

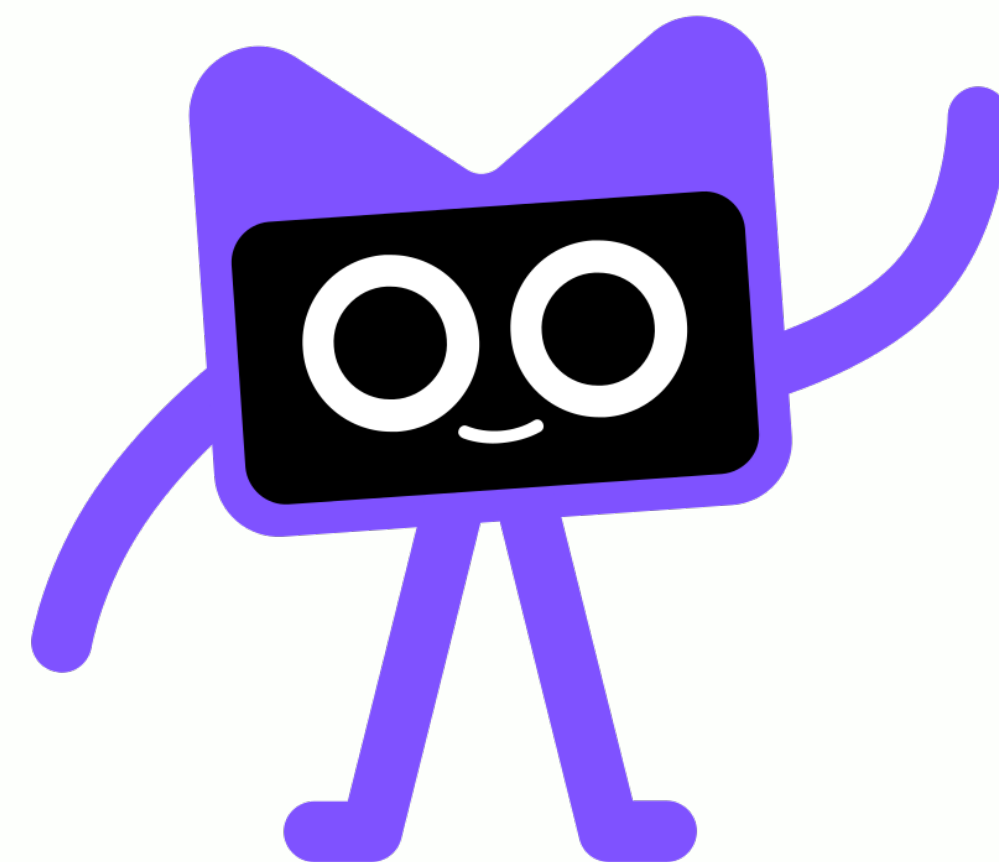


UNIVERSIDADE
FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Jetpack Compose: fundamentos

QXD0276 - Desenvolvimento de Software para Dispositivos Móveis

Prof. Bruno Góis Mateus (brunomateus@ufc.br)



Conteúdo

- Introdução
- JetPack Compose
- Composables
- Layout básicos
- Preview

Introdução

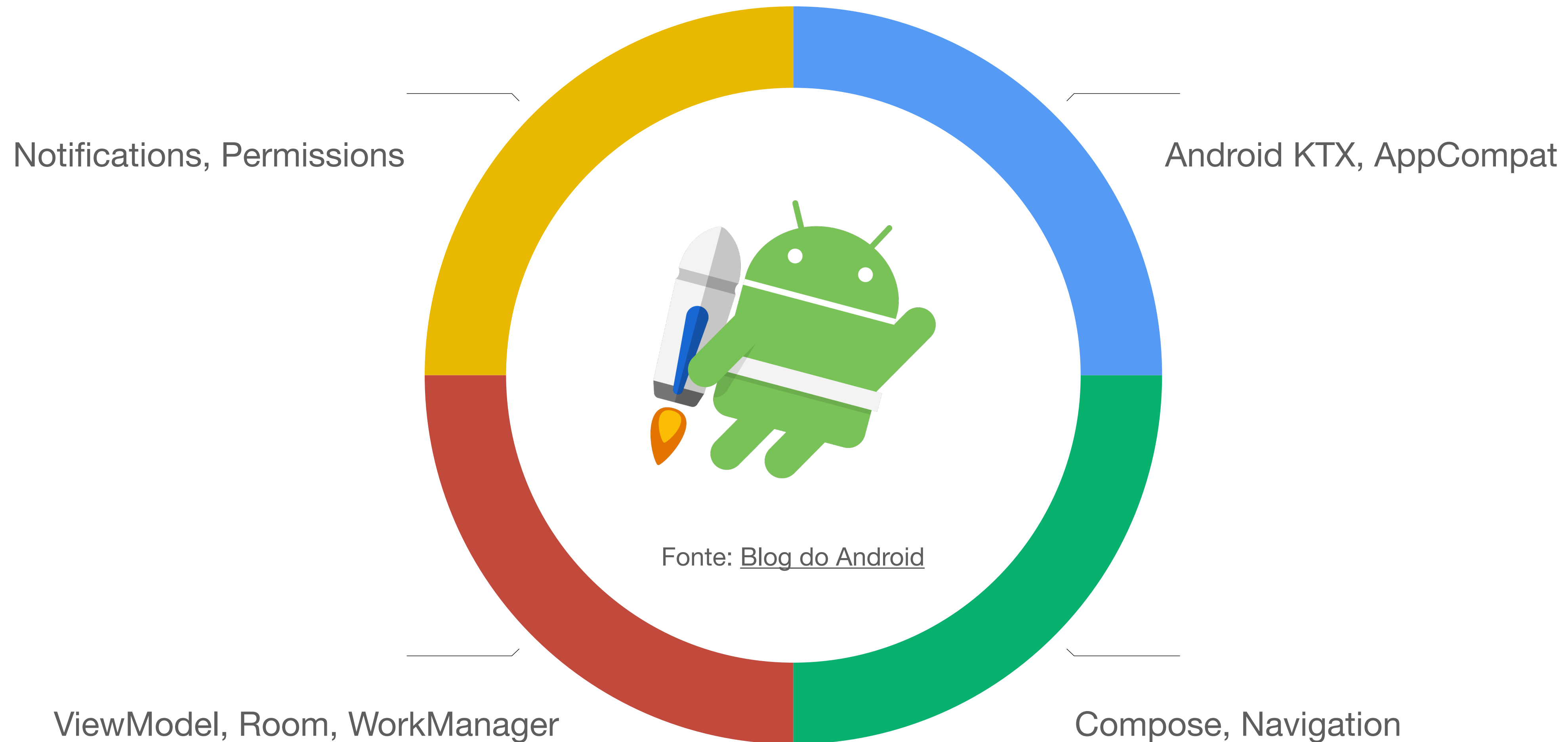
O Jetpack é um pacote de bibliotecas que ajuda desenvolvedores a seguir as práticas recomendadas, reduzir códigos boilerplate e programar códigos que funcionam de maneira consistente em diferentes dispositivos e versões do Android.



Fonte: [Documentação do Android](#)

Introdução

Android Jetpack



Introdução

Android Jetpack

- Compose → forma moderna de criar interfaces
 - Substitui gradualmente o XML
- ViewModel + LiveData/StateFlow → separação entre lógica e interface
- Navigation Component → navegação entre telas simplificada
- Room → acesso a banco de dados SQLite mais fácil
- WorkManager → tarefas em segundo plano

Jetpack Compose

Jetpack Compose

O que é?

- É um toolkit moderno para criar interfaces nativas
- 🔥 Oficial desde 2020, atualmente é o padrão recomendado
- 🧩 UI como código → mais simples, legível e reativo
- ⚡ Integração direta com ViewModel, StateFlow, Room
- 🎨 Suporte nativo a Material Design 3 e Dark Mode



Fonte: [Blog do Android](#)

Com o **Compose**, fica mais fácil criar e manter a interface do app, fornecendo uma **API declarativa** que permite renderizar a interface do app sem modificar de forma imperativa as visualizações do front-end.

Fonte: [Documentação oficial do Android](#)

Jetpack Compose

O paradigm de programação declarativa

- Historicamente, no Android, uma interface era representada por uma **hierarquia de view (View API)** que por suas vez eram uma árvore de **widgets**
- Com a necessidade de atualização o caminho comum era manipular a árvore
 - **findViewById, button.setText, container.addChild(View)**
 - Algo similar com que era feito no desenvolvimento web no passado

Jetpack Compose

O paradigma de programação declarativa

```
class MainActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
  
        // Find the TextView and Button by their IDs  
        val myTextView: TextView = findViewById(R.id.myTextView)  
        val myButton: Button = findViewById(R.id.myButton)  
  
        // Set an OnClickListener for the button  
        myButton.setOnClickListener {  
            // Change the text of the TextView when the button is clicked  
            myTextView.text = "Button Clicked!"  
        }  
    }  
}
```

Jetpack Compose

O paradigma de programação declarativa

- Nos últimos anos a indústria moveu-se em direção ao modelo de UI declarativa
- Funciona regerando toda a tela do zero e aplicando apenas as alterações necessária
- Evita a complexidade de atualizar manualmente uma hierarquia de views e seus estados



Imperative UI vs Declarative UI

Imperative UI



Step-by-step instructions



Manual update



XML (UI) + Code (Logic)

vs

Declarative UI



Describe the final UI state



Auto Update



Unified with Kotlin

#PARTHA

Jetpack Compose

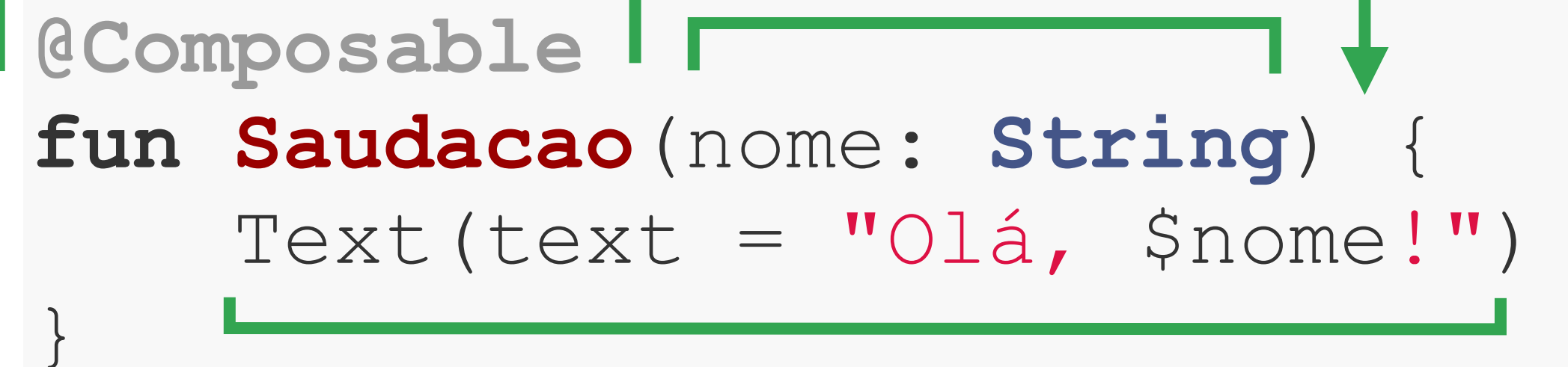
Principais conceitos

- Composable functions: funções que definem UI
- State: o estado atual que é refletido na UI reativa
- Layouts e Modifiers: controlam o visual e estilização
- Navegação: Feita via Navigation Compose

Composables

Composables

- São funções anotadas com **@Composable**
 - Informa ao compilador que a função é usada para converter dados em UI
 - Recebem dados e emitem um elemento da UI
 - Não possuem retorno (**Unit**)
- Blocos reutilizáveis / Blocos de construção
- Podem compor outras funções



The diagram illustrates the `@Composable` annotation. A green box highlights the `@Composable` annotation. A green arrow points from the annotation to the `Saudacao` function. Another green box highlights the `Text` function call within the `Saudacao` function.

```
@Composable  
fun Saudacao (nome: String) {  
    Text (text = "Olá, $nome!")  
}
```


Composables

Recomposition

- Em um modelo de IU imperativo, para alterar um widget, chamamos um setter no widget para alterar seu estado interno
- No Compose, chamamos novamente a função Composable, com novos dados
 - Isso faz com que a função seja recomposta (*recomposition*)
 - Os widgets emitidos pela função são redesenhados, se necessário, com os novos dados
- De forma inteligente, apenas componentes alterados são atualizados

Composables

Nomeando composables

✓ Escritos em CamelCase

✓ DEVE ser um substantivo: `DoneButton()`

Essa convenção de nomenclatura promove e reforça esse modelo mental declarativo

✗ NÃO deve ser um verbo ou locução verbal: `DrawTextField()`

✗ NÃO deve ter uma preposição substantiva: `TextFieldWithLink()`

✗ NÃO deve ter um adjetivo: `Bright()`

✗ NÃO deve ter um advérbio: `Outside()`

✓ Substantivos PODEM ser prefixados por adjetivos descritivos: `RoundIcon()`

Composables

- O Compose oferece vários componentes que implementam o Material Design
 - Esses componentes (`@Composable`) podem utilizados para compor outras funções
 - Ações: `Button`, `Floating action button`, `Icon Button`
 - Comunicação: `Badge`, `Progress Indicators`, `Tooltips`
 - Contenção: `Cards`, `Carousel`, `Dialogs`
 - Navegação: `App bars`, `Navigation Bars`
- [Confira a lista completa](#)

Composables

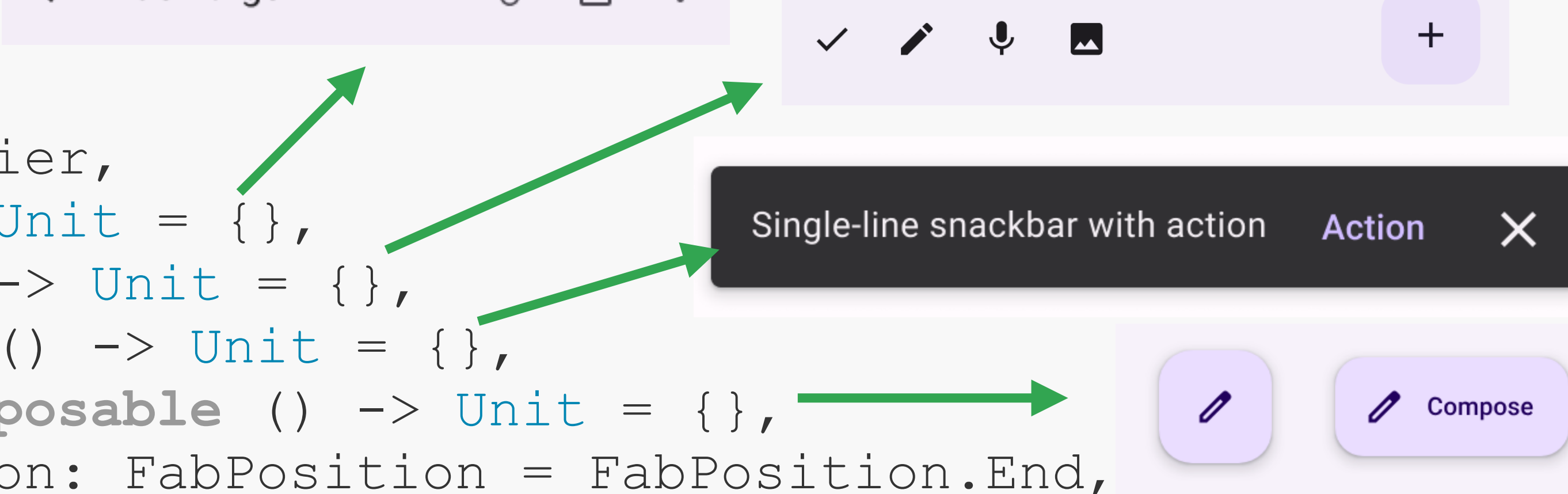
Scaffold

- Prover uma API simples para estruturar uma aplicação conforme o Material Design
- Aceita diversos composables como parâmetros:
 - Ex: topBar, bottomBar, floatingActionButton

Composables

Scaffold

```
@Composable
fun Scaffold(
    modifier: Modifier = Modifier,
    topBar: @Composable () -> Unit = {},
    bottomBar: @Composable () -> Unit = {},
    snackbarHost: @Composable () -> Unit = {},
    floatingActionButton: @Composable () -> Unit = {},
    floatingActionButtonPosition: FabPosition = FabPosition.End,
    containerColor: Color = MaterialTheme.colorScheme.background,
    contentColor: Color = contentColorFor(containerColor),
    contentWindowInsets: WindowInsets = ScaffoldDefaults.contentWindowInsets,
    content: @Composable (PaddingValues) -> Unit
): Unit
```



The diagram illustrates the visual components of the Scaffold composable and their mapping to the code parameters:

- Top Bar:** A light purple bar containing a back arrow, the text "Title Large", and three icons (attachment, calendar, menu). It corresponds to the `topBar` parameter.
- Bottom Bar:** A light purple bar containing a checkmark, edit, voice, and image icons, followed by a plus icon. It corresponds to the `bottomBar` parameter.
- Snackbar:** A dark grey bar with the text "Single-line snackbar with action", the word "Action" in purple, and a close icon. It corresponds to the `snackbarHost` parameter.
- Floating Action Button (FAB):** Two light purple buttons with rounded corners. The first has an edit icon, and the second has an edit icon and the text "Compose". It corresponds to the `floatingActionButton` parameter.

Green arrows point from the code parameters to their respective UI components: `topBar` to the top bar, `bottomBar` to the bottom bar, `snackbarHost` to the snackbar, and `floatingActionButton` to the FAB.

Exemplo completo

Top app bar

This is an example of a scaffold. It uses the Scaffold composable's parameters to create a screen with a simple top app bar, bottom app bar, and floating action button.

It also contains some basic inner content, such as this text.

You have pressed the floating action button 4 times.



Bottom app bar

Composables

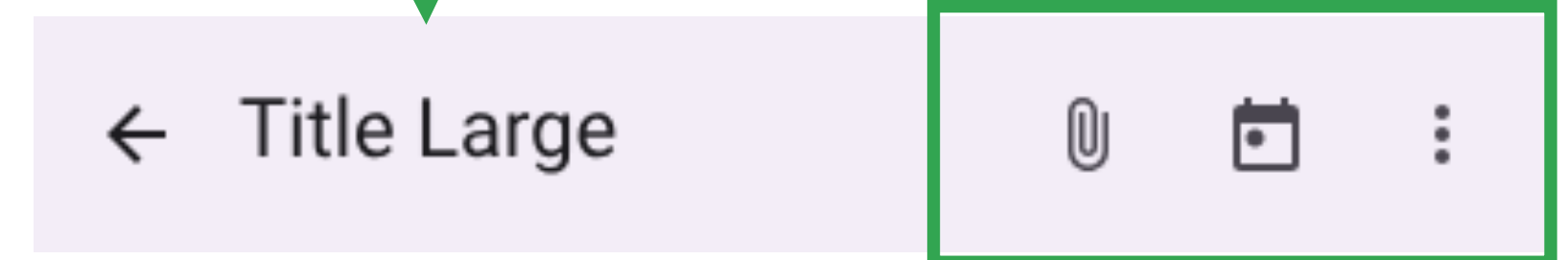
Barras

Tipo	Aparência	Propósito
Top app bar	Na parte de cima da tela	Oferece acesso a tarefas e informações importantes. Geralmente hospeda um título, itens de ação principais e determinados itens de navegação.
Bottom app bar	Na parte de baixo da tela	Normalmente inclui itens de navegação principais. Também pode dar acesso a outras ações importantes, como um botão de ação flutuante contido.

Composables

Top bar

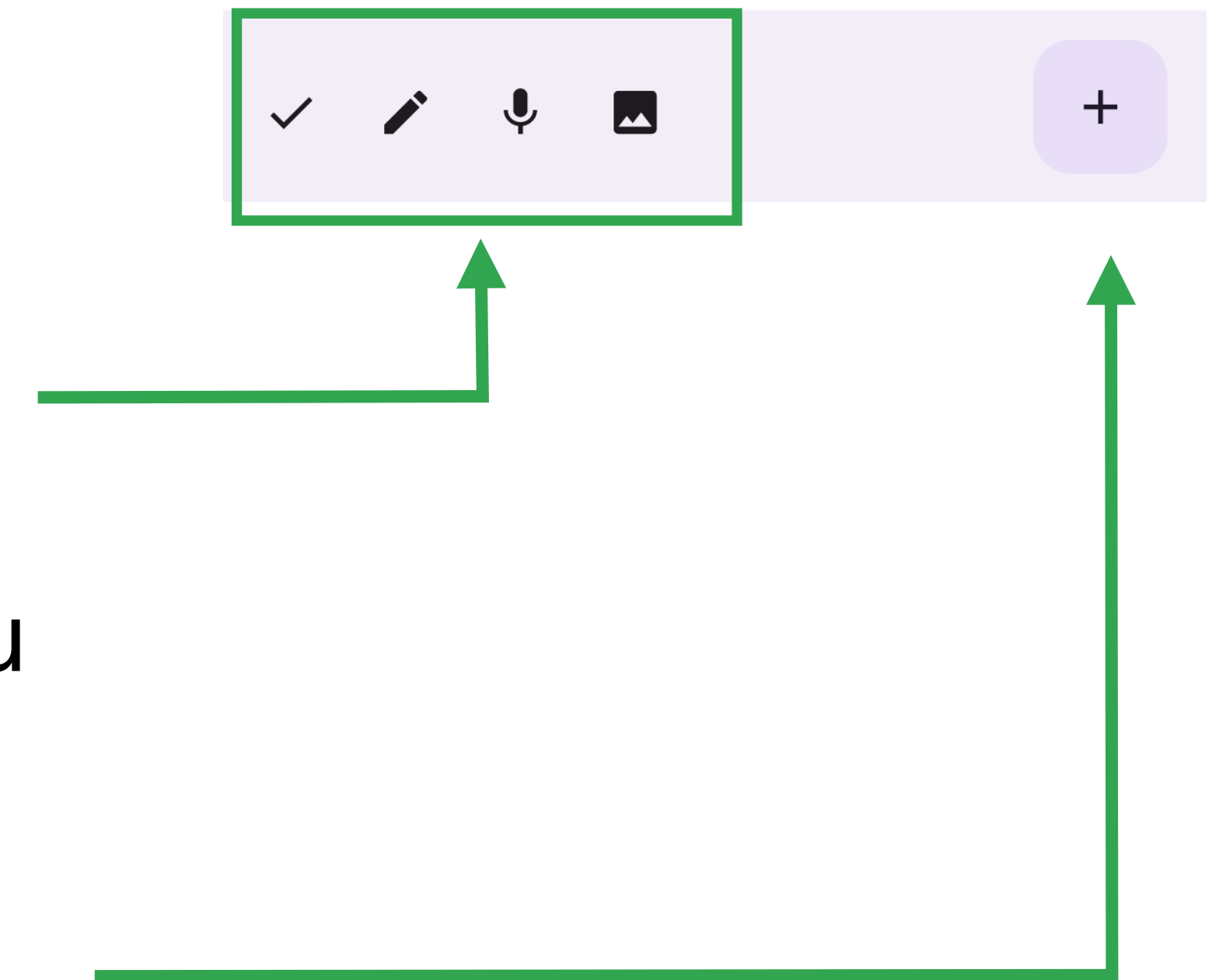
- Principais parâmetros
 - **title**: o texto que aparece na barra de apps
 - **navigationIcon**: o ícone principal para navegação
 - Aparece à esquerda da barra de apps.
 - **actions**: ícones que dão ao usuário acesso a ações importantes
 - Aparecem à direita da barra de apps.
- **scrollBehavior**: determina como a barra de apps superior responde à rolagem do conteúdo interno do scaffold.
- **colors**: determina como a barra de apps aparece.



Composables

Bottom bar

- Principais parâmetros
 - **actions**: uma série de ícones que aparecem no lado esquerdo da barra
 - Normalmente, são ações principais da tela ou itens de navegação
 - **floatingActionButton**: o botão de ação flutuante que aparece no lado direito da barra



Composables

Text

- Texto é uma peça central em qualquer interface
- A maneira mais simples de mostrar um texto é usando Text com uma String

```
@Composable
fun SimpleText() {
    Text("Hello World")
}
```

Composables

Text

- A boa prática indica que ao invés de String hardcoded, String resources devem ser utilizadas

```
@Composable
fun StringResourceText() {
    Text(stringResource(R.string.hello_world))
}
```

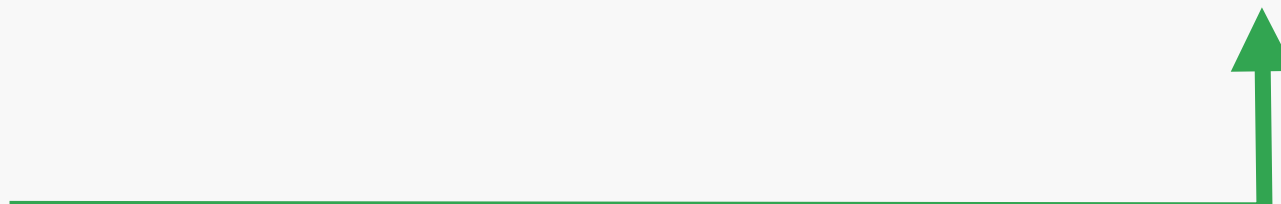
- Além disso, podemos estilizar o texto por meio de diversos parâmetros
 - color, fontSize, fontStyle, fontWeight, shadow ...
- Também é possível adicionar múltiplos estilos em um parágrafo

Composables

Text

```
@Composable
fun Text(
    text: AnnotatedString,
    modifier: Modifier = Modifier,
    color: Color = Color.Unspecified,
    fontSize: TextUnit = TextUnit.Unspecified,
    fontStyle: FontStyle? = null,
    fontWeight: FontWeight? = null,
    fontFamily: FontFamily? = null,
    letterSpacing: TextUnit = TextUnit.Unspecified,
    textDecoration: TextDecoration? = null,
    textAlign: TextAlign? = null,
    lineHeight: TextUnit = TextUnit.Unspecified,
    overflow: TextOverflow = TextOverflow.Clip,
    softWrap: Boolean = true,
    maxLines: Int = Int.MAX_VALUE,
    minLines: Int = 1,
    inlineContent: Map<String, InlineTextContent> = mapOf(),
    onTextLayout: (TextLayoutResult) -> Unit = {},
    style: TextStyle = LocalTextStyle.current
): Unit
```

Pode ser usada com fontes disponíveis no Google Fonts



Composables

Image

- No Android, existem algumas maneira de renderizar algo na tela
 - Usando vetores, bitmap ou desenhando propriamente
- O componente **Image** é utilizado para mostrar gráficos na tela
- A maneira mais simples é carregar uma imagem do disco
- Para isso é utilizando **painterResource**
 - Atualmente suporta: **AnimatedVectorDrawable**, **BitmapDrawable** (PNG, JPG, WEBP), **ColorDrawable**, **VectorDrawable**
 - Não é necessário conhecer o tipo do asset

Composables

Image

```
@Composable
fun ImageFromDisk () {
    Image (
        painter = painterResource(id = R.drawable.dog),
        contentDescription = stringResource(id = R.string.dog_content_description)
    )
}
```

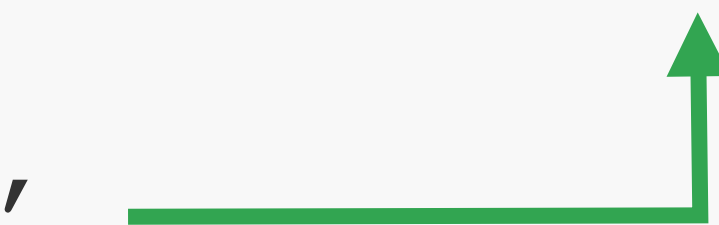
- Para que sua aplicação seja acessível, é preciso definir o **contentDescription**
 - É utilizado pelo TalkBack
 - Passe **null** quando a imagem for puramente decorativa

Composables

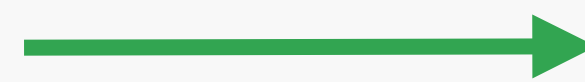
Image

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

[Exemplos de uso na documentação oficial](#)



[Exemplos de uso na documentação oficial](#)



Composables

Icons

- Utilizado para desenhar um ícone de uma cor só aderente ao Material Design
 - É preciso adicionar a biblioteca Compose Material ou Compose Material 3

```
@Composable
fun SimpleIcon () {
    Icon(
        painter = painterResource(R.drawable.baseline_directions_bus_24),
        contentDescription = stringResource(id = R.string.bus_content_description)
    )
}
```

É possível baixar os ícones mais atualizados no [Google Fonts](#)

Layout básicos

Layouts básicos

- O JetPack Compose transforma estados em elementos da UI da seguinte forma:
 1. Composição dos elementos
 2. Disposição dos elementos
 3. Desenho dos elementos



Layouts básicos

- Uma função composable pode emitir diversos elementos do UI
- Sem informação sobre como dispô-los, podemos ter problemas

```
@Composable
fun ArtistCard() {
    Text("Alfred Sisley")
    Text("3 minutes ago")
}
```

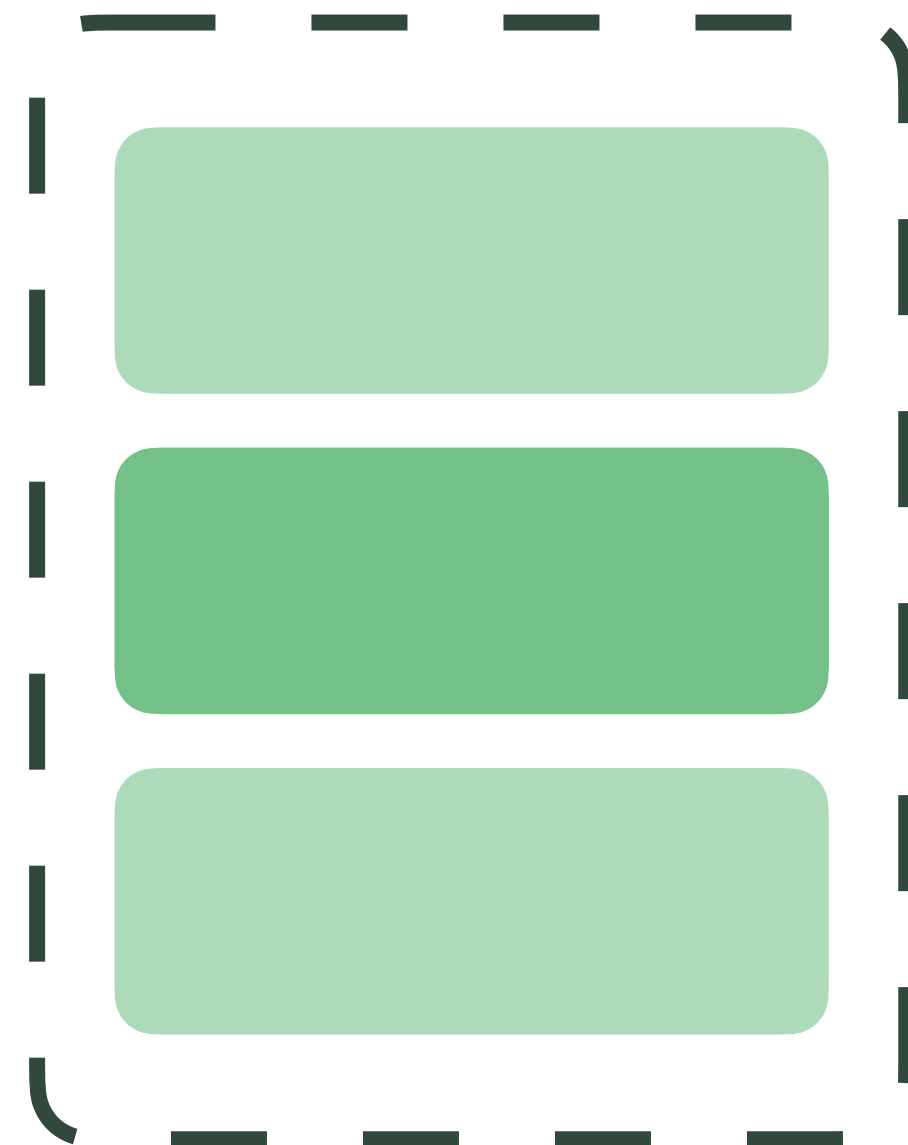
3 minutes ago
Alfred Sisley

Fonte: [Documentação oficial](#)

Layouts básicos

- Compose prover uma série de layouts prontos para uso
 - Na maioria dos casos é o que vamos utilizar

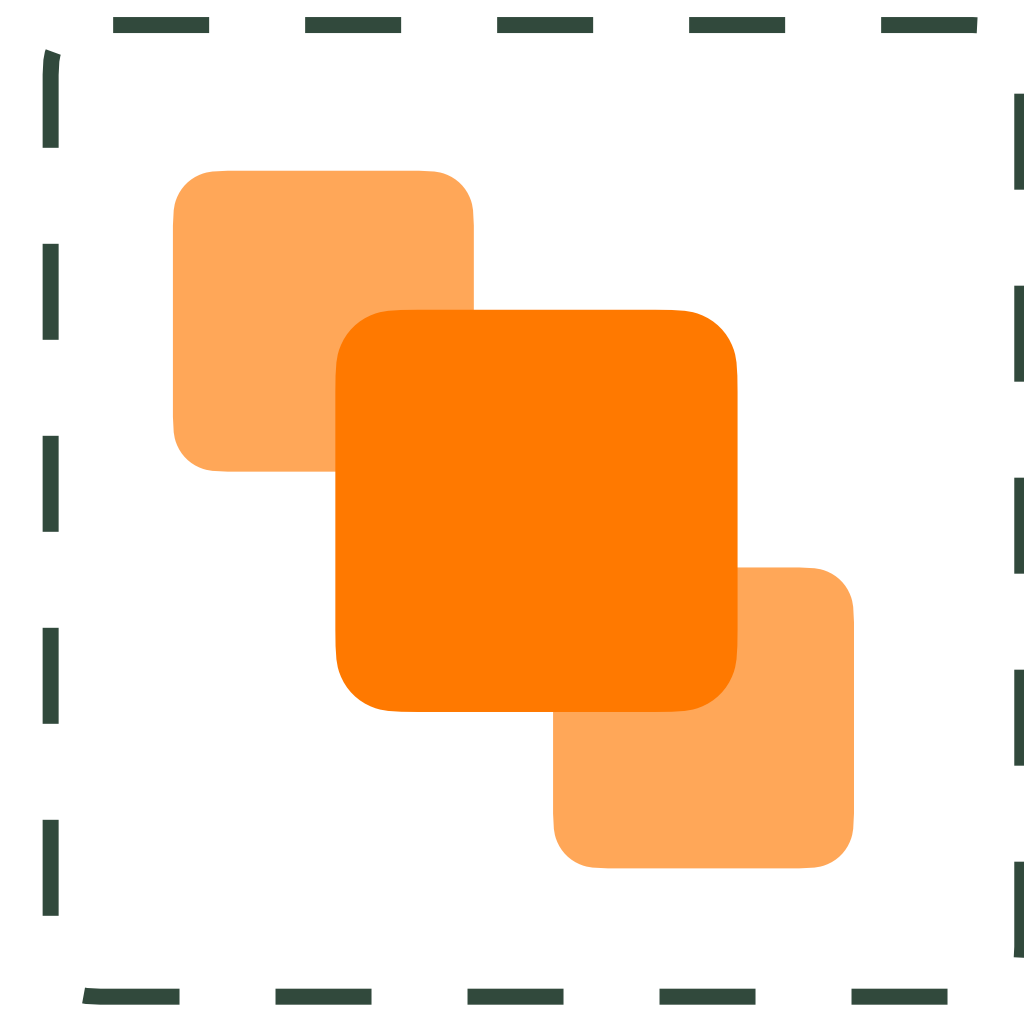
Column



Row



Box



Layouts básicos

Column vs Row

```
@Composable
fun ArtistCardColumn() {
    Column {
        Text("Alfred Sisley")
        Text("3 minutes ago")
    }
}
```

Alfred Sisley

3 minutes ago

Fonte: [Documentação oficial](#)

```
@Composable
fun ArtistCardRow(artist: Artist) {
    Row(verticalAlignment = Alignment.CenterVertically) {
        Image(bitmap = artist.image, contentDescription = "Artist image")
        Column {
            Text(artist.name)
            Text(artist.lastSeenOnline)
        }
    }
}
```



Alfred Sisley

3 minutes ago

Fonte: [Documentação oficial](#)

Layouts básicos

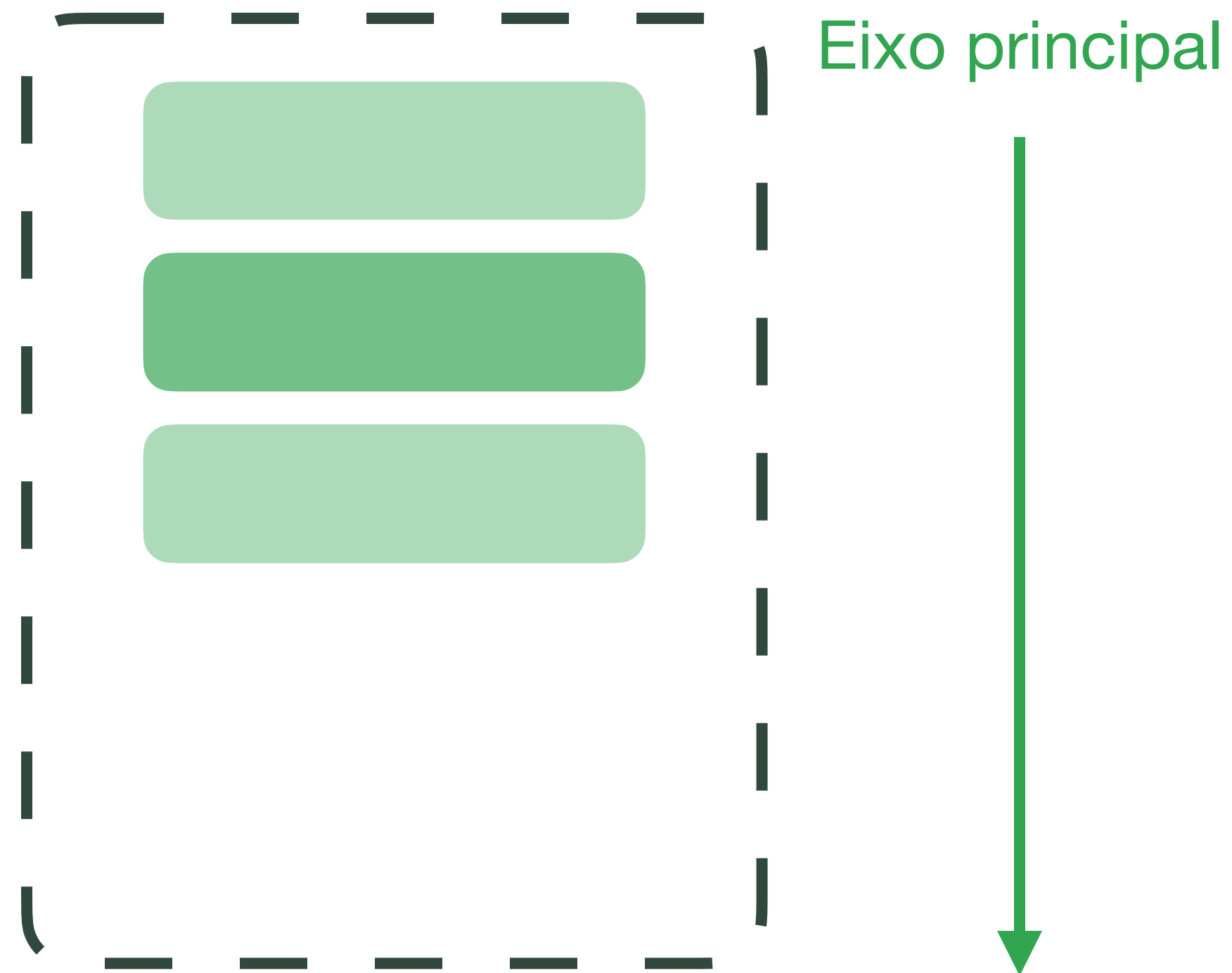
Column vs Row

- Existem dois conceitos importantes quando lidamos com posicionamento em Column e Row
 - Arrangement
 - Controla o espaço entre o filhos no eixo principal
 - Alignment
 - Controla o espaço entre os filhos no eixo transversal

Layouts básicos

Column - Arrangement vs Alignment

```
Column (  
  verticalArrangement = Arrangement.Top  
  horizontalAlignment = Alignment.End,  
) { /*...*/ }
```



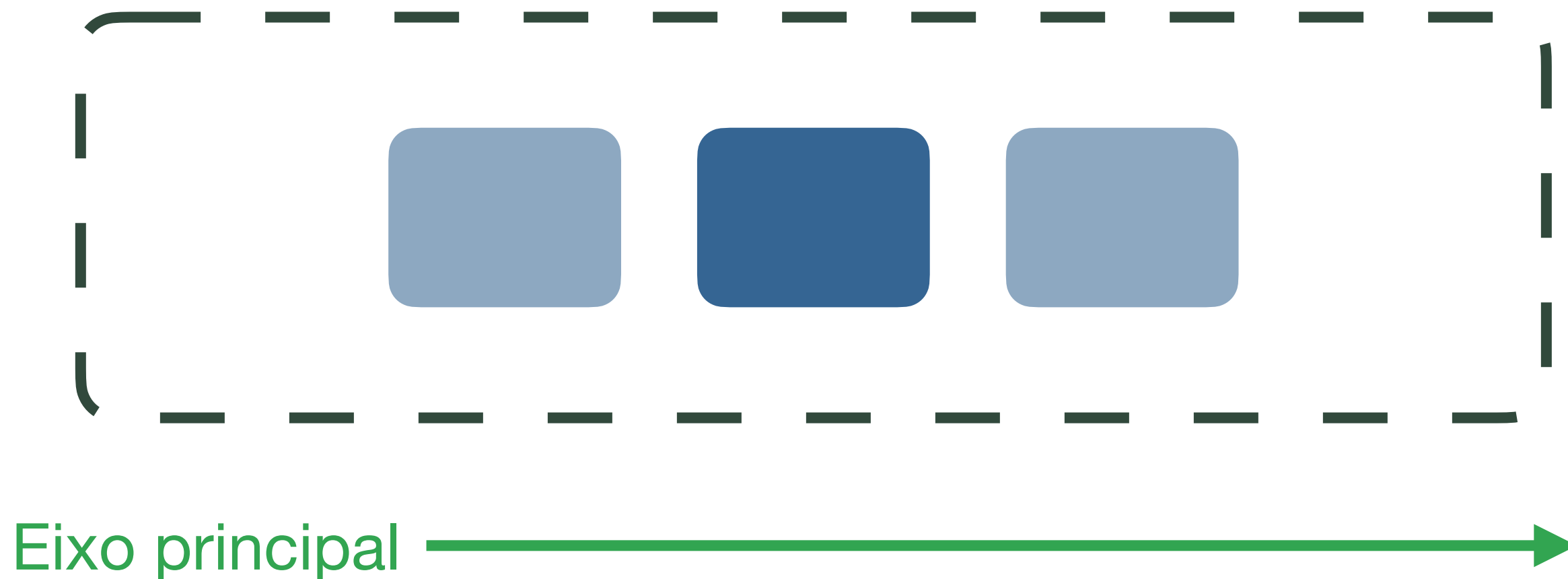
Equal Weight Space Between Space Around Space Evenly Top Center Bottom



Layouts básicos

Row - Arrangement vs Alignment

```
Row (  
  horizontalArrangement = Arrangement.Center  
  verticalAlignment = Alignment.Bottom,  
) { /*...*/ }
```



Equal Weight

Space Between

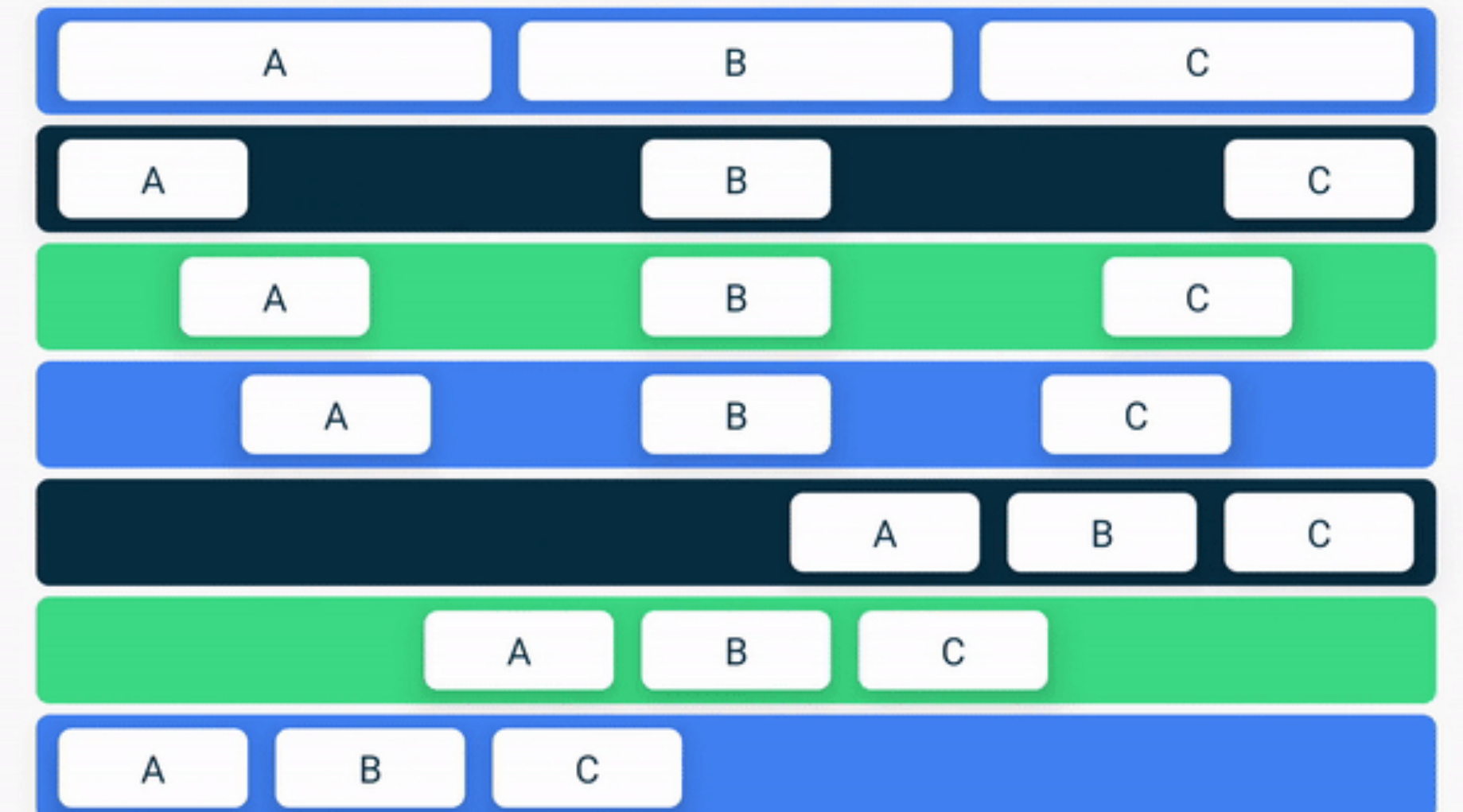
Space Around

Space Evenly

End (LTR)

Center

Start (LTR)



Layouts básicos

Box

```
@Composable
fun ArtistAvatar(artist: Artist) {
    Box {
        Image(bitmap = artist.image, contentDescription = "Artist image")
        Icon(Icons.Filled.Check, contentDescription = "Check mark")
    }
}
```



Fonte: [Documentação oficial](#)

Layouts básicos

A fase de layout

- Durante a fase de layout, **árvore de UI** é percorrida em um único passo
 - A cada nó é perguntado sobre o seu próprio tamanho
 - Cada filho é medido em seguida ao mesmo tempo que as restrições de tamanho são passadas para os filhos
 - Por fim, o tamanho das folhas é calculado e elas são posicionadas
 - As informações de posicionamento são repassadas de volta
- Resumidamente, os pais são medidos antes dos filhos, porém são dimensionados e posicionados depois deles

Layout

Layouts básicos

A fase de layout

```
@Composable
```

```
fun SearchResult() {
```

```
    Row {
```

```
        Image (
```

```
            // ...
```

```
        )
```

```
        Column {
```

```
            Text (
```

```
                // ...
```

```
            )
```

```
            Text (
```

```
                // ...
```

```
            )
```

```
        }
```

```
    }
```

```
}
```

1 Qual o seu tamanho?

10 Tamanho calculado Posicionado

2 Qual o seu tamanho?

3 Tamanho calculado Posicionado

4 Qual o seu tamanho?

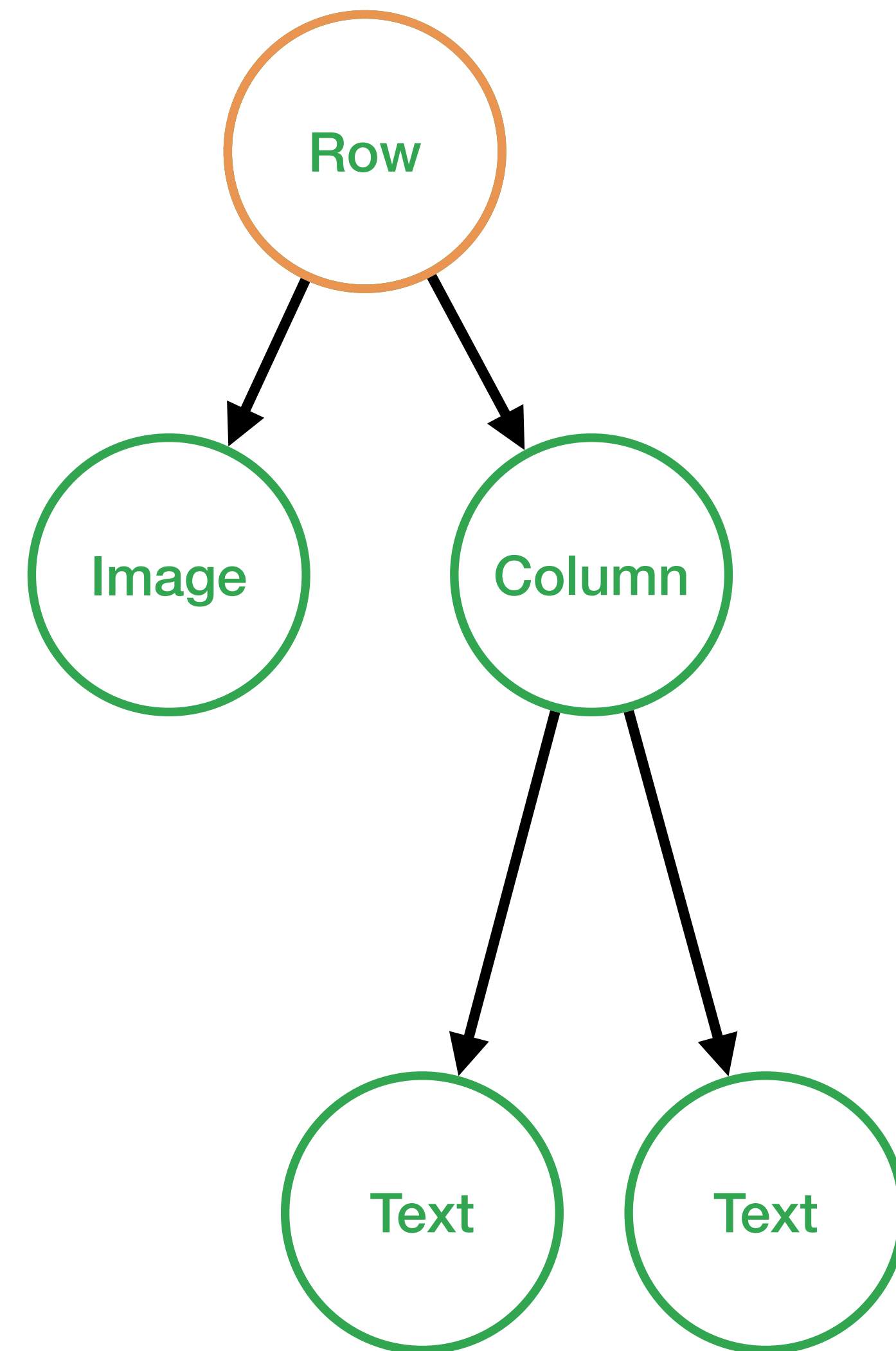
9 Tamanho calculado Posicionado

5 Qual o seu tamanho?

6 Tamanho calculado Posicionado

7 Qual o seu tamanho?

8 Tamanho calculado Posicionado



@Preview

Preview

- Para pré-visualizar um composable basta anotá-las com **@Preview**
 - Informa ao Android Studio que o composable deve ser mostrado na aba de design
 - A pré-visualização é atualizada em tempo real

```
@Composable
fun Saudacao(nome: String) {
    Text(text = "Olá, $nome!")
}
```

```
@Preview
@Composable
fun SaudacaoPreview() {
    Saudacao(nome = "Ana")
}
```

Preview

- É possível adicionar parâmetros que customizam a maneira como `@Preview` é renderizando
- A principal vantagem dessa técnica é evitar o uso do emulador e ainda assim ter a acesso a aparência do seu composable

Preview

Plano de fundo

- Por padrão o plano de fundo no preview é transparente
- Para mostrar um plano de fundo precisamos de dois parâmetros

```
@Preview(showBackground = true, backgroundColor = 0xFF00FF00)
@Composable
fun WithGreenBackground() {
    Text("Hello World")
}
```



O valor desse ser um Long no formato ARGB

Preview

Dimensões

- Por padrão as dimensões são ajustas para conter o composable
- É possível configurar as dimensões manualmente (dp)

```
@Preview(widthDp = 50, heightDp = 50)
@Composable
fun SquareComposablePreview() {
    Box(modifier.background(Color.Yellow)) {
        Text("Hello World")
    }
}
```


Preview

Diferentes dispositivos

- A partir do Android Studio Flamingo, é possível informar o dispositivo
 - O autocomplete pode ler ajudar

```
@Preview(device = "id:pixel 4")
@Composable
fun SquareComposablePreview() {
    Box (Modifier.background(Color.Yellow)) {
        Text ("Hello World")
    }
}
```

Preview

Locale

- É possível adicionar mais de uma anotação `@Preview` ao mesmo composable
- É possível testar com diferente `locale`

```
@Preview(locale = "en", showBackground = true, name = "English")
@Preview(locale = "es", showBackground = true, name = "Spanish")
@Preview(locale = "ko", showBackground = true, name = "Korean")
```

```
@Composable
```

```
fun PreviewHomeTopBar() {
```

```
    SocialsTheme {
```

```
        Scaffold(topBar = {
```

```
            SocialsTopBar(
```

```
                title = stringResource(id = R.string.home_page_title),
```

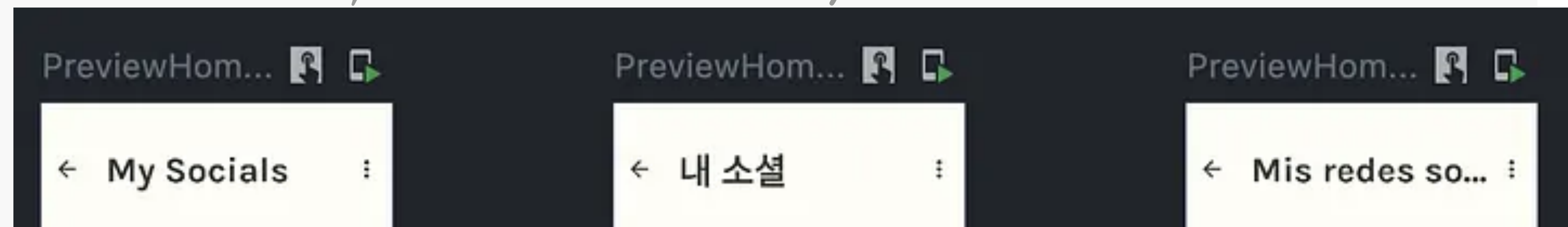
```
                ...
```

```
            )
```

```
        }
```

```
    }
```

```
}
```

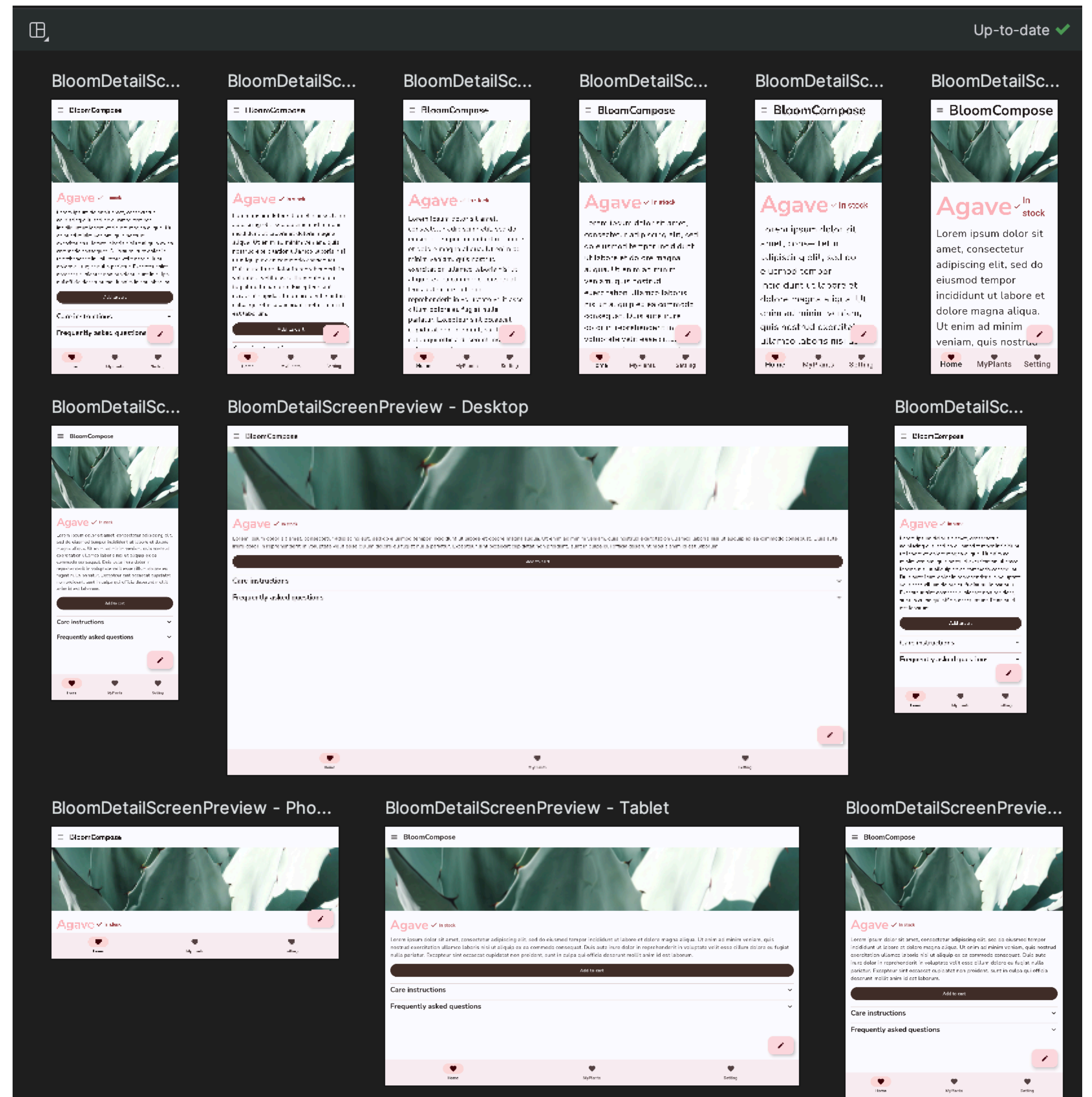


Fonte: <https://medium.com/@kyle.dahlin0/>

Preview

A partir da versão 1.6-alpha01+ é possível verificar a aparência em diversos cenários usando uma única anotação

@PreviewScreenSizes,
@PreviewFontScales,
@PreviewLightDark e
@PreviewDynamicColors



Modifier

Bibliografia

- [Documentação oficial do Kotlin](#)
- [Developer Ecosystem - Kotlin](#)
- [12 Top Kotlin Features to Enhance Android App Development Process](#)
- [Adicionando Extension Functions no Kotlin](#)
- [Understand Arrangement and Alignment in Jetpack Compose](#)

Por hoje é só