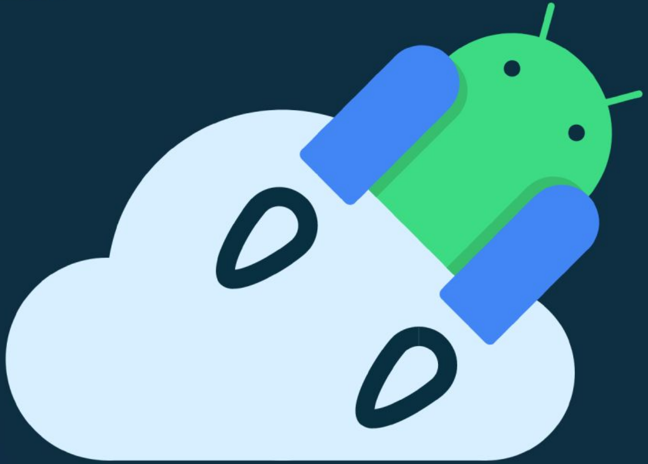


Android for Everyone

Getting Started with Jetpack Compose

Mobile is hard.

Jetpack Compose makes it easier to get started with Android.

Sierra

Veteran Android Developer
learning Jetpack Compose

Alexx

Frontend Developer
learning Jetpack Compose

What is Jetpack Compose?

Build better apps faster with Jetpack Compose.

Jetpack Compose is Android’s modern toolkit for building native UI.

Less Code

Accelerate Development

Intuitive

Powerful

What does “Modern native toolkit” mean?

- Declarative framework

Describe your UI

What does "Modern native toolkit" mean?

- Declarative framework
- All Kotlin all the time

$\xrightarrow{}$ Can build apps with high levels of complexity and polish

What does “Modern native toolkit” mean?

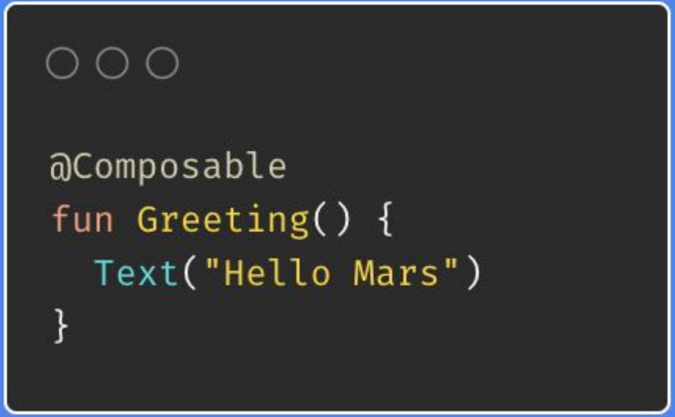
- Declarative framework
- All Kotlin all the time
- Still in Beta

the land of Declaration

Working with Jetpack Compose

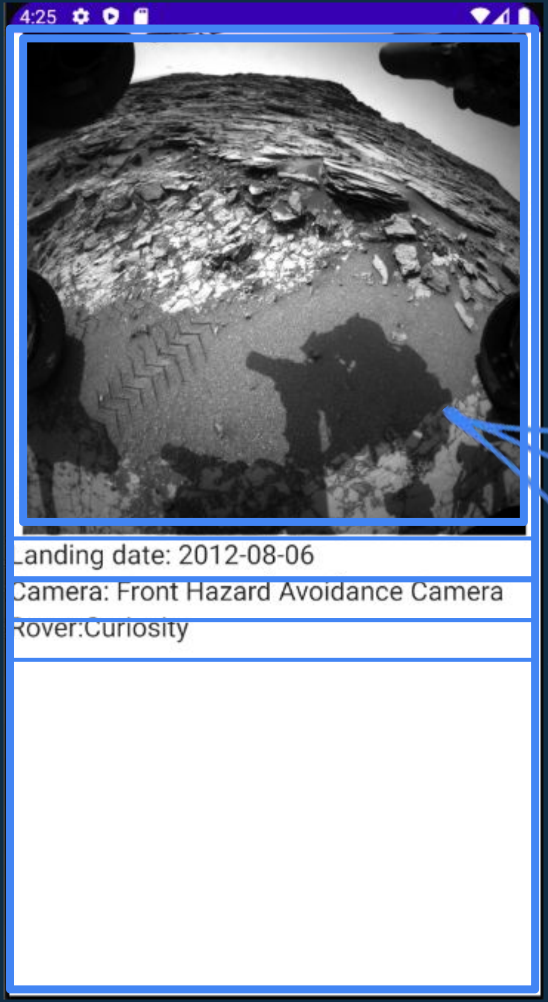
Anatomy of a composable

- @Composable notation
- A pure function that can take in parameters
- A composable can be used multiple times during the app
    - Each time it is used generates a new instance of the composable



```kotlin
@Composable
fun Greeting() {
  Text("Hello Mars")
}
```

Building the Details Screen


4:25

Landing date: 2012-08-06

Camera: Front Hazard Avoidance Camera

Rover:Curiosity

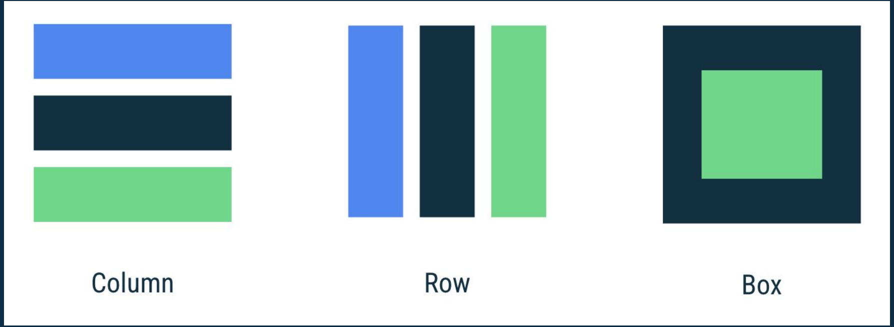
The entire screen is a composable

The image is a composable

The lines of text are also composables

Layouts are a breeze

```
@Composable
fun PhotoDetails(
  photo: Photo
) {
 Column {
  PhotoItem(photo = photo)
  Text(text = "Landing date:" +
    photo.rover.landingDate)
  Text(text = "Camera: " +
    photo.camera.fullName)
  Text(text = "Rover:" +
    photo.rover.name)
  }
}
```



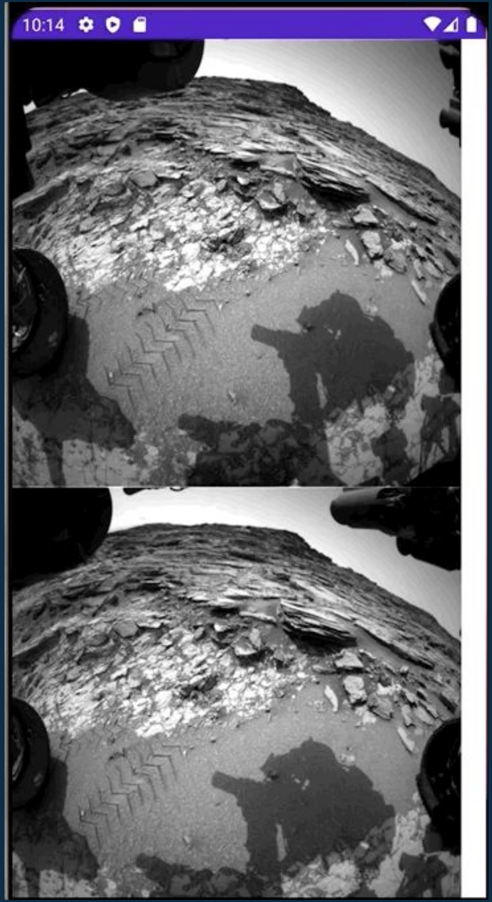
Lists are also breeze

```
Column(modifier = Modifier
  .verticalScroll(rememberScrollState())
) {
    photos.value.forEachIndexed { index,photo ->
        PhotoItem(photo = photo, onClick = {
            photoDetail = index
            showDetails = !showDetails }
        )
    }
}
```

LazyColumn to the rescue!

```kotlin
val photos = viewModel
    .photosState
    .collectAsState()

LazyColumn {
    items(photos.value) { photo ->
        PhotoItem(
            photo = photo,
        )
    }
}
```



Okay so how do we start customizing our composables?

Height + width

Background + shape

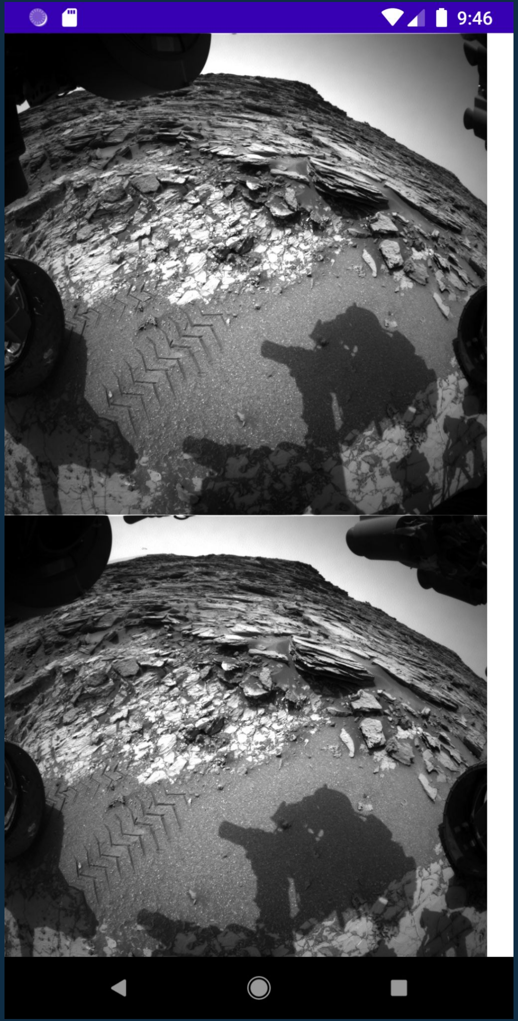
Modifier

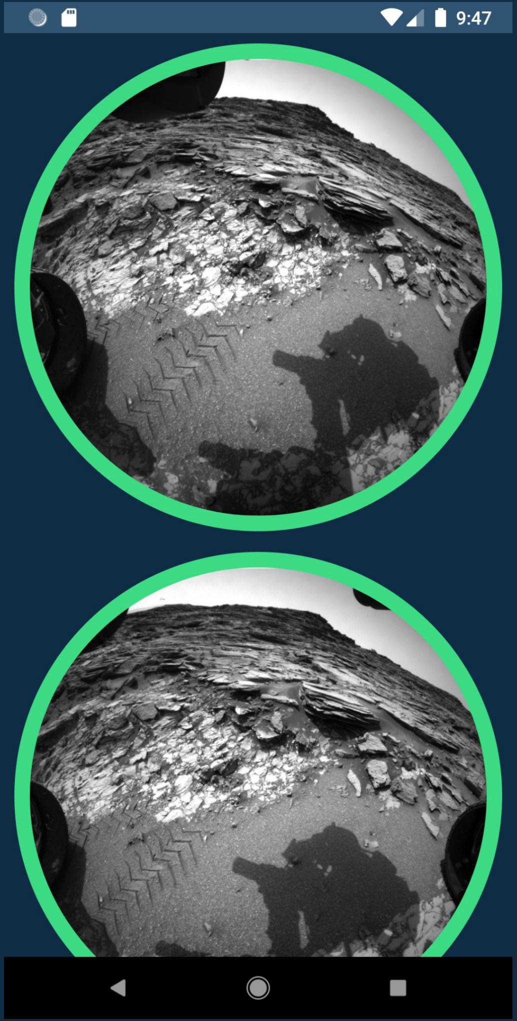
Padding + elevation

Clickable + focusable

How do we use a modifier?

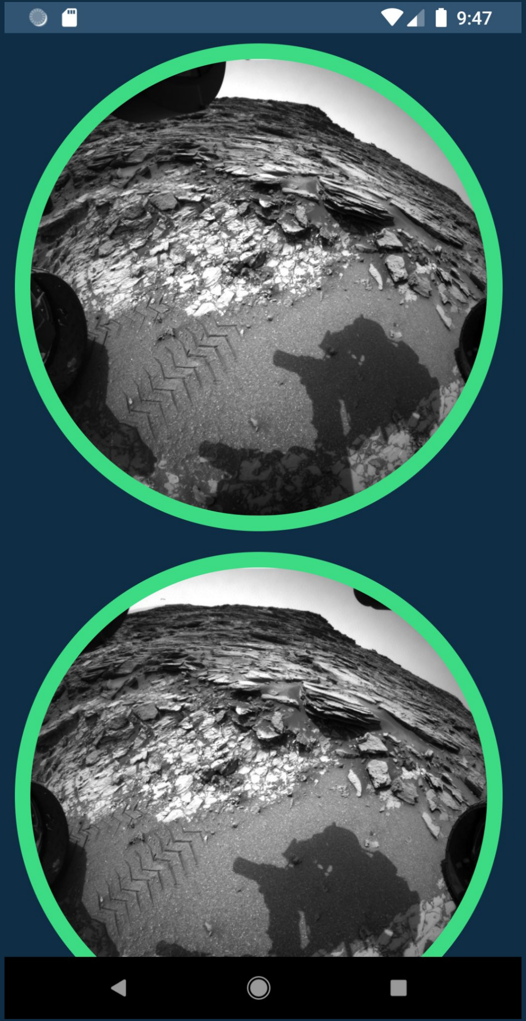
```
@Composable
fun Image(
    painter: Painter,
    contentDescription: String?,
    modifier: Modifier = Modifier,
    alignment: Alignment = Alignment.Center,
    contentScale: ContentScale = ContentScale.Fit,
    alpha: Float = DefaultAlpha,
    colorFilter: ColorFilter? = null
)
```





Let’s make our app match our presentation

```
LazyColumn(
    modifier = Modifier.fillMaxWidth(),
    horizontalAlignment =
Alignment.CenterHorizontally
) {
    items(photos.value) { photo ->
         Image(
              painter = ourPainter,
              contentDescription =
ourContentDescription,
              modifier = Modifier
                   .padding(8.dp)
                   .clip(CircleShape)
                   .background(color = green)
                   .padding(12.dp)
                   .clip(CircleShape)
         )
    }
}
```



What about the background?

Theming is easier!

```
setContent {
  MyTheme(
      darkTheme = true
) {
    MyApp { MainScreen() }
  }
}
```

```
@Composable
fun MyTheme(darkTheme: Boolean, content) {
  val colors = if (darkTheme) DarkColorPalette
               else LightColorPalette

  MaterialTheme(
      colors = colors,
      content = content
  )
}
```

How do we pull all this together?

State

State

Uni-directional data flow

remember { mutableStateOf() }

State hoisting

```kotlin
val photos = viewModel
    .photosState
    .collectAsState()
var showDetails by remember { mutableStateOf(false) }
var photoDetail by remember { mutableStateOf(-1) }

PhotoList(
    photos = photos.value,
    onClick = { index ->
        photoDetail = index
        showDetails = !showDetails
    }
)

if (showDetails)
    PhotoDetails(
        photo = photos.value[photoDetail],
        isShown = { showDetails = false }
    )
```

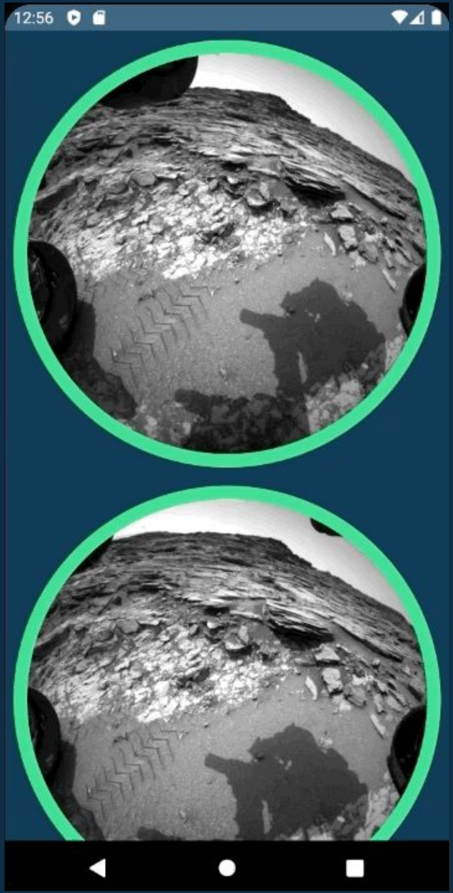
Declare our state variables

Use state in our composables to control UI

Recomposition - what you need to know

- Composable functions can execute in any order
- Composable functions can run in parallel
- Recomposition is smart and optimistic
- Composable functions might run quite frequently

The final product



Wrapping up... Why do we love Compose?

- Integration

1. with your app
2. with libraries
3. with MVVM
   (but also other architectures!)

Wrapping up… Why do we love Compose?

- Integration
- All Kotlin all the time!

Compose can be used in a lot of different places!

Wrapping up… Why do we love Compose?

- Integration
- All Kotlin all the time!
- Great time to get started!

Thank you!

Where do you find us?

@_sierraObryan

sierraobryan.com

@_alexxMitchell

alexxmitchell.com

Resources

https://www.droidcon.com/media-detail?video=543570509

https://developer.android.com/courses/pathways/compose

https://developer.android.com/jetpack/compose/mental-model

https://www.raywenderlich.com/books/jetpack-compose-by-tutorials/v1.1/chapters/2-learning-jetpack-compose-fundamentals

Questions?