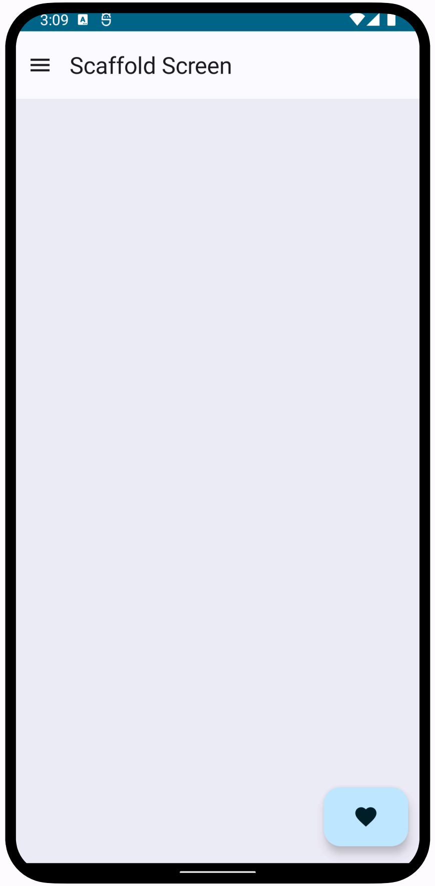
```
Scaffold(
  topBar = { SmallTopAppBar(/* ... */) },
  floatingActionButtonPosition = FabPosition.End,
  floatingActionButton = {
     FloatingActionButton(/* ... */)
  },
  content = { /* ... */ }
)
```

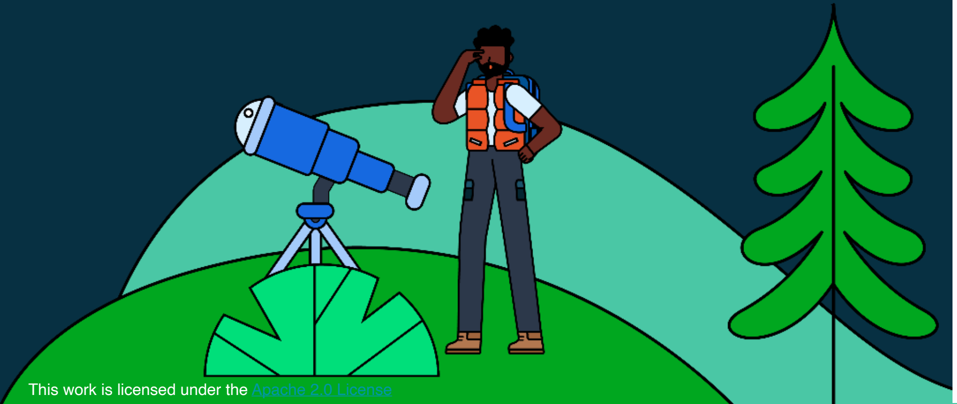




android

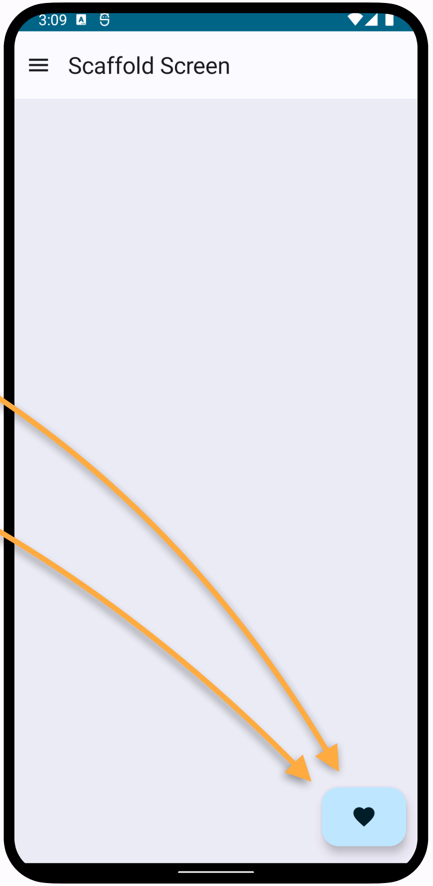
```
Scaffold(
  topBar = { SmallTopAppBar(/* ... */) },
  floatingActionButtonPosition = FabPosition.End,
  floatingActionButton = {
     FloatingActionButton(/* ... */)
  },
  content = { /* ... */ }
)
```

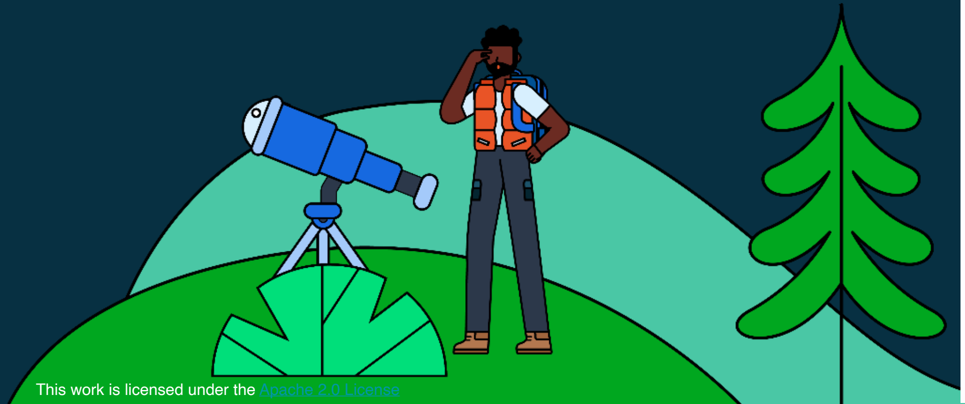




android

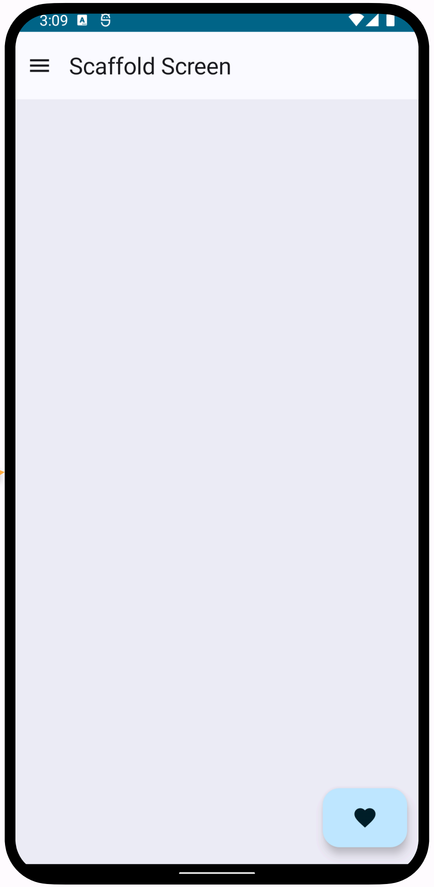
```
Scaffold(
  topBar = { SmallTopAppBar(/* ... */) },
  floatingActionButtonPosition = FabPosition.End,
  floatingActionButton = {
      FloatingActionButton(/* ... */)
  },
  content = { /* ... */ }
)
```

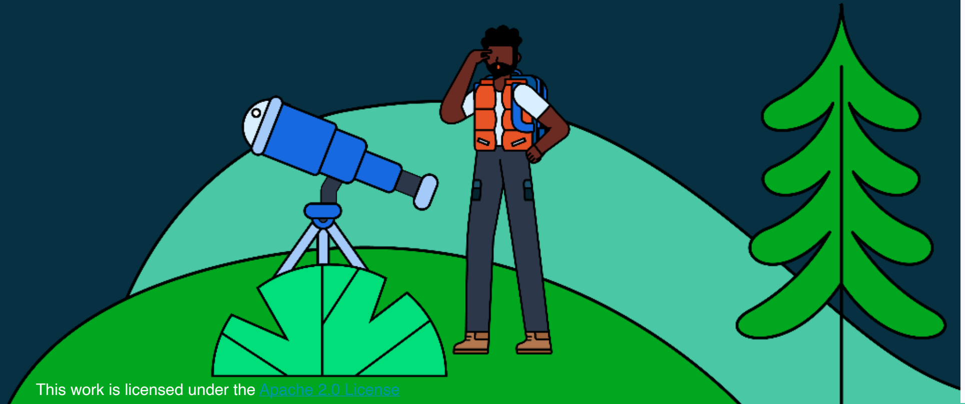




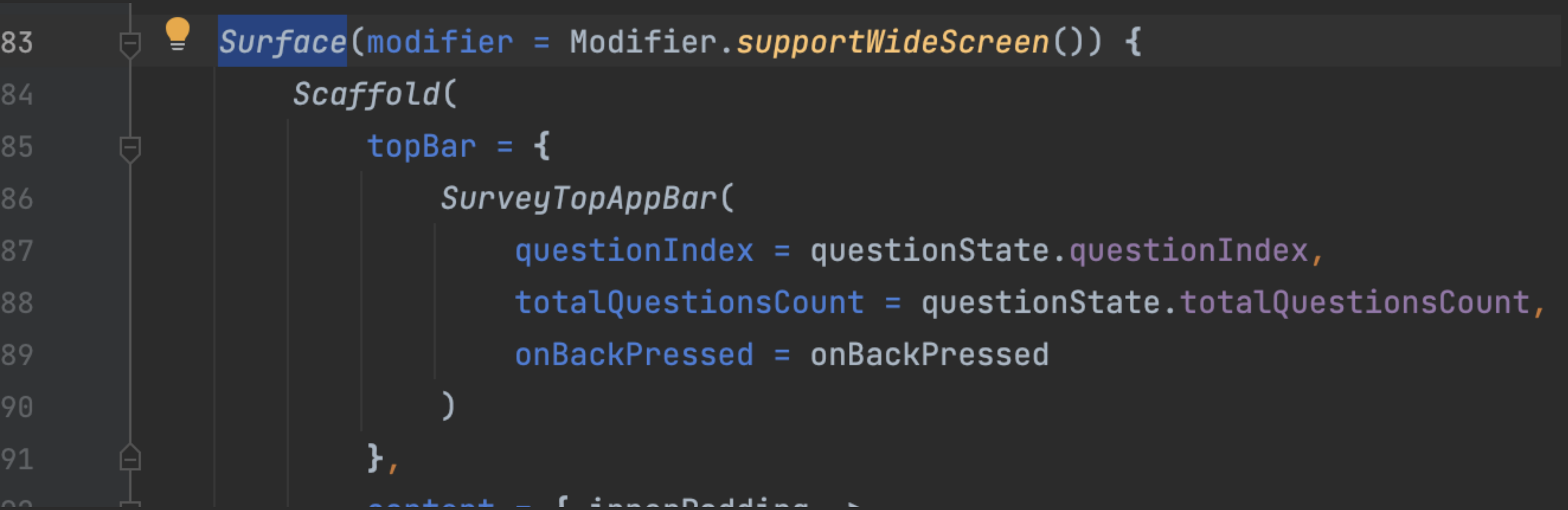
android

```
Scaffold(
  topBar = { SmallTopAppBar(/* ... */) },
  floatingActionButtonPosition = FabPosition.End,
  floatingActionButton = {
    FloatingActionButton(/* ... */)
  },
  content = { /* ... */ }
)
```





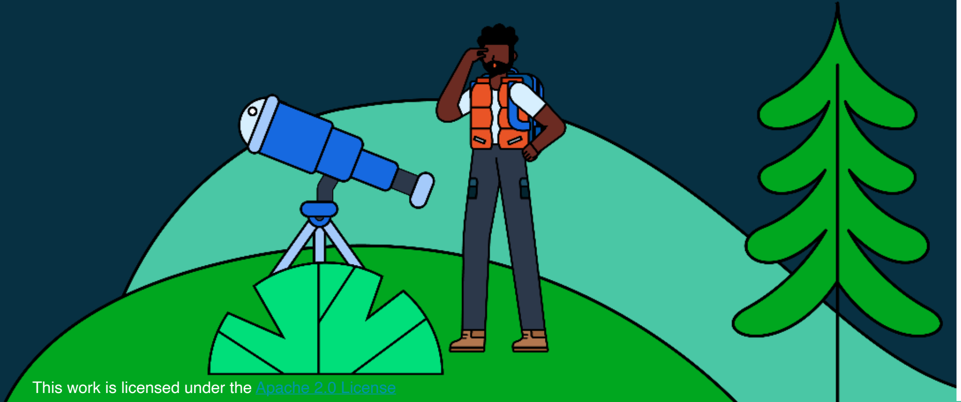
android



```kotlin
Surface(modifier = Modifier.supportWideScreen()) {
    Scaffold(
        topBar = {
            SurveyTopAppBar(
                questionIndex = questionState.questionIndex,
                totalQuestionsCount = questionState.totalQuestionsCount,
                onBackPressed = onBackPressed
            )
        },
        content = { innerPadding ->
```

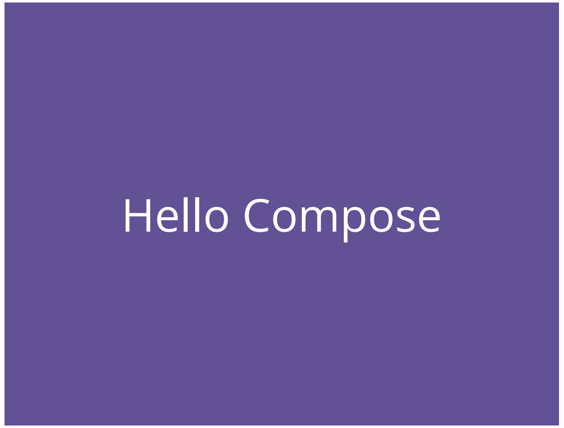
```
Surface {
    Text("Hello Compose")
}
```

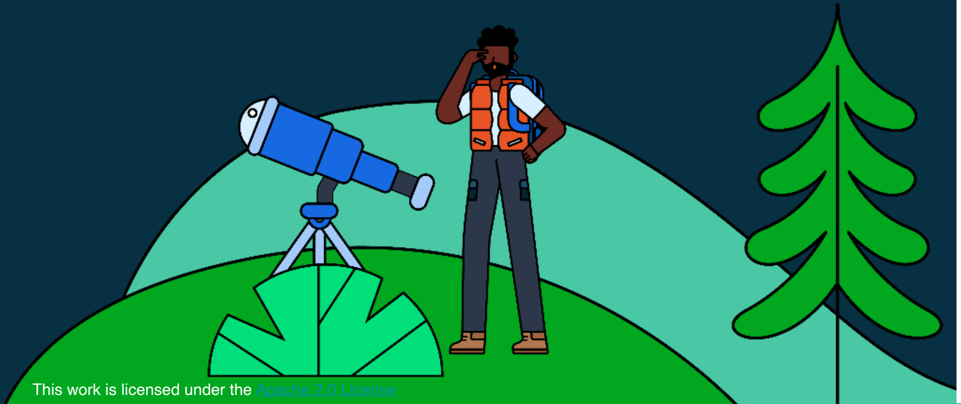
Hello Compose



android

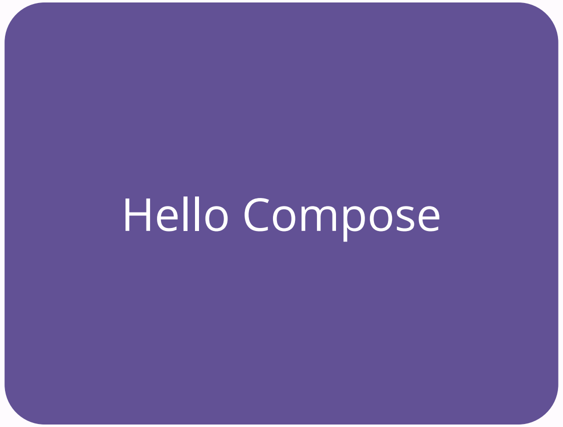
```
Surface(
  color = MaterialTheme.colorScheme.primary,
) {
   Text("Hello Compose")
}
```

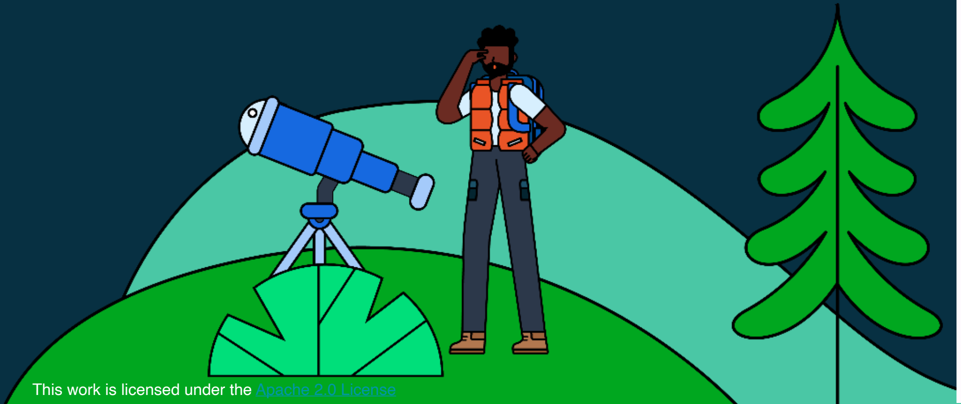




android

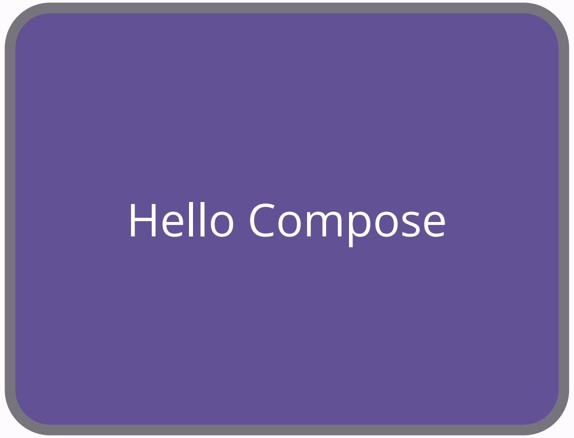
```
Surface(
  color = MaterialTheme.colorScheme.primary,
  shape = RoundedCornerShape(8.dp),
) {
   Text("Hello Compose")
}
```

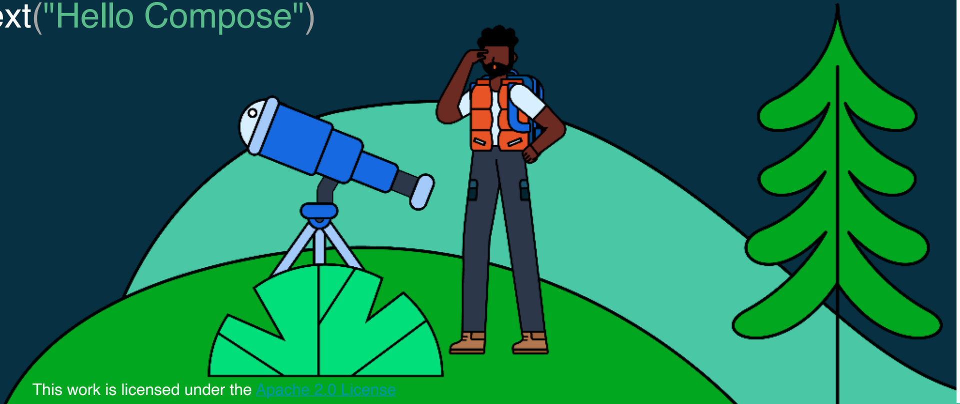

Hello Compose



android

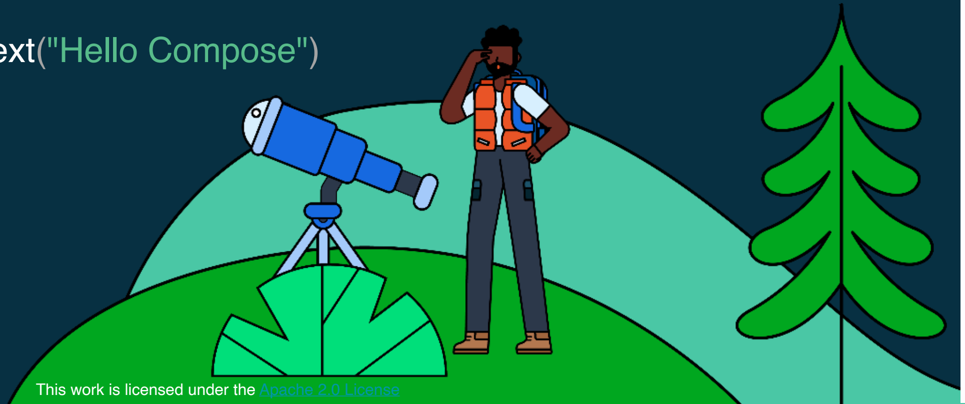
```
Surface(
  color = MaterialTheme.colorScheme.surface,
  shape = RoundedCornerShape(8.dp),
  border = BorderStroke(2.dp,
    MaterialTheme.colorScheme.outline
  )
) {
   Text("Hello Compose")
}
```

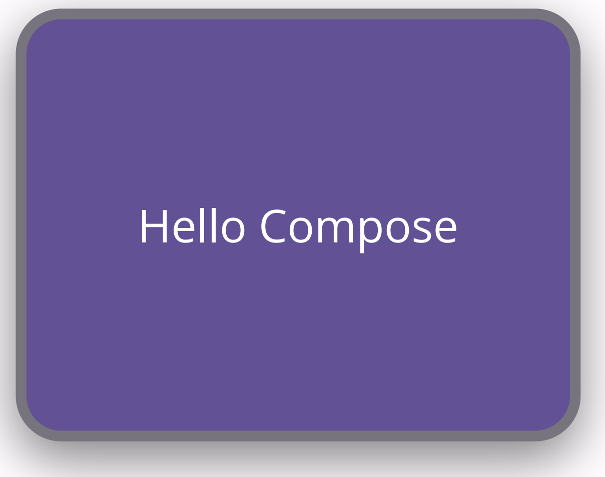




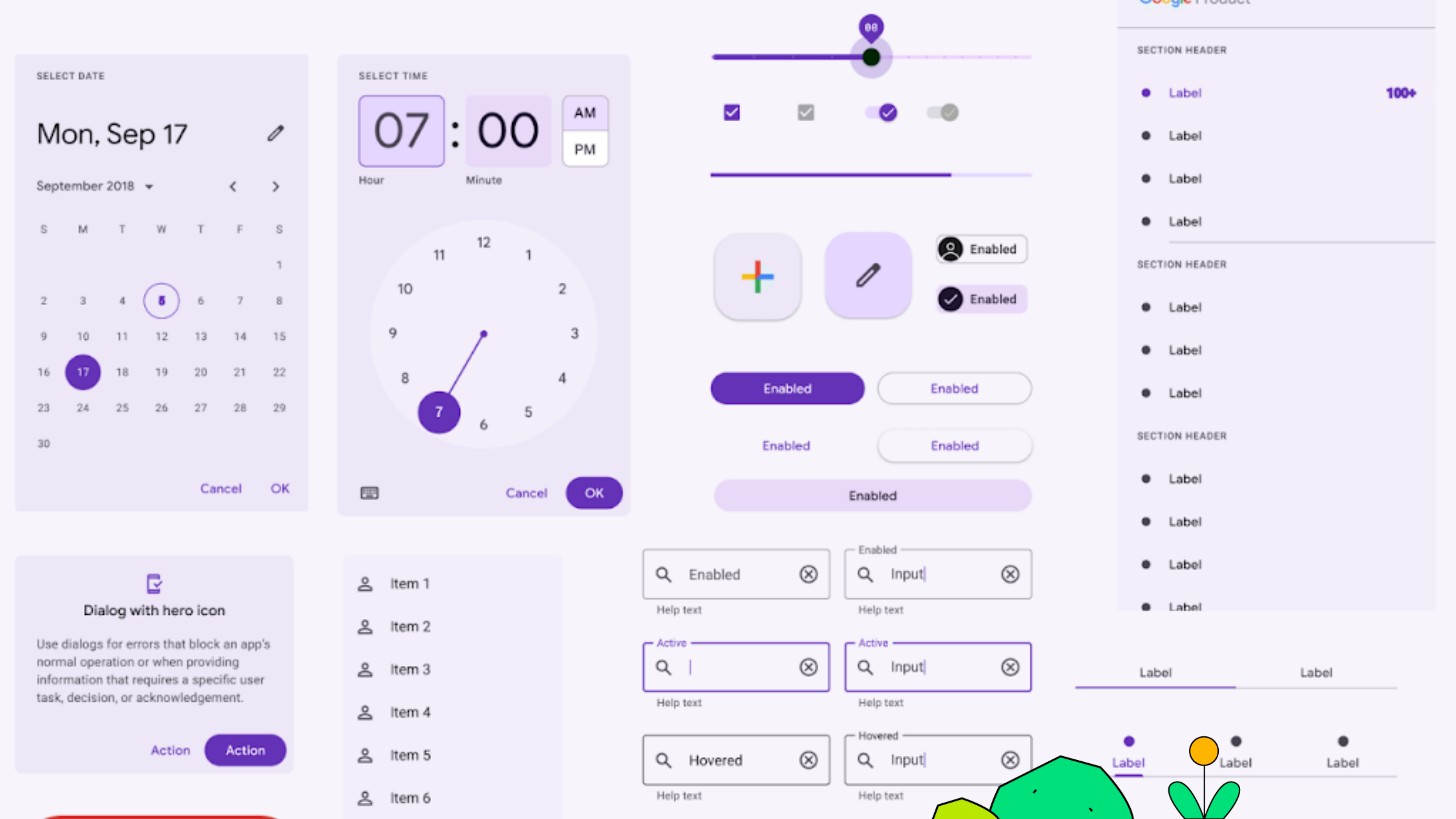
android

```
Surface(
  color = MaterialTheme.colorScheme.surface,
  shape = RoundedCornerShape(8.dp),
  border = BorderStroke(2.dp,
    MaterialTheme.colorScheme.surfaceVariant
  ),
  shadowElevation = 8.dp,
  tonalElevation = 8.dp,
) {
    Text("Hello Compose")
}
```

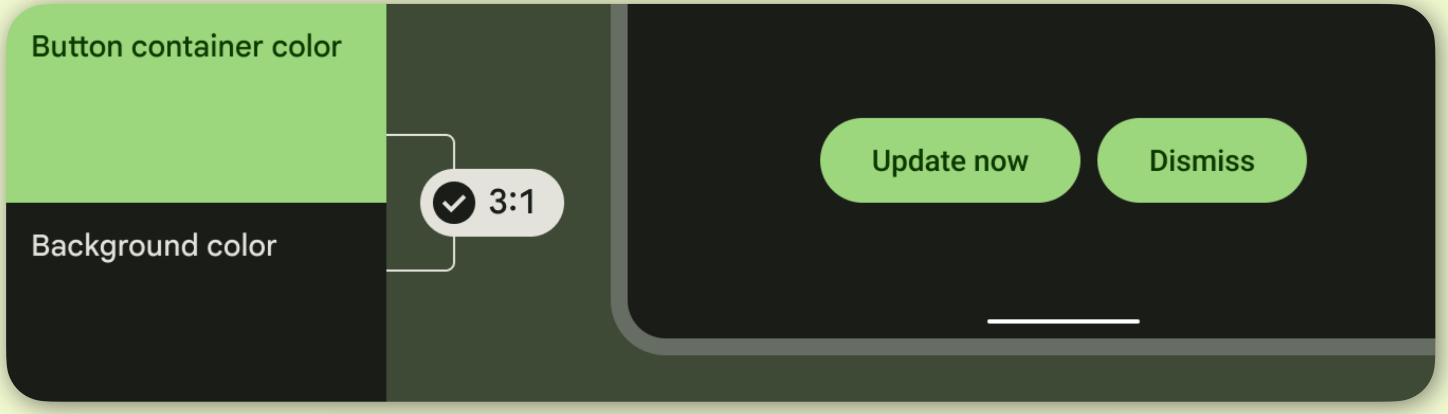


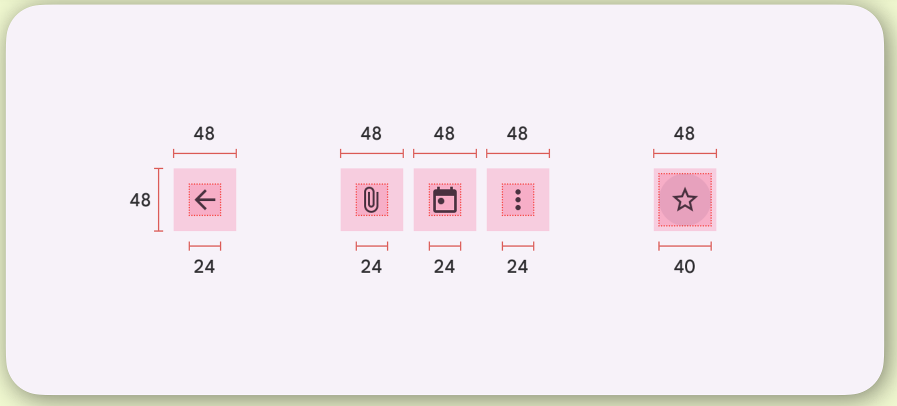


android

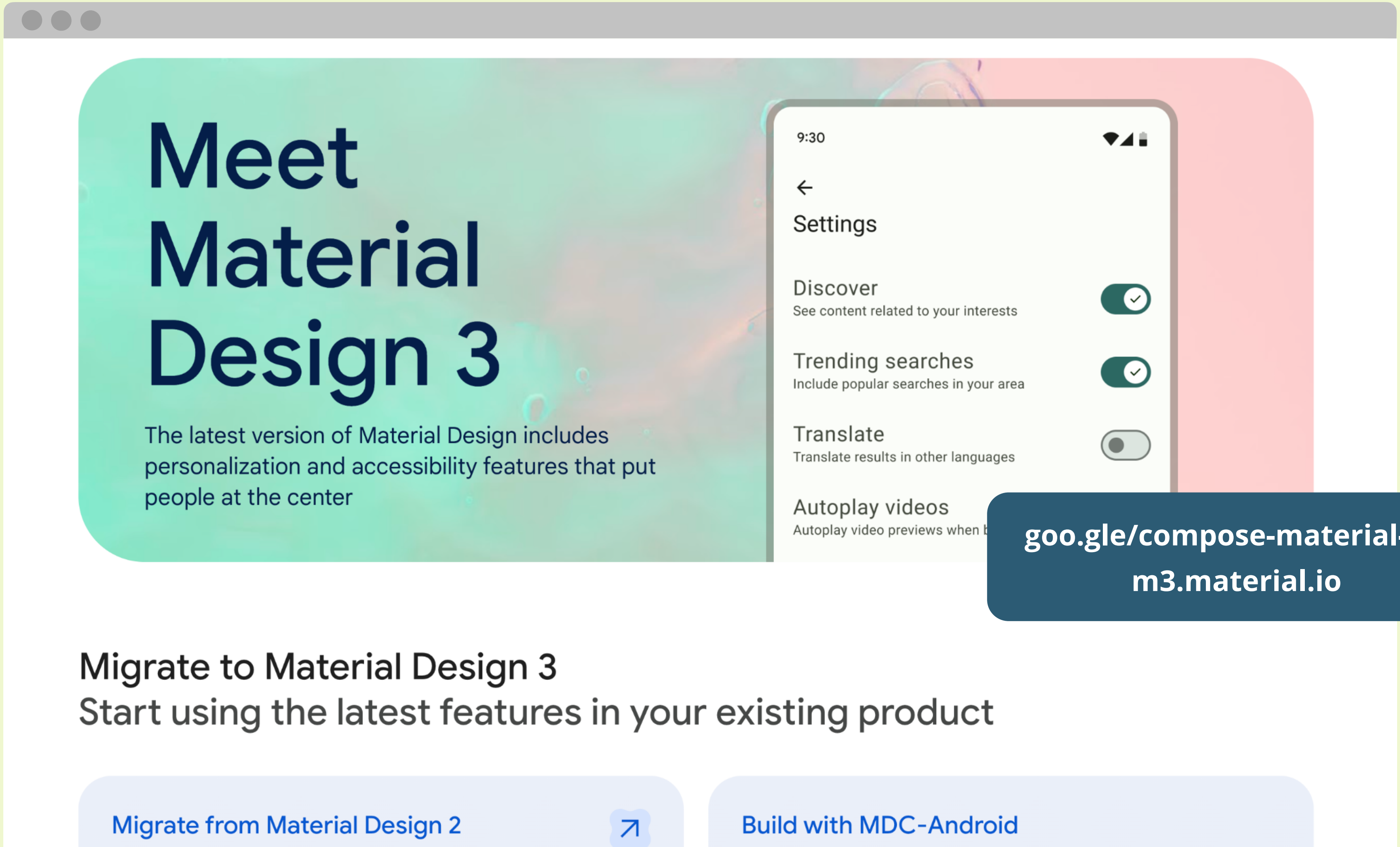


Use dialogs for errors that block an app's normal operation or when providing information that requires a specific user task, decision, or acknowledgement.


Button container color
Background color
3:1
Update now
Dismiss


48
48
24
48
48
48
24
24
24
48
40

android



Meet Material Design 3

The latest version of Material Design includes personalization and accessibility features that put people at the center

9:30

Settings

Discover
See content related to your interests ✓

Trending searches
Include popular searches in your area ✓

Translate
Translate results in other languages

Autoplay videos
Autoplay video previews when

goo.gle/compose-material-ref

m3.material.io

Migrate to Material Design 3

Start using the latest features in your existing product

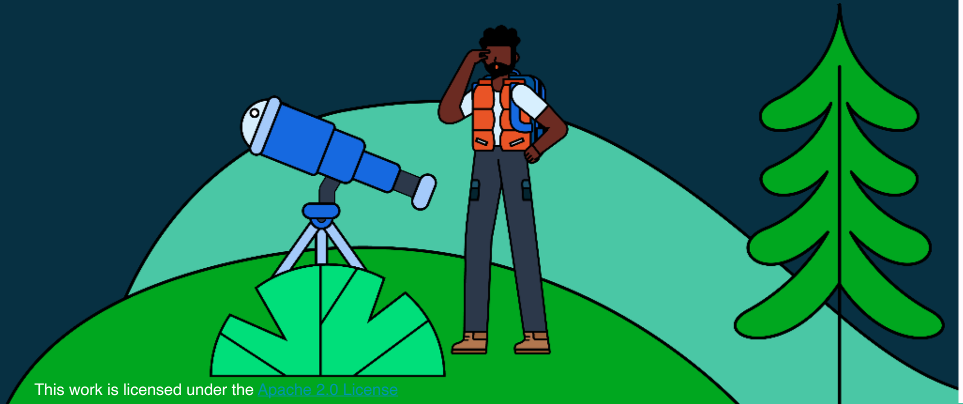
Migrate from Material Design 2

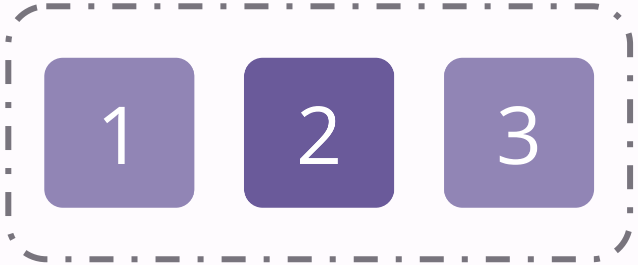
Build with MDC-Android

Standard layouts

android

```
Row {
  Component1()
  Component2()
  Component3()
}
```

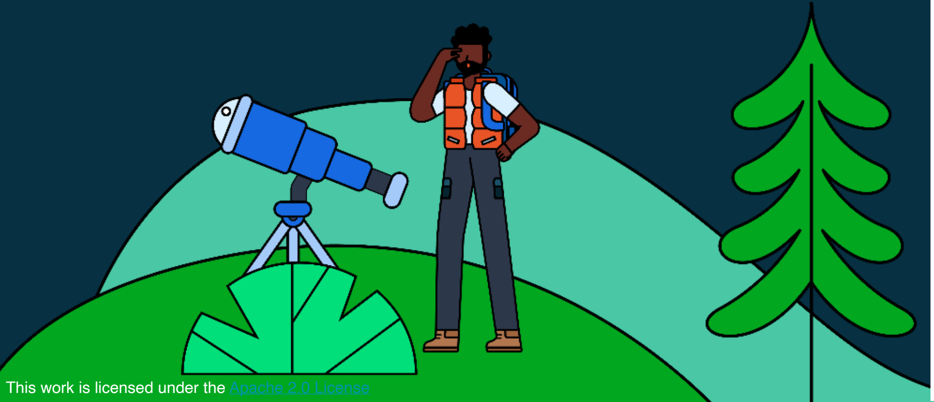


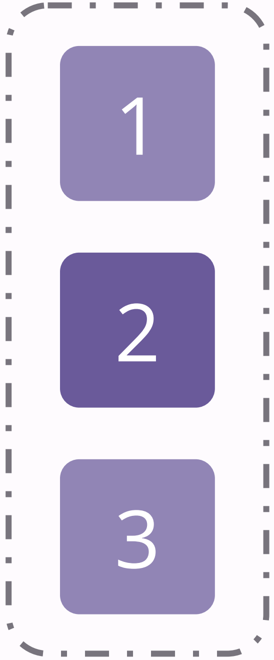


Row

android

```
Column {
  Component1()
  Component2()
  Component3()
}
```

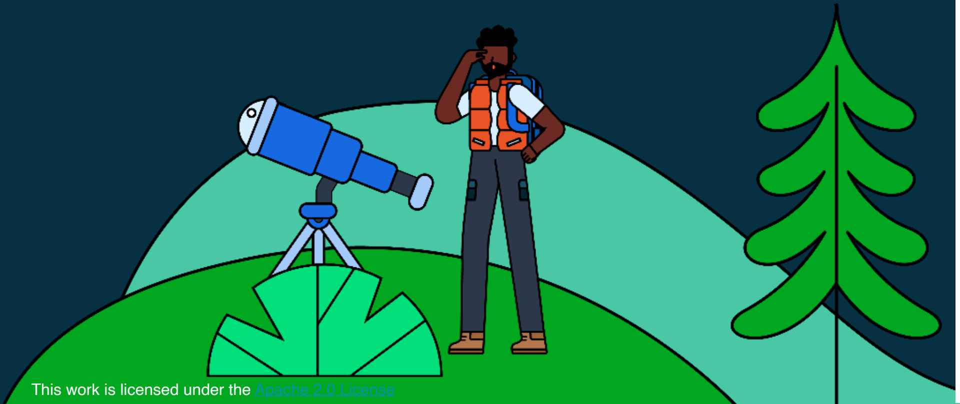


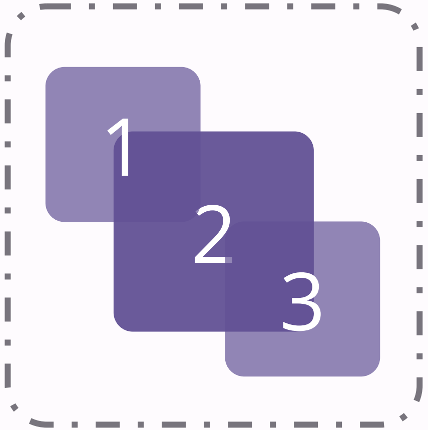


Column

android

```
Box {
  Component1()
  Component2()
  Component3()
}
```

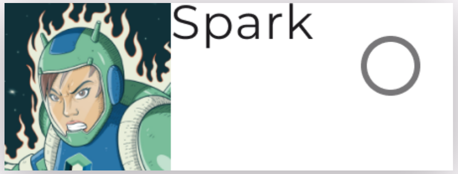


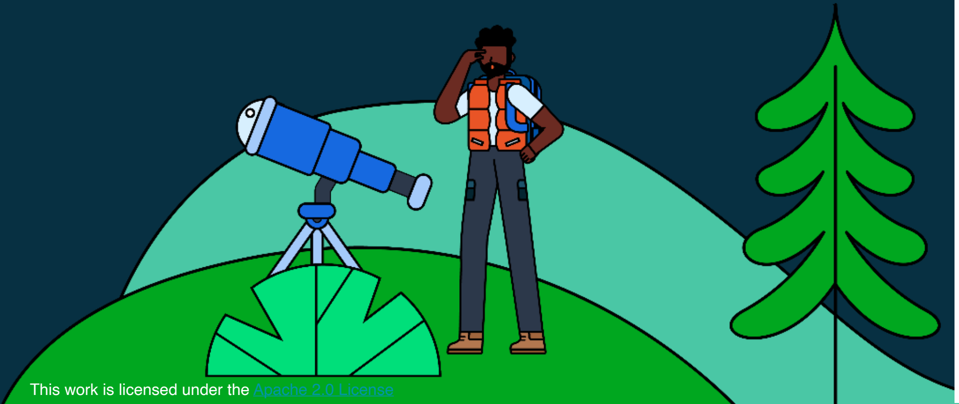


Box

android

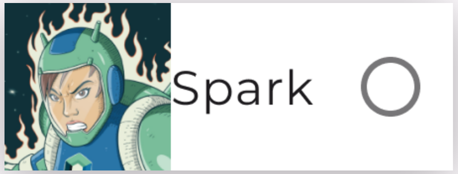
```
@Composable
fun SurveyAnswer(answer: Answer) {
  Row {
    Image(answer.image)
    Text(answer.text)
    RadioButton(/* … */)
  }
}
```

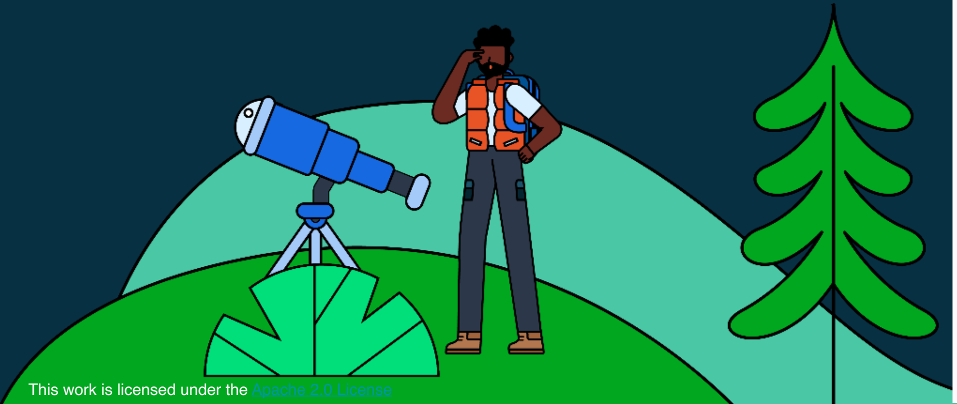




android

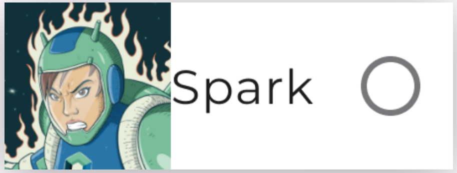
```
@Composable
fun SurveyAnswer(answer: Answer) {
  Row(
    verticalAlignment =
      Alignment.CenterVertically
  ) {
    /* ... */
  }
}
```

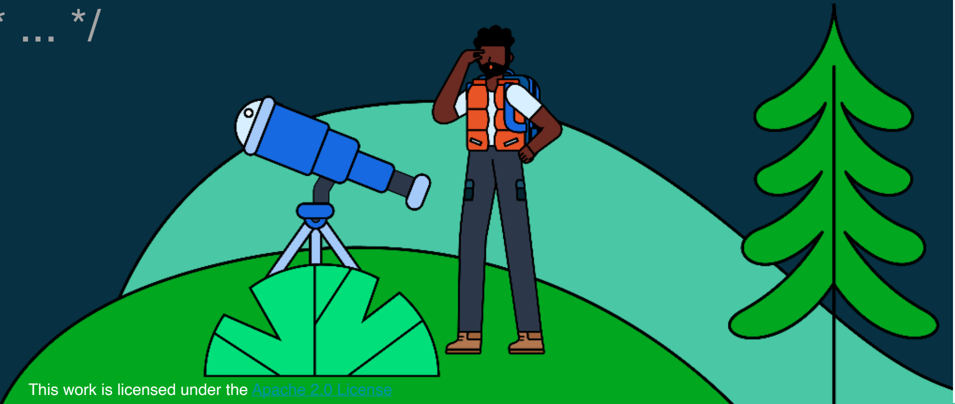




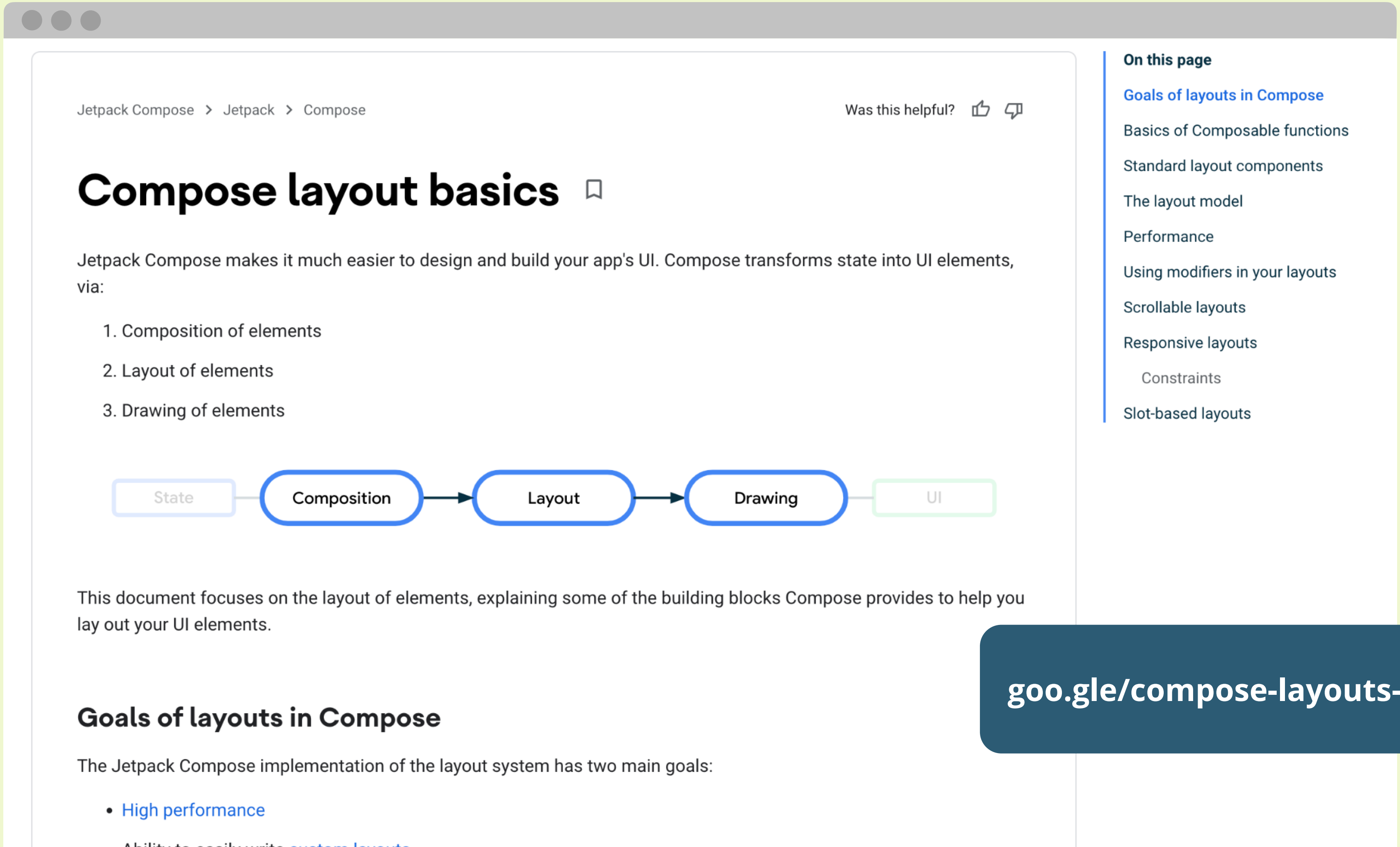
android

```
@Composable
fun SurveyAnswer(answer: Answer) {
  Row(
    verticalAlignment =
      Alignment.CenterVertically,
    horizontalArrangement =
      Arrangement.SpaceBetween
  ) {
    /* ... */
  }
}
```





android



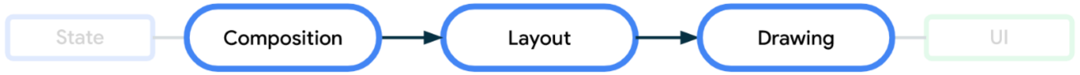
Jetpack Compose > Jetpack > Compose

Was this helpful?

Compose layout basics

Jetpack Compose makes it much easier to design and build your app's UI. Compose transforms state into UI elements, via:

1. Composition of elements
2. Layout of elements
3. Drawing of elements



This document focuses on the layout of elements, explaining some of the building blocks Compose provides to help you lay out your UI elements.

Goals of layouts in Compose

The Jetpack Compose implementation of the layout system has two main goals:

- High performance
- Ability to easily write custom layouts

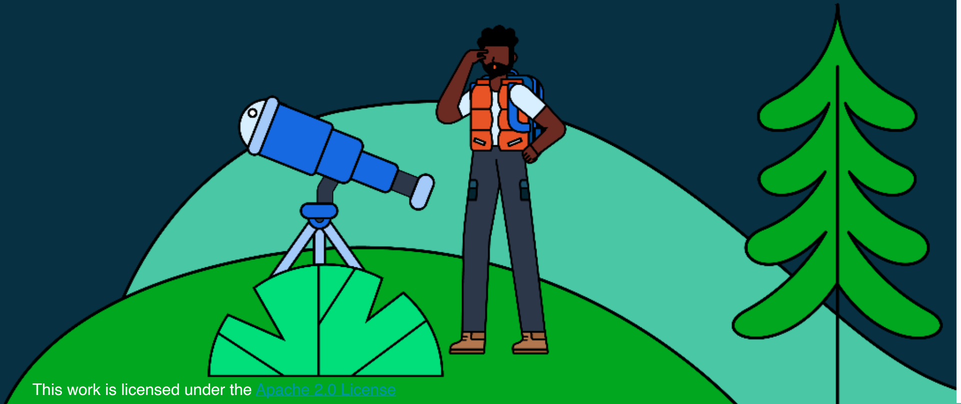
goo.gle/compose-layouts-docs

Modifiers

android

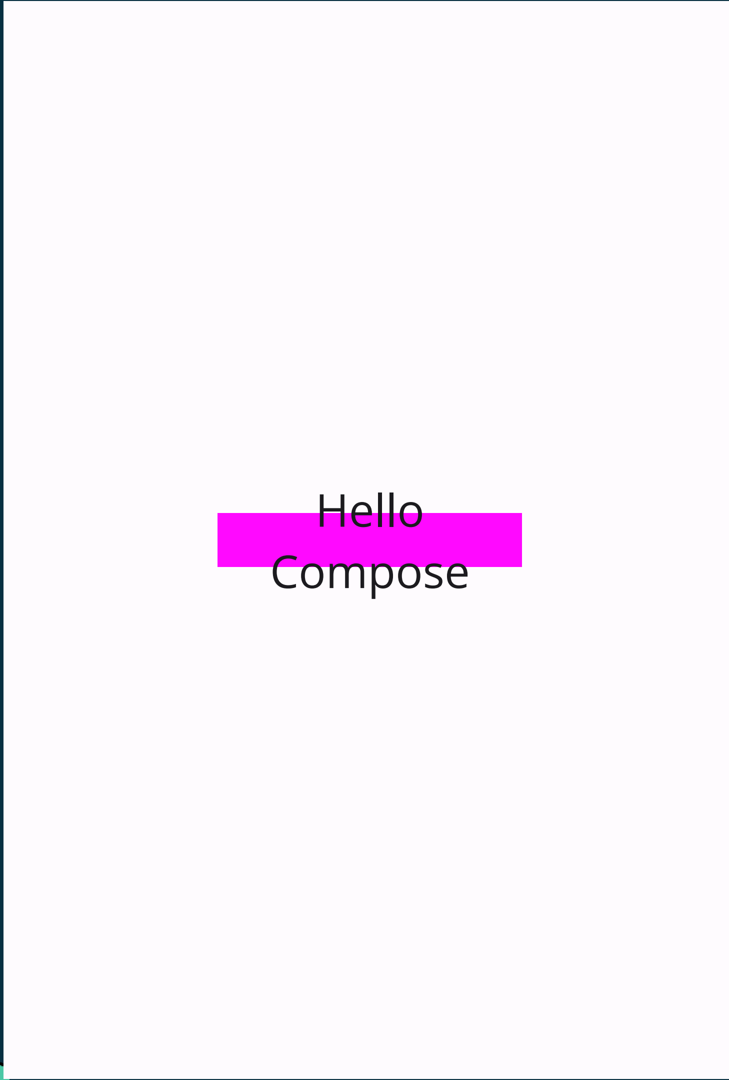
```
Text("Hello Compose")
```

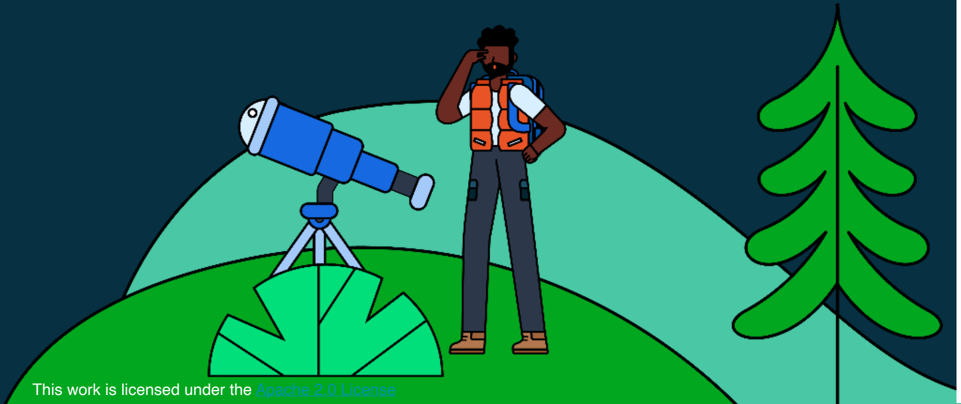
Hello
Compose



android

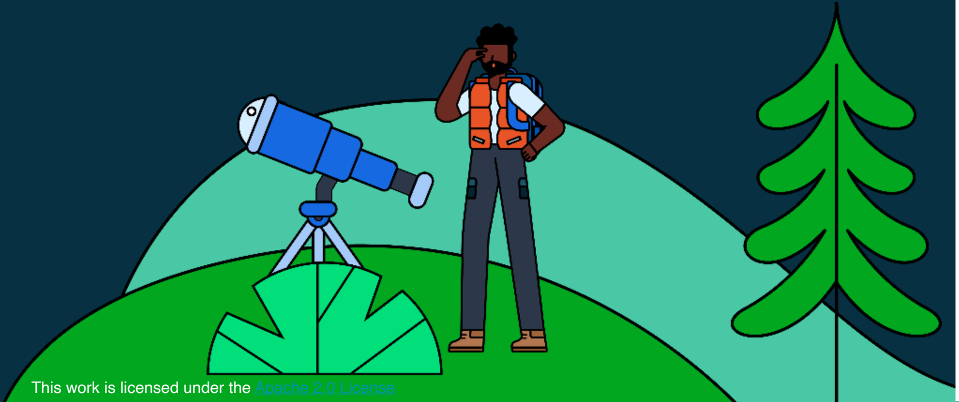
```
Text(
  "Hello Compose!",
  Modifier.background(Color.Magenta)
)
```

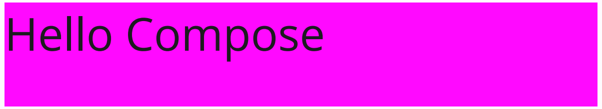




android

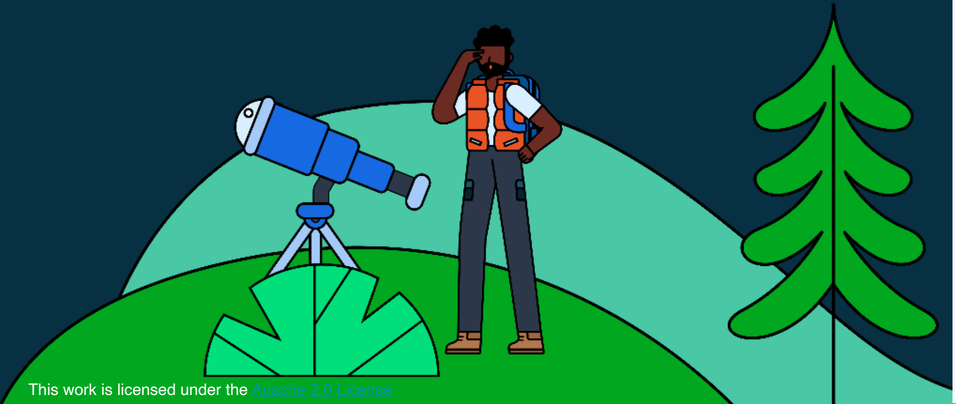
```
Text(
  "Hello Compose!",
  Modifier.background(Color.Magenta)
    .size(200.dp, 30.dp)
)
```

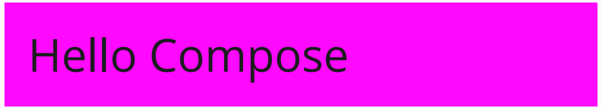




android

```
Text(
  "Hello Compose!",
  Modifier.background(Color.Magenta)
    .size(200.dp, 30.dp)
    .padding(5.dp)
)
```

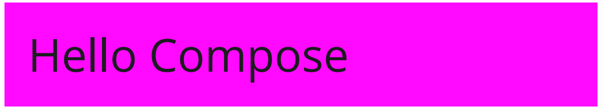


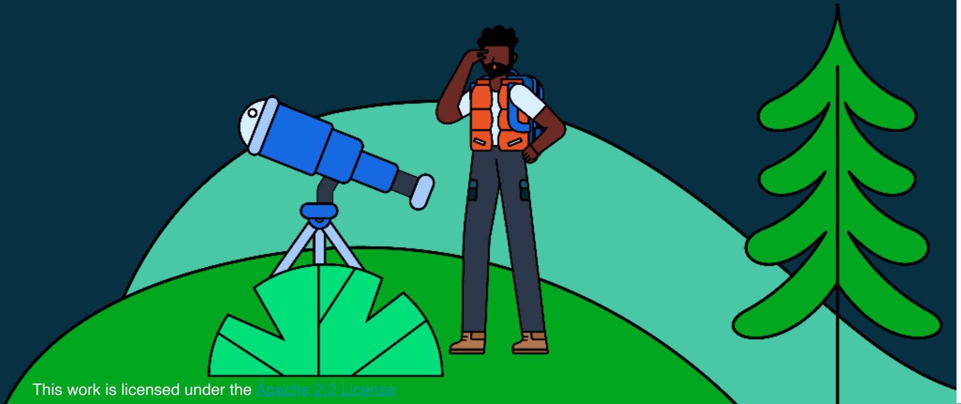


Hello Compose

android

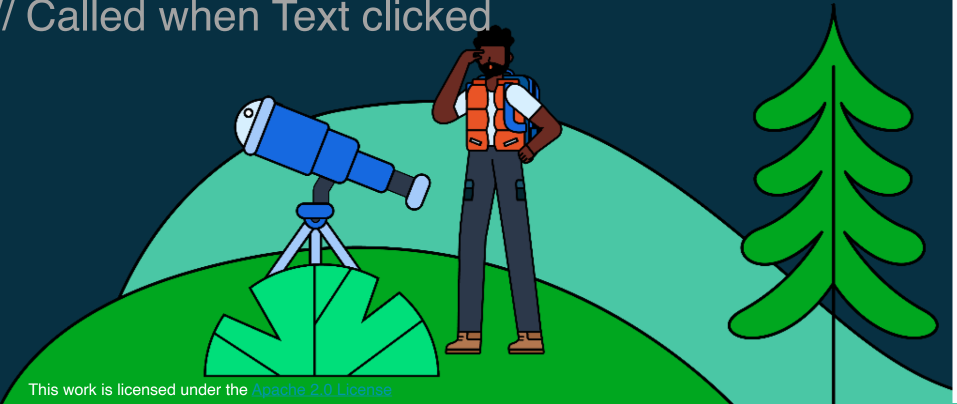
```
Text(
  "Hello Compose!",
  Modifier.background(Color.Magenta)
    .size(200.dp, 30.dp)
    .padding(5.dp)
    .alpha(0.5f)
)
```

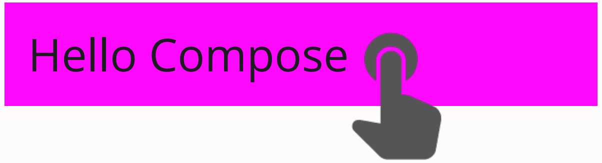

Hello Compose



android

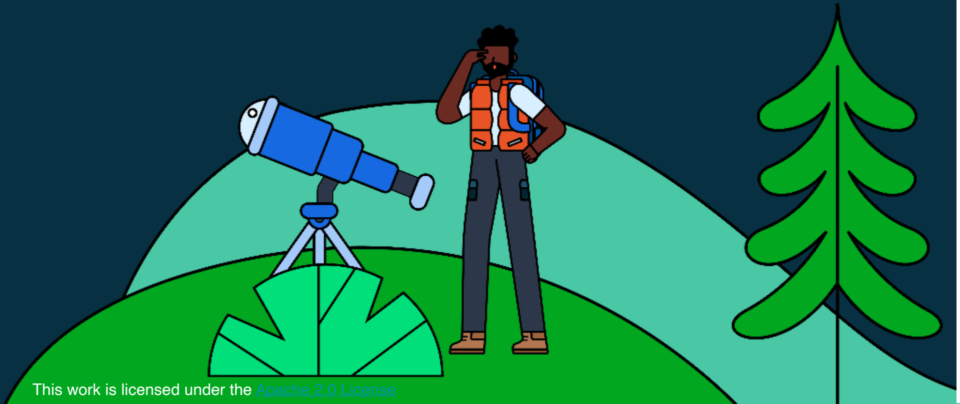
```
Text(
  "Hello Compose!",
  Modifier.background(Color.Magenta)
    .size(200.dp, 30.dp)
    .padding(5.dp)
    .alpha(0.5f)
    .clickable {
      // Called when Text clicked
    }
)
```

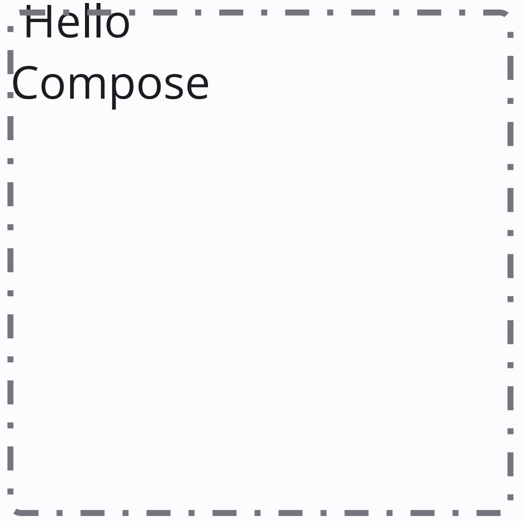




android

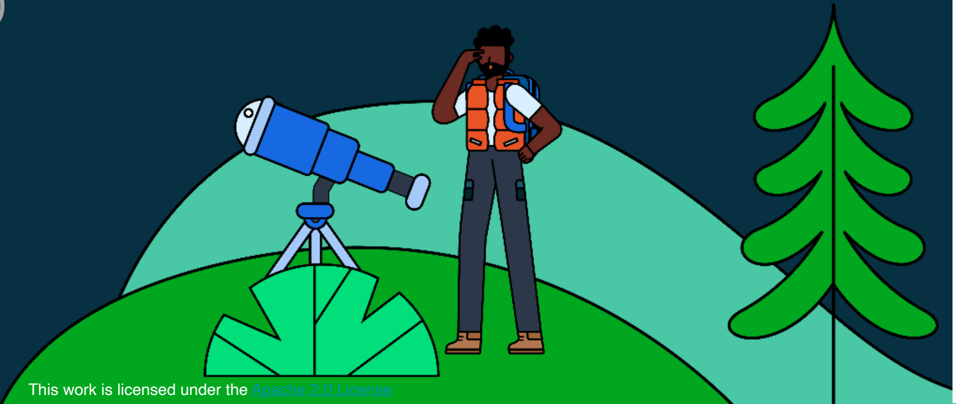
```
Box(Modifier.size(150.dp)) {
  Text("Hello Compose")
}
```

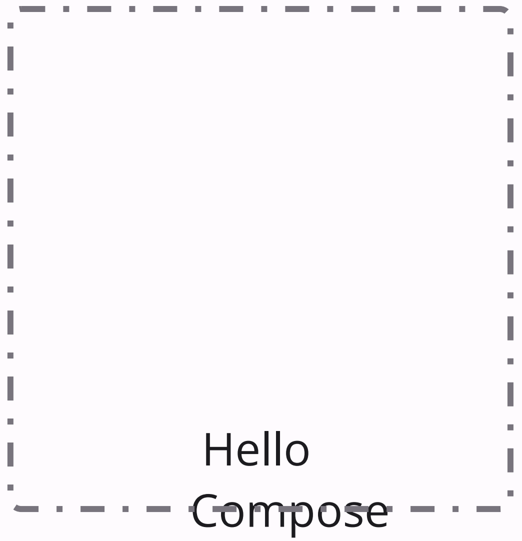




android

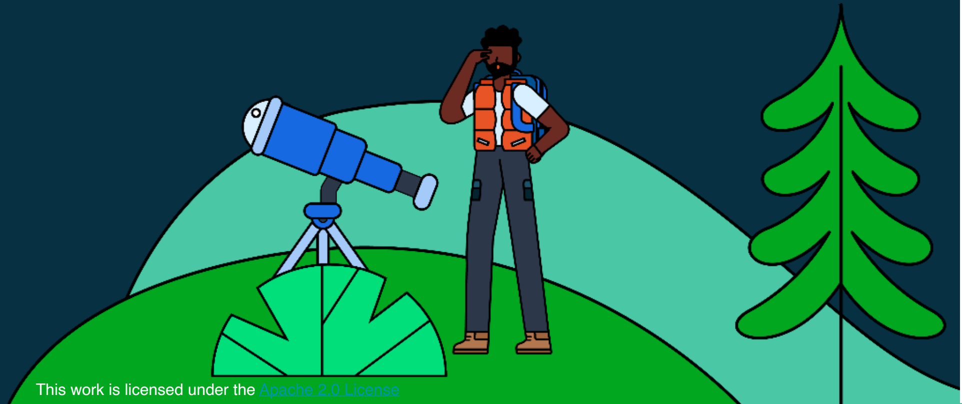
```
Box(Modifier.size(150.dp)) {
 Text(
  "Hello Compose!",
  Modifier.align(
   Alignment.BottomEnd
  )
 )
}
```



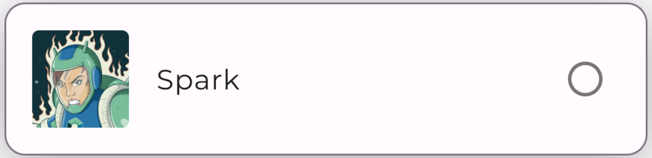


android

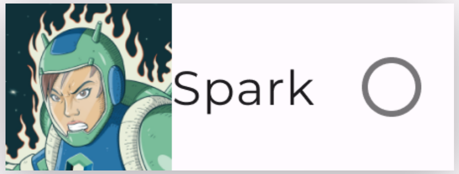
```
@Composable
fun SurveyAnswer(answer: Answer) {
  Row(...) {
    Image(answer.image)
    Text(answer.text)
    RadioButton(/* … */)
  }
}
```



Desired

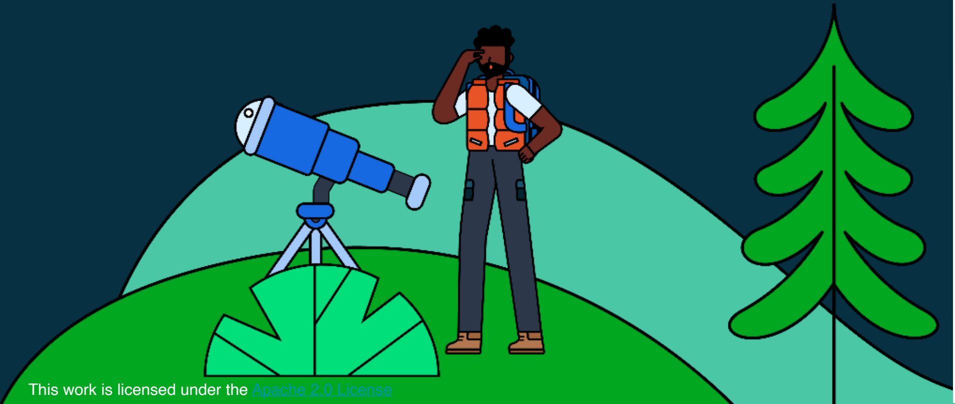


Current

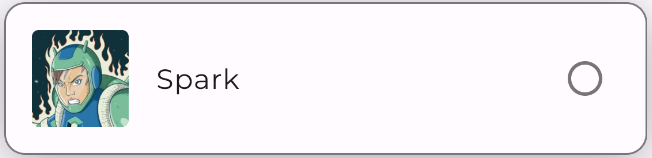


android

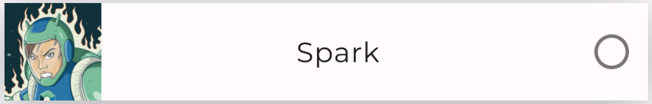
```
@Composable
fun SurveyAnswer(answer: Answer) {
  Row(
    Modifier.fillMaxWidth(),
    /* ... */
  ) {
    Image(answer.image)
    Text(answer.text)
    RadioButton(/* ... */)
  }
}
```



Desired



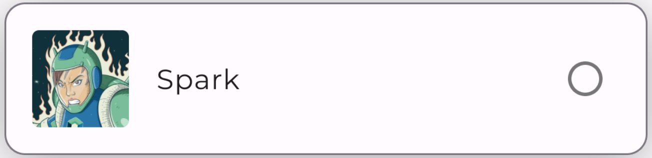
Current



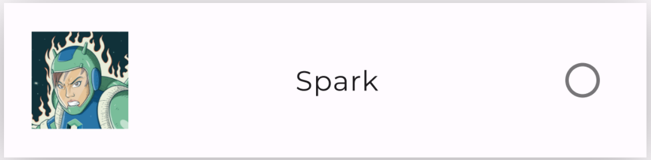
android

```
@Composable
fun SurveyAnswer(answer: Answer) {
  Row(
    Modifier.fillMaxWidth()
      .padding(16.dp),
    /* ... */
  ) {
    Image(answer.image)
    Text(answer.text)
    RadioButton(/* ... */)
  }
}
```

Desired

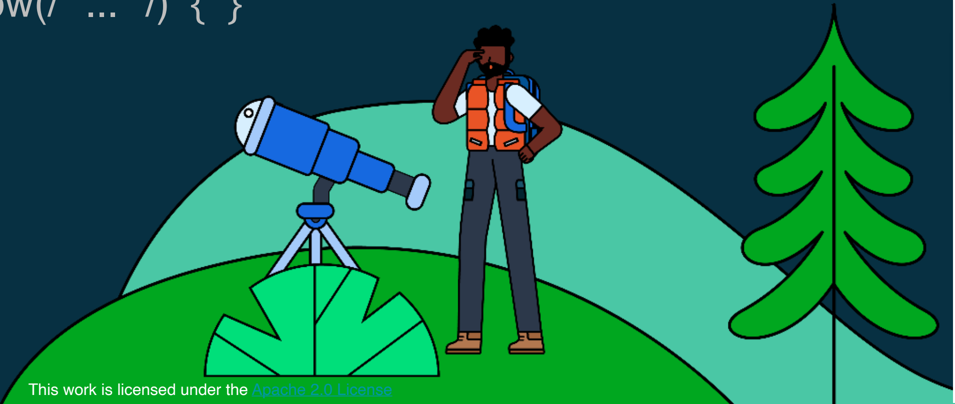


Current

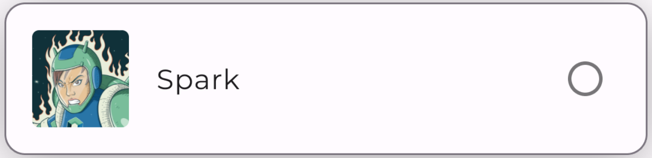


android

```
@Composable
fun SurveyAnswer(answer: Answer) {
  Surface(
   border = BorderStroke(
     1.dp,
     MaterialTheme.colorScheme.outline
  ),
   shape = MaterialTheme.shapes.small
  ) {
    Row(/* ... */)  {  }
  }
}
```

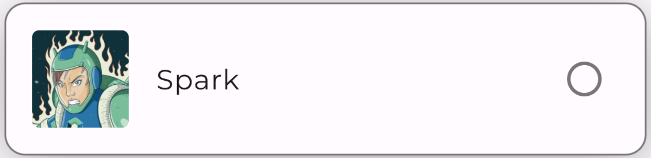


Desired

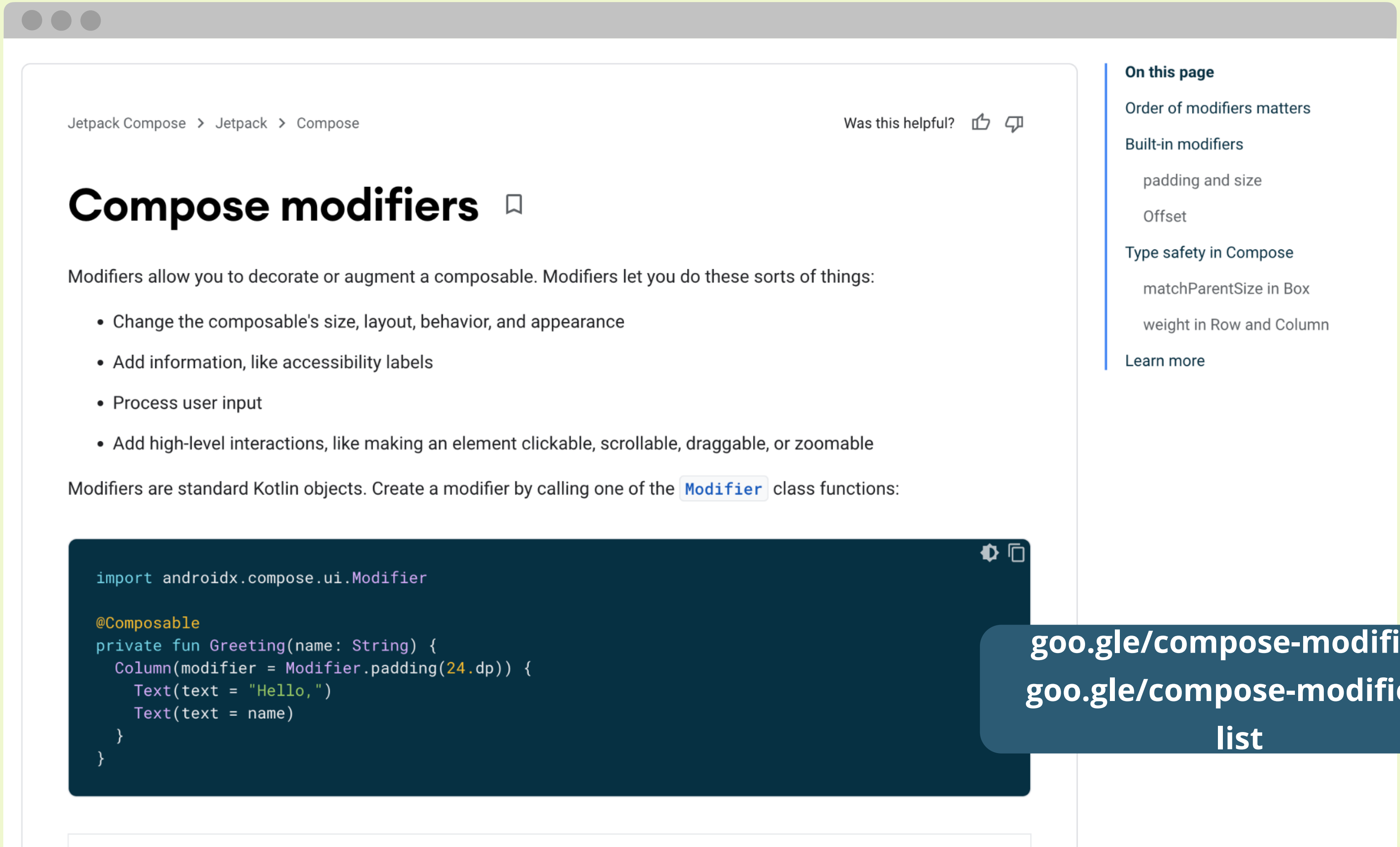


Spark

Current



Spark


Jetpack Compose > Jetpack > Compose

Was this helpful?

Compose modifiers

Modifiers allow you to decorate or augment a composable. Modifiers let you do these sorts of things:

- Change the composable's size, layout, behavior, and appearance
- Add information, like accessibility labels
- Process user input
- Add high-level interactions, like making an element clickable, scrollable, draggable, or zoomable

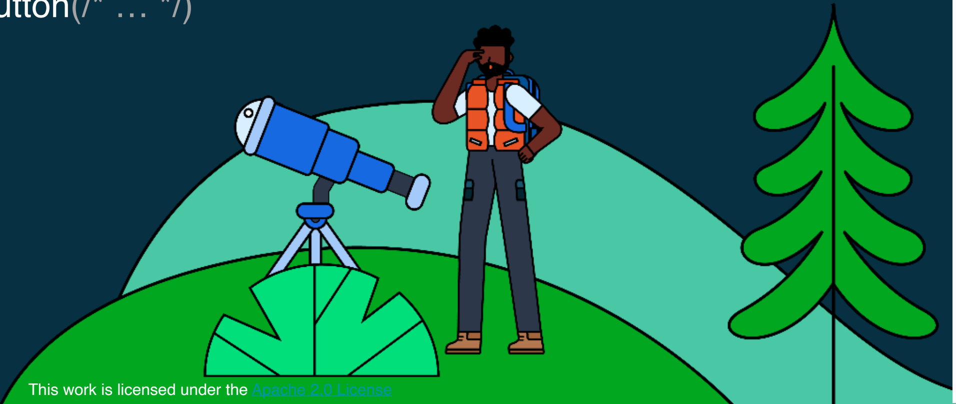
Modifiers are standard Kotlin objects. Create a modifier by calling one of the `Modifier` class functions:

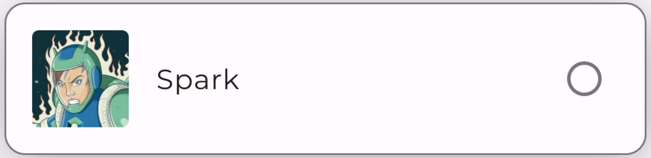
```kotlin
import androidx.compose.ui.Modifier

@Composable
private fun Greeting(name: String) {
  Column(modifier = Modifier.padding(24.dp)) {
    Text(text = "Hello,")
    Text(text = name)
  }
}
```

goo.gle/compose-modifiers

goo.gle/compose-modifiers-list

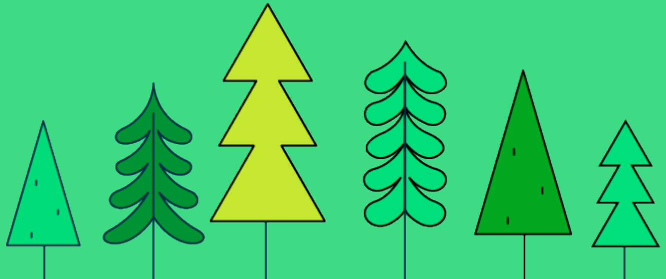
```
@Composable
fun SurveyAnswer(answer: Answer) {
  Surface(
   border = BorderStroke(
     1.dp,
     MaterialTheme.colorScheme.outline
  ),
   shape = MaterialTheme.shapes.small
  ) {
    Row(Modifier.fillMaxWidth().padding(16.dp)) {
      Image(answer.image)
      Text(answer.text)
      RadioButton(/* … */)
    }
  }
}
```





android

Compose Tooling



android