

TUTORIAL

Tutorial do Jetpack Compose

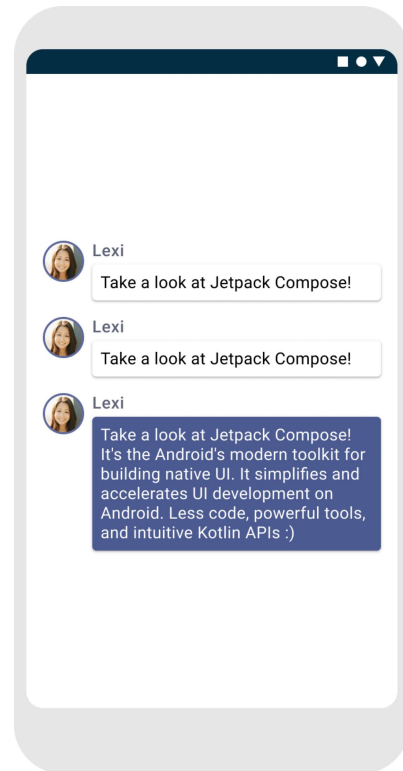
4
lições

[Configurar >](#)

(<https://developer.android.com/develop/ui/compose/setup?hl=pt-br>)

O Jetpack Compose é um toolkit moderno para criação de interface nativa do Android. Ele simplifica e acelera o desenvolvimento de IUs no Android com menos código, ferramentas com a mais alta tecnologia e APIs Kotlin intuitivas.

Neste tutorial, você vai criar um componente de interface simples com funções declarativas. Você não vai editar nenhum layout XML nem usar o Layout Editor. Em vez disso, você chamará funções de composição para definir quais elementos quer usar e o compilador do Compose fará o restante.

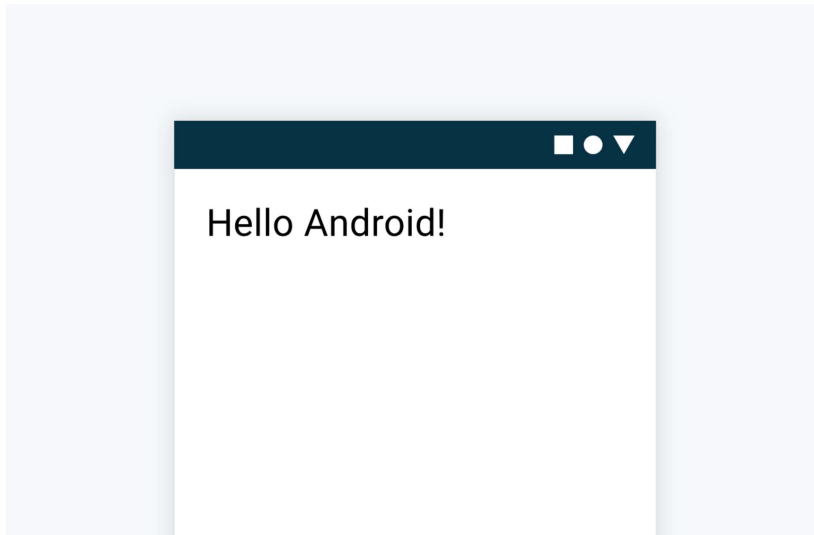


(#top_of_page)

Lição 1: funções de composição

O Jetpack Compose foi criado com base em funções que podem ser compostas. Essas funções permitem que você defina a interface do app de maneira programática, descrevendo as

dependências de dados e de formas dela, em vez de se concentrar no processo de construção da interface (inicializando um elemento, anexando esse elemento a um pai etc.). Para criar uma função que pode ser composta, basta adicionar a anotação `@Composable` ao nome da função.



Adicionar um elemento de texto

Para começar, faça download da versão mais recente do [Android Studio](https://developer.android.com/studio?hl=pt-br) (<https://developer.android.com/studio?hl=pt-br>),

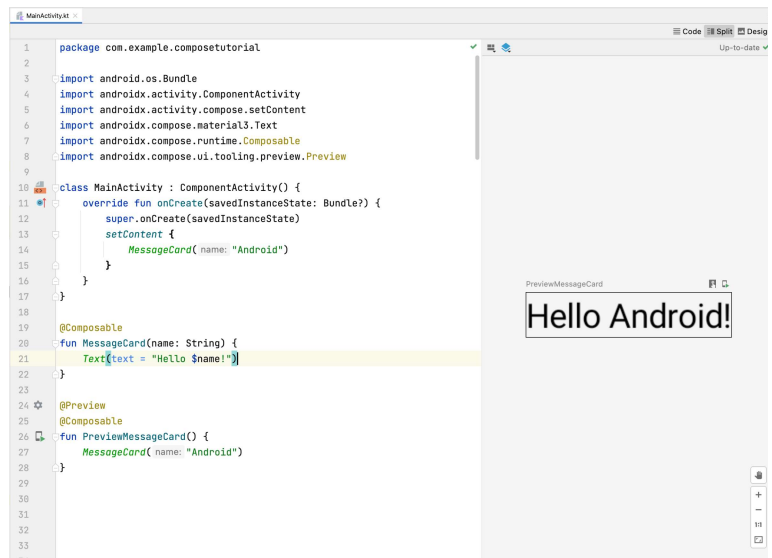
e crie um app selecionando **Novo projeto** e, na **Phone and Tablet**, selecione **Empty Activity**. Nomeie o app como **ComposeTutorial** e clique em **Finish**. O modelo padrão já contém alguns elementos do Compose, mas neste tutorial você vai criar um novo por etapas.

Primeiro, vamos mostrar a mensagem "Hello world!" adicionando um elemento de texto ao método `onCreate`. Para fazer isso, defina um bloco de conteúdo e chame a função de composição `Text`

([https://developer.android.com/reference/kotlin/androidx/compose/material3/package-summary?hl=pt-br#Text\(kotlin.String,androidx.compose.ui.Modifier,androidx.compose.ui.graphics.Color,androidx.compose.ui.unit.TextUnit,androidx.compose.ui.text.font.FontStyle,androidx.compose.ui.text.font.FontWeight,androidx.compose.ui.text.ext.font.FontFamily,androidx.compose.ui.unit.TextUnit,androidx.compose.ui.text.style.TextDecoration,androidx.compose.ui.text.style.TextAlign,androidx.compose.ui.unit.TextUnit,androidx.compose.ui.text.style.TextOverflow,kotlin.Boolean,kotlin.Int,kotlin.Function1,androidx.compose.ui.text.TextStyle\)\)](https://developer.android.com/reference/kotlin/androidx/compose/material3/package-summary?hl=pt-br#Text(kotlin.String,androidx.compose.ui.Modifier,androidx.compose.ui.graphics.Color,androidx.compose.ui.unit.TextUnit,androidx.compose.ui.text.font.FontStyle,androidx.compose.ui.text.font.FontWeight,androidx.compose.ui.text.ext.font.FontFamily,androidx.compose.ui.unit.TextUnit,androidx.compose.ui.text.style.TextDecoration,androidx.compose.ui.text.style.TextAlign,androidx.compose.ui.unit.TextUnit,androidx.compose.ui.text.style.TextOverflow,kotlin.Boolean,kotlin.Int,kotlin.Function1,androidx.compose.ui.text.TextStyle))))

O bloco `setContent` define o layout da atividade em que as funções de composição são chamadas. Elas só podem ser chamadas usando outras funções desse tipo.

O Jetpack Compose usa um plug-in do compilador Kotlin para transformar essas funções de composição nos elementos de IU do app. Por exemplo, a função `Text` que é definida pela biblioteca de IU do Compose mostra um identificador de texto na tela.



Definir uma função que pode ser composta

Para tornar uma função composta, adicione a anotação `@Composable`. Para testar isso, defina uma função `MessageCard` que recebe um nome e o usa para configurar o elemento de texto.

Visualizar a função no Android Studio

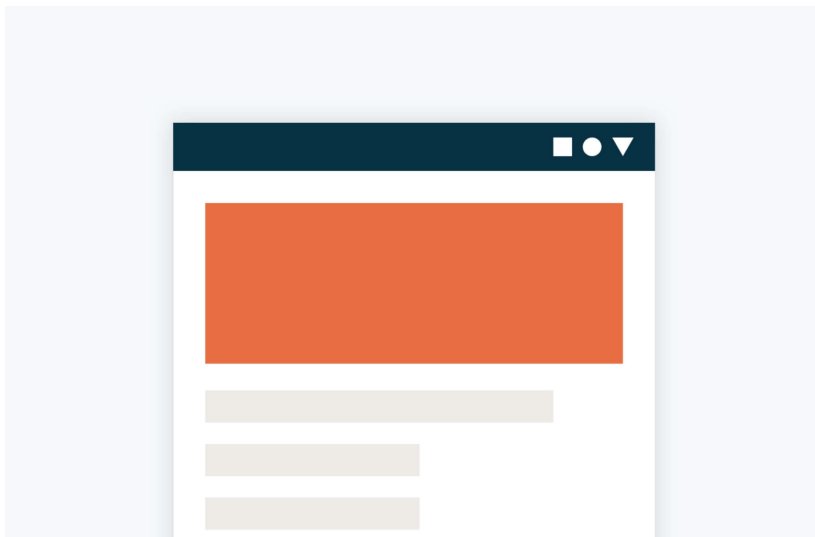
A anotação `@Preview` permite visualizar as funções de composição no Android Studio sem precisar criar e instalar o app em um emulador ou dispositivo Android. A anotação precisa ser usada em uma função de composição que não use parâmetros. Por esse motivo, não é possível visualizar a função `MessageCard` diretamente. Em vez disso, crie uma segunda função nomeada como `PreviewMessageCard`, que chama `MessageCard` com um parâmetro adequado. Adicione a anotação `@Preview` antes da `@Composable`.

Recrie seu projeto. O app em si não muda, porque a nova função `PreviewMessageCard` não é chamada em nenhum lugar, mas o Android Studio adiciona uma janela de prévia que pode ser aberta clicando na visualização (de design/código) dividida. Essa janela mostra uma prévia dos elementos da IU criados por funções de composição marcadas com a anotação `@Preview`. Para atualizar as prévias a qualquer momento, clique no botão "Atualizar" na parte de cima da janela delas.

(#top_of_page)

Lição 2: layouts

Os elementos da IU são hierárquicos, com elementos contidos em outros. No Compose, você cria uma hierarquia de interface chamando funções combináveis usando outras funções desse tipo.



Adicionar vários
textos

OCULTAR PRÉVIA

```
// ...  
import androidx.compose.foundation.layout.Spacer  
import androidx.compose.foundation.layout.height  
import androidx.compose.foundation.layout.padding  
import androidx.compose.foundation.layout.size
```

Até agora, você criou sua primeira função de composição e uma prévia. Para conhecer mais recursos do Jetpack Compose, vamos criar uma tela de mensagens simples com uma lista de mensagens que pode ser aberta com algumas animações.

Comece complementando o elemento de composição da mensagem, mostrando o nome do autor e o conteúdo dela. Primeiro, você precisa mudar o parâmetro de composição para aceitar um objeto `Message` em vez de `String` e adicionar outra função de composição `Text` à `MessageCard`. Não se esqueça de também atualizar a prévia.

Esse código cria dois elementos de texto dentro da visualização do conteúdo. No entanto, como você não deu nenhuma informação sobre como organizar os elementos, eles são mostrados uns sobre os outros, o que deixa o texto ilegível.

Como usar uma coluna

A função `Column`

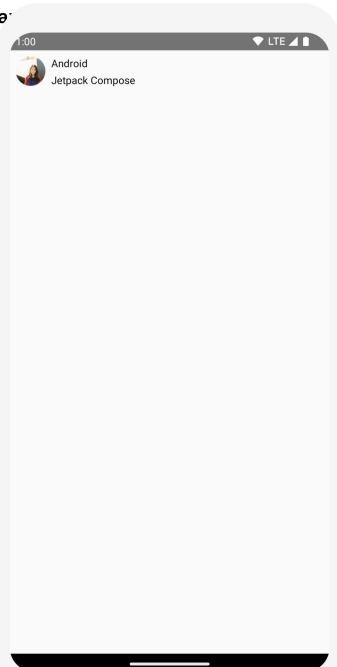
([https://developer.android.com/reference/kotlin/androidx/compose/foundation/layout/package-summary?hl=pt-br#Column\(androidx.compose.ui.Modifier,androidx.compose.foundation.layout.Arrangement.Vertical,androidx.compose.ui.Alignment.Horizontal,kotlin.Function1\)\(androidx.compose.ui.Modifier,androidx.compose.foundation.layout.Arrangement.Vertical,androidx.compose.ui.Alignment.Horizontal,kotlin.Function1\)](https://developer.android.com/reference/kotlin/androidx/compose/foundation/layout/package-summary?hl=pt-br#Column(androidx.compose.ui.Modifier,androidx.compose.foundation.layout.Arrangement.Vertical,androidx.compose.ui.Alignment.Horizontal,kotlin.Function1)(androidx.compose.ui.Modifier,androidx.compose.foundation.layout.Arrangement.Vertical,androidx.compose.ui.Alignment.Horizontal,kotlin.Function1)))

```
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.layout.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.unit.dp

@Composable
fun MessageCard(msg: Message) {
    // Add padding around our message
    Row(modifier = Modifier.padding(
        Image(
            painter = painterResource("Co
            contentDescription = "Co
            modifier = Modifier
                // Set image size to
                .size(40.dp)
                // Clip image to be
                .clip(CircleShape)
        )

    // Add a horizontal space be
    Spacer(modifier = Modifier.w

    Column {
        Text(text = msg.author)
        // Add a vertical space b
        Spacer(modifier = Modifier.height(4.dp))
        Text(text = msg.body)
    }
}
```



permite organizar os elementos verticalmente. Adicione uma `Column` à função `MessageCard`.

Você pode usar a `Row`

([https://developer.android.com/reference/kotlin/androidx/compose/foundation/layout/package-summary?hl=pt-br#Row\(androidx.compose.ui.Modifier,androidx.compose.foundation.layout.Arrangement.Horizontal,androidx.compose.ui.Alignment.Vertical,kotlin.Function1\)](https://developer.android.com/reference/kotlin/androidx/compose/foundation/layout/package-summary?hl=pt-br#Row(androidx.compose.ui.Modifier,androidx.compose.foundation.layout.Arrangement.Horizontal,androidx.compose.ui.Alignment.Vertical,kotlin.Function1)))

(`androidx.compose.ui.Modifier,androidx.compose.foundation.layout.Arrangement.Horizontal,androidx.compose.ui.Alignment.Vertical,kotlin.Function1`)

para organizar os itens

horizontalmente e a `Box`

([https://developer.android.com/reference/kotlin/androidx/compose/foundation/layout/package-summary?hl=pt-br#Box\(androidx.compose.ui.Modifier\)](https://developer.android.com/reference/kotlin/androidx/compose/foundation/layout/package-summary?hl=pt-br#Box(androidx.compose.ui.Modifier)))

para empilhar os elementos.

Adicionar um elemento de imagem

Complemente seu card de mensagens adicionando uma foto do perfil do remetente. Use o `Resource Manager`

(<https://developer.android.com/studio/write/resource-manager?hl=pt-br#import>)

para importar uma imagem da sua biblioteca de fotos ou use

esta

(https://developer.android.com/static/develop/ui/compose/images/compose-tutorial/profile_picture.png?hl=pt-br)

. Adicione uma função de composição `Row` para ter um design bem estruturado e uma

`Image`

([https://developer.android.com/reference/kotlin/androidx/compose/foundation/package-summary?hl=pt-br#Image\(androidx.compose.ui.graphics.painter.Painter,kotlin.String,androidx.compose.ui.Modifier,androidx.compose.ui.Alignment,androidx.compose.ui.layout.ContentScale,kotlin.Float,androidx.compose.ui.graphics.ColorFilter\)](https://developer.android.com/reference/kotlin/androidx/compose/foundation/package-summary?hl=pt-br#Image(androidx.compose.ui.graphics.painter.Painter,kotlin.String,androidx.compose.ui.Modifier,androidx.compose.ui.Alignment,androidx.compose.ui.layout.ContentScale,kotlin.Float,androidx.compose.ui.graphics.ColorFilter))) dentro dela:

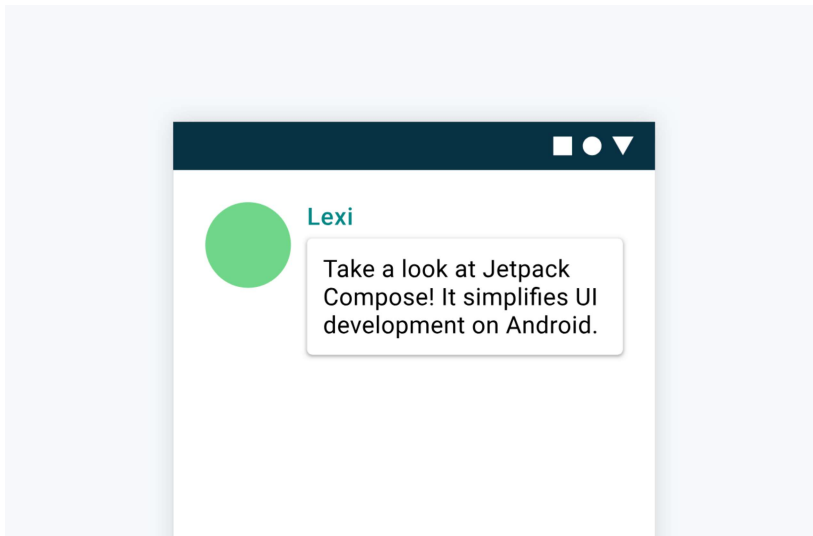
Configurar o layout

O layout da mensagem tem a estrutura certa, mas os elementos não estão bem espaçados e a imagem é muito grande. Para decorar ou configurar uma função de composição, o Compose usa **modificadores**. Eles permitem alterar o tamanho, o layout e a aparência dos elementos que podem ser compostos ou adicionar interações de alto nível, como tornar um elemento clicável. Eles podem ser encadeados para criar funções combináveis mais detalhadas. Você vai usar alguns deles para melhorar o layout.

(#top_of_page)

Lição 3: Material Design

O Compose foi criado para oferecer suporte aos princípios do Material Design. Muitos dos elementos de IU dele implementam o Material Design por padrão. Nesta lição, você vai estilizar seu app com widgets do Material Design.



Usar o Material Design

O design da mensagem agora tem um layout, mas ainda não está bom.

O Jetpack Compose oferece uma implementação do Material Design 3 e elementos de IU prontos para uso. Você vai melhorar a aparência da nossa função de composição `MessageCard` usando os estilos do Material Design.

Para começar, una a função `MessageCard` ao tema do Material Design criado no projeto, `ComposeTutorialTheme`, bem como a um `Surface`. Faça isso nas funções `@Preview` e

OCULTAR PRÉVIA

```
// ...
import android.content.res.Configuration

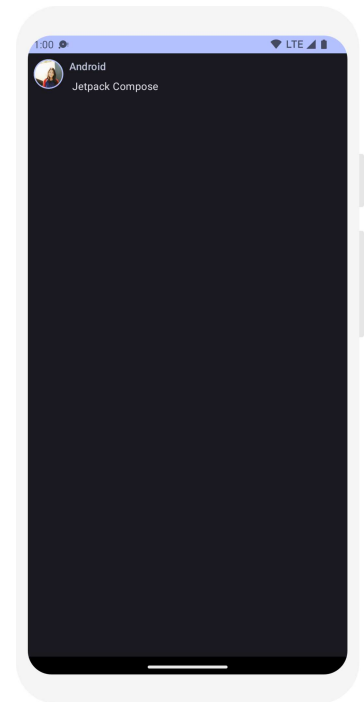
@Preview(name = "Light Mode")
@Preview(
    uiMode = Configuration.UI_MODE_NIGHT_YES,
    showBackground = true,
    name = "Dark Mode"
)
@Composable
fun PreviewMessageCard() {
    ComposeTutorialTheme {
        Surface {
            MessageCard(
                msg = Message("Lexi", "Hey, take a look at Jetpack Compose,")
            )
        }
    }
}
```

`setContent`. Isso vai permitir que suas funções de composição herdem os estilos definidos no tema do app, garantindo a consistência dentro dele.

O Material Design foi criado com base em três pilares: `Color` (cor), `Typography` (tipografia) e `Shape` (forma). Eles vão ser adicionados um por um.

★ **Observação:** o modelo Empty Compose Activity gera um tema padrão para o projeto, que permite personalizar o `MaterialTheme` ([https://developer.android.com/reference/kotlin/androidx/compose/material3/package-summary?hl=pt-br#MaterialTheme\(androidx.compose.material3.ColorScheme,androidx.compose.material3.Shapes,androidx.compose.material3.Typography,kotlin.Function0\)](https://developer.android.com/reference/kotlin/androidx/compose/material3/package-summary?hl=pt-br#MaterialTheme(androidx.compose.material3.ColorScheme,androidx.compose.material3.Shapes,androidx.compose.material3.Typography,kotlin.Function0)))

. Se você deu ao seu projeto um nome diferente de **ComposeTutorial**, pode encontrar o tema personalizado no arquivo `Theme.kt` do subpacote `ui.theme`.



Cor

Use `MaterialTheme.colorScheme` para definir o estilo com as cores do tema envolvido. Você pode usar esses valores do tema em qualquer lugar em que uma cor seja necessária. Este exemplo usa cores de temas dinâmicas (definidas pelas preferências do dispositivo). Você pode definir `dynamicColor` como `false` no arquivo `MaterialTheme.kt` para mudar isso.

Defina o estilo do título e adicione uma borda à imagem.

Tipografia

Os estilos de tipografia do Material Design ficam disponíveis no `MaterialTheme`, eles só precisam ser adicionados à função de composição `Text`.

Forma

Com o recurso `Shape`, você pode adicionar os toques finais.

Primeiro, envolva o texto do corpo da mensagem em uma

`Surface`

([https://developer.android.com/reference/kotlin/androidx/compose/material/package-summary?hl=pt-br#Surface\(androidx.compose.ui.Modifier,androidx.compose.ui.graphics.Shape,androidx.compose.ui.graphics.Color,androidx.compose.ui.graphics.Color,androidx.compose.foundation.BorderStroke,androidx.compose.ui.unit.Dp,kotlin.Function0\)\)](https://developer.android.com/reference/kotlin/androidx/compose/material/package-summary?hl=pt-br#Surface(androidx.compose.ui.Modifier,androidx.compose.ui.graphics.Shape,androidx.compose.ui.graphics.Color,androidx.compose.ui.graphics.Color,androidx.compose.foundation.BorderStroke,androidx.compose.ui.unit.Dp,kotlin.Function0))))

que pode ser composta. Isso permite personalizar a forma e a elevação do corpo da mensagem. Um padding também é adicionado para melhorar o layout.

Ativar tema escuro

O `tema escuro`

(<https://developer.android.com/guide/topics/ui/look-and-feel/darktheme?hl=pt-br>)

(ou modo noturno) pode ser ativado para evitar uma tela clara, especialmente à noite, ou

apenas para economizar bateria do dispositivo. Graças ao suporte ao Material Design, o Jetpack Compose pode processar o tema escuro por padrão. Ao usar as cores do Material Design, o texto e os planos de fundo são adaptados automaticamente ao plano de fundo escuro.

É possível criar diversas prévias no seu arquivo como funções separadas ou adicionar diversas anotações à mesma função.

Adicione uma nova anotação de prévia e ative o modo noturno.

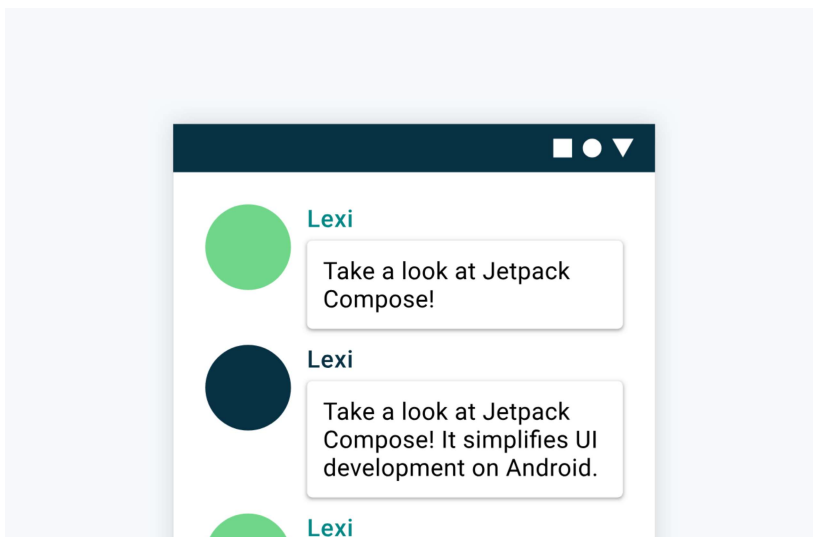
As opções de cores para temas claros e escuros são definidas no arquivo `Theme.kt` gerado pelo ambiente de desenvolvimento integrado.

Até agora, você criou um elemento de IU de mensagem que mostra uma imagem e dois textos com estilos diferentes. Ele fica bem tanto em temas claros quanto escuros.

(#top_of_page)

Lição 4: listas e animações

Listas e animações estão por toda parte nos apps. Nesta lição, você vai aprender como o Compose facilita a criação de listas e torna divertido o acréscimo de animações.



Criar uma lista de mensagens

Um chat com apenas uma mensagem parece um pouco solitário, então vamos mudar a conversa para que tenha mais de uma mensagem. Você vai precisar criar uma função `Conversation` que mostrará várias mensagens. Para este caso de uso, use a `LazyColumn`

OCULTAR PRÉVIA

```
// ...
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items

@Composable
fun Conversation(messages: List<Message>) {
    LazyColumn {
        items(messages) { message ->
            MessageCard(message)
        }
    }
}

@Preview
@Composable
```

([https://developer.android.com/reference/kotlin/androidx/compose/foundation/lazy/package-summary?hl=pt-br#LazyColumn\(androidx.compose.ui.Modifier,androidx.compose.foundation.lazy.LazyListState,androidx.compose.foundation.layout.PaddingValues,kotlin.Boolean,androidx.compose.foundation.layout.Arrangement.Vertical,androidx.compose.ui.Alignment.Horizontal,androidx.compose.foundation.gestures.FlingBehavior,kotlin.Boolean,kotlin.Function1\)](https://developer.android.com/reference/kotlin/androidx/compose/foundation/lazy/package-summary?hl=pt-br#LazyColumn(androidx.compose.ui.Modifier,androidx.compose.foundation.lazy.LazyListState,androidx.compose.foundation.layout.PaddingValues,kotlin.Boolean,androidx.compose.foundation.layout.Arrangement.Vertical,androidx.compose.ui.Alignment.Horizontal,androidx.compose.foundation.gestures.FlingBehavior,kotlin.Boolean,kotlin.Function1)))

e a `LazyRow`

([https://developer.android.com/reference/kotlin/androidx/compose/foundation/lazy/package-summary?hl=pt-br#LazyRow\(androidx.compose.ui.Modifier,androidx.compose.foundation.lazy.LazyListState,androidx.compose.foundation.layout.PaddingValues,kotlin.Boolean,androidx.compose.foundation.layout.Arrangement.Horizontal,androidx.compose.ui.Alignment.Vertical,androidx.compose.foundation.gestures.FlingBehavior,kotlin.Boolean,kotlin.Function1\)](https://developer.android.com/reference/kotlin/androidx/compose/foundation/lazy/package-summary?hl=pt-br#LazyRow(androidx.compose.ui.Modifier,androidx.compose.foundation.lazy.LazyListState,androidx.compose.foundation.layout.PaddingValues,kotlin.Boolean,androidx.compose.foundation.layout.Arrangement.Horizontal,androidx.compose.ui.Alignment.Vertical,androidx.compose.foundation.gestures.FlingBehavior,kotlin.Boolean,kotlin.Function1)))

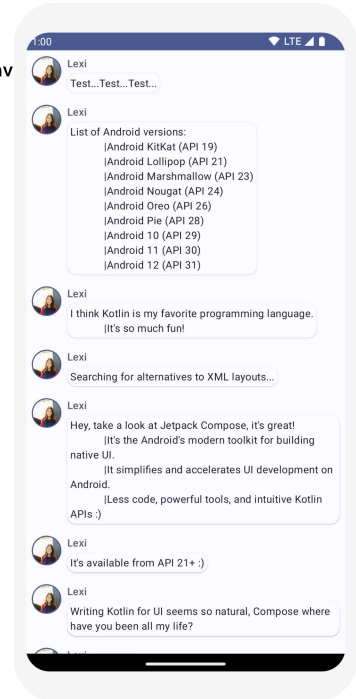
do Compose. Essas funções de composição processam apenas os elementos visíveis na tela. Portanto, elas são muito eficientes para listas longas.

Neste snippet de código, é possível ver que a `LazyColumn` tem um elemento `items` filho. Uma `List` é usada como parâmetro, e o lambda recebe um parâmetro que chamamos de `message`, mas poderíamos ter dado qualquer outro nome, que é uma instância da `Message`. Em resumo, essa lambda é chamada para cada item da `List` fornecida. Copie o [conjunto de dados de exemplo](#)

(<https://developer.android.com/static/develop/ui/compose/tutorial/lessons/lesson-4/steps/code/SampleData.kt?hl=pt-br>)

no seu projeto para inicializar a conversa mais rápido.

```
fun PreviewConversation() {  
    ComposeTutorialTheme {  
        Conversation(SampleData.conv  
    }  
}
```



Animar mensagens
ao abrir

A conversa está ficando mais interessante. Chegou a hora de brincar com animações. Você vai adicionar a capacidade de abrir uma mensagem para mostrar uma parte maior dela, animando o tamanho do conteúdo e a cor do plano de fundo. Para armazenar esse estado da IU local, precisamos conferir se uma mensagem foi aberta ou não. Para monitorar essa mudança de estado, é preciso usar as funções `remember` e `mutableStateOf`.

As funções de composição podem armazenar o estado local na memória usando `remember` e monitorar as mudanças no valor transmitido para `mutableStateOf`. As funções de composição (e os filhos delas) que usam esse estado serão redesenhadas automaticamente quando o valor for atualizado. Isso é chamado de recomposição (<https://developer.android.com/developp/ui/compose/mental-model?hl=pt-br#recomposition>)

Com o uso das APIs de estado do Compose, como `remember` e `mutableStateOf`, qualquer mudança no estado atualiza automaticamente a IU.

★ **Observação:** é necessário adicionar as seguintes importações para usar corretamente o Kotlin sintaxe de propriedade delegada (a palavra-chave `by`). `Alt + Enter` ou `Option + Enter` os adiciona para você.

```
import
androidx.compose.runtime.getValu
e import
androidx.compose.runtime.setValu
e
```

Agora, você pode mudar o plano de fundo do conteúdo da mensagem com base em

`isExpanded` ao clicar em uma mensagem. Você vai usar o modificador `clickable` para processar eventos de clique na função de composição. Em vez de apenas alternar a cor do plano de fundo da `Surface`, você vai criar uma animação com essa cor modificando gradativamente o valor dela de `MaterialTheme.colorScheme.surface` para `MaterialTheme.colorScheme.primary` e vice-versa. Para isso, você vai usar a função `animateColorAsState`. Por fim, você vai usar o modificador `animateContentSize` para animar o tamanho do contêiner da mensagem de modo suave:

Próximas etapas

Parabéns! Você concluiu o tutorial do Compose! Você criou uma tela de chat simples que mostra com eficiência uma lista de mensagens que podem ser abertas e animadas com uma imagem e textos. A tela foi projetada usando os princípios do Material Design com um tema escuro e visualizações, tudo em **menos de cem linhas de código**.

Veja o que você aprendeu até agora:

- Como definir funções combináveis
- Como adicionar elementos diferentes à função que pode ser composta
- Como estruturar o componente de IU usando elementos de layout que podem ser compostos
- Como estender elementos que podem ser compostos usando modificadores
- Como criar uma lista eficiente
- Como acompanhar e modificar o estado
- Como adicionar interação do usuário a um elemento combinável
- Como animar mensagens ao expandi-las

Se você quiser se aprofundar em algumas dessas etapas, conheça os recursos abaixo.

Próximas etapas

CONFIGURAÇÃO

Comece a configurar

(<https://developer.android.com/develop/ui/compose/setup?hl=pt-br>)

Agora que você concluiu o tutorial, pode começar a criar com o Compose.

[Criar um novo app...](#)

[Adicionar a um app já existente...](#)

PROGRAMA DE APRENDIZADO

Continue aprendendo

(<https://developer.android.com/courses/pathways/compose?hl=pt-br>)

Confira nosso caminho de codelabs e vídeos selecionados que ajudarão você a aprender como usar e dominar o Jetpack Compose.

[Iniciar o curso...](#)

