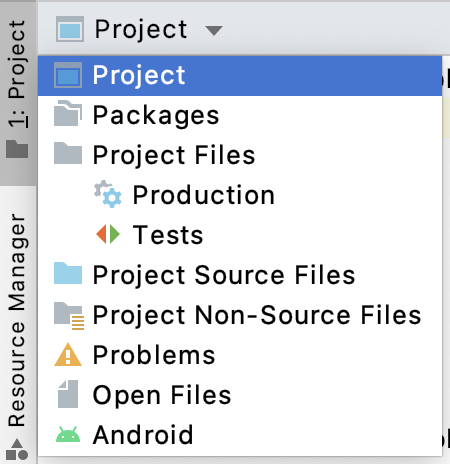
[3. Ícones na tela de início](https://developer.android.com/codelabs/basic-android-kotlin-training-change-app-icon?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-2&hl=pt-br" \l "2)

O objetivo é que o ícone na tela de início pareça incrível (claro e elegante), independentemente do modelo do dispositivo ou da densidade da tela. Especificamente, a densidade de pixel da tela se refere a quantos pixels por polegada (ou dpi, pontos por polegada) estão na tela. Em um dispositivo de densidade média (mdpi), a tela tem 160 pontos por polegada, enquanto em um dispositivo de densidade extra-extra-extra-alta (xxxhdpi) ela tem 640 pontos por polegada.

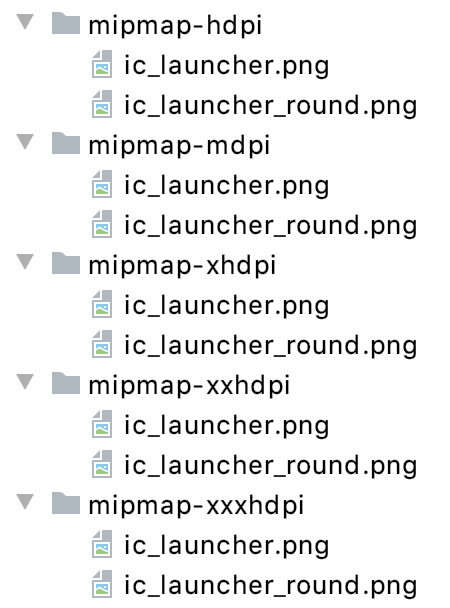
Para levar em consideração dispositivos com várias densidades de tela, é necessário fornecer versões diferentes do ícone do app.

Ver arquivos de ícones na tela de início

1. Para ver a aparência dos ícones, abra o projeto no Android Studio. Se o app foi iniciado de um modelo, você terá ícones na tela de início padrão já fornecidos pelo Android Studio.
2. Na **janela Project**, mude para a visualização **Project**. Os arquivos do projeto serão exibidos de acordo com a estrutura de arquivos do seu computador.



1. Navegue até o diretório de recursos (**app > src > main > res**) e expanda algumas das pastas mipmap. Coloque os recursos do ícone na tela de início do seu app Android nessas pastas mipmap.



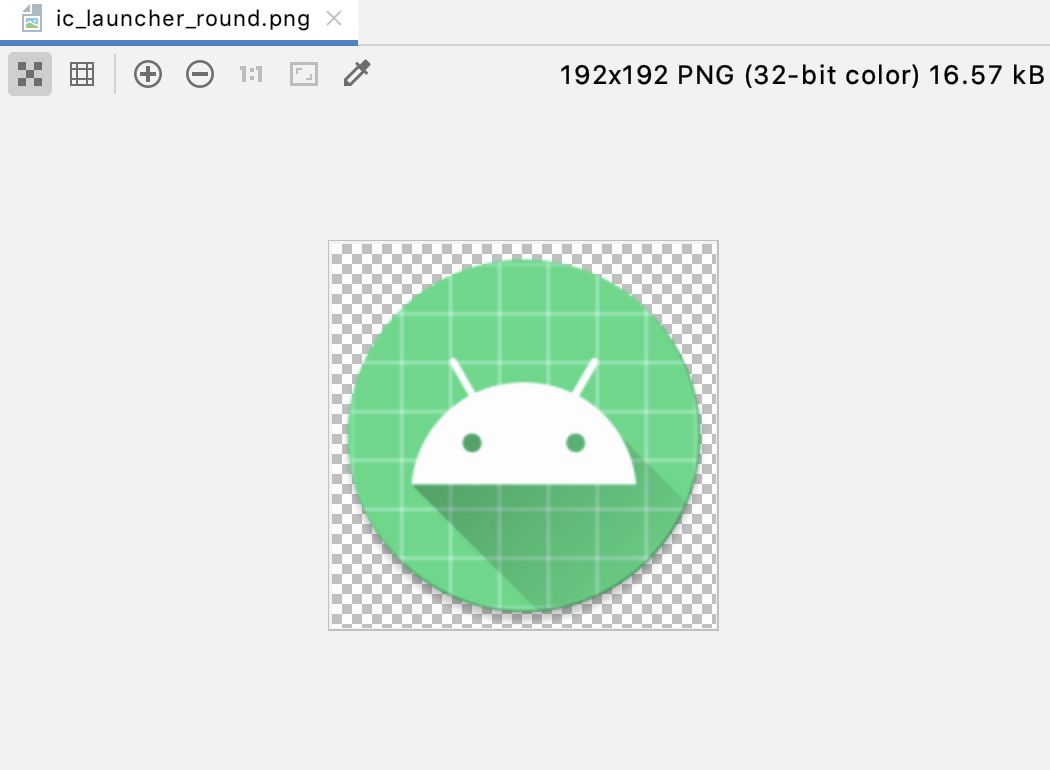
mdpi, hdpi, xhdpi etc. são qualificadores de densidade que podem ser anexados ao nome de um diretório de recursos (como mipmap) para indicar que são recursos para dispositivos com determinada densidade da tela. Veja uma lista de [qualificadores de densidade](https://developer.android.com/training/multiscreen/screendensities?hl=pt-br#TaskProvideAltBmp) no Android:

* mdpi: recursos para telas de média densidade (aproximadamente 160 dpi)
* hdpi: recursos para telas de alta densidade (aproximadamente 240 dpi)
* xhdpi: recursos para telas de densidade extra-alta (aproximadamente 320 dpi)
* xxhdpi: recursos para telas de densidade extra-extra-alta (aproximadamente 480 dpi)
* xxxhdpi: recursos para telas de densidade extra-extra-extra-alta (aproximadamente 640 dpi)
* nodpi: recursos que não podem ser dimensionados, independentemente da densidade de pixels da tela
* anydpi: recursos que são dimensionados para qualquer densidade

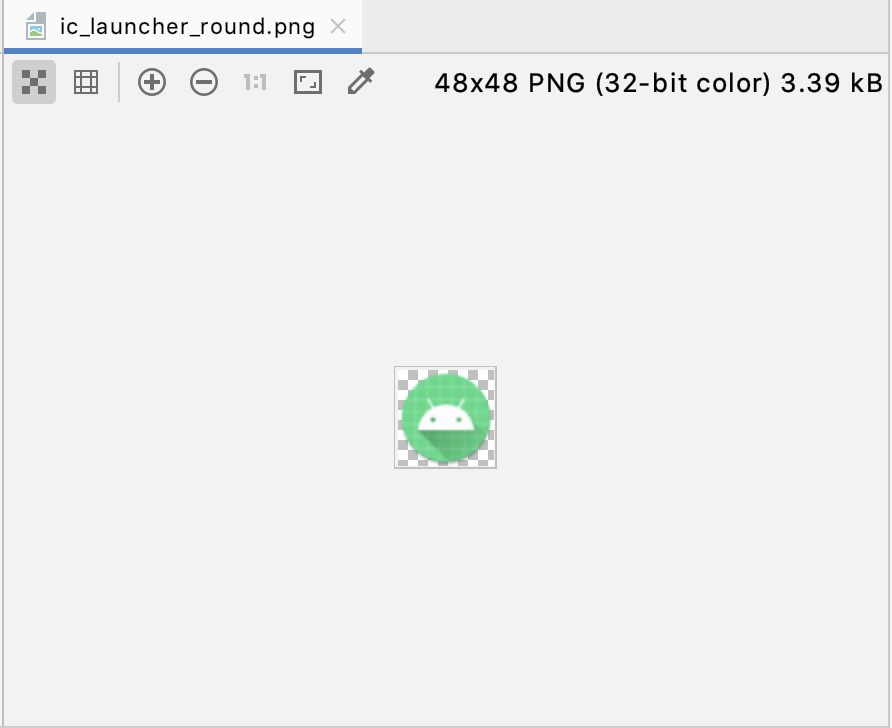
**Observação**: você pode se perguntar por que os recursos do ícone na tela de início estão localizados em diretórios mipmap separados de outros recursos de apps localizados em diretórios drawable. Isso ocorre porque algumas telas de início exibem o ícone do app em um tamanho maior do que o fornecido pelo bucket de densidade padrão do dispositivo. Por exemplo, em um dispositivo hdpi, é possível que a tela de início de um dispositivo use a versão xhdpi do ícone do app.

1. Se você clicar nos arquivos de imagem, verá uma visualização. Os arquivos ic\_launcher.png contêm a versão quadrada do ícone e os arquivos ic\_launcher\_round.png contêm a versão circular. Ambos são fornecidos nos respectivos diretório de recurso.

Por exemplo, **res > mipmap-xxxhdpi > ic\_launcher\_round.png**. Observe também que o tamanho do recurso fica no canto superior direito. O tamanho desta imagem é de 192 px x 192 px.



Por outro lado, **res > mipmap-mdpi > ic\_launcher\_round.png** é assim. O tamanho é de apenas 48 px x 48 px.



Como você pode ver, esses arquivos de imagem de bitmap são compostos por uma grade fixa de pixels. Eles foram criados para determinada resolução de tela. Dessa forma, a qualidade pode diminuir quando você redimensioná-los.

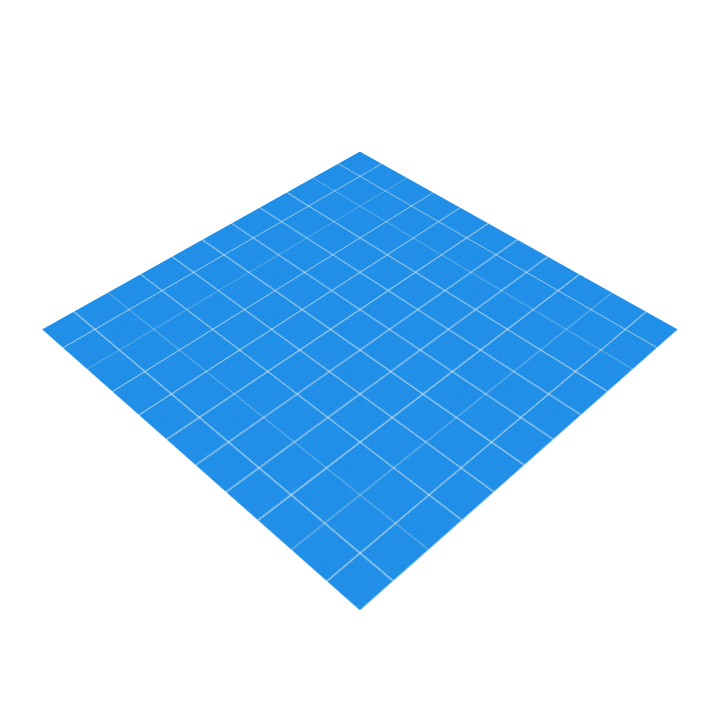
Se você diminuir uma imagem de bitmap, ela provavelmente terá uma boa aparência porque você estará eliminando as informações de pixel. Se você aumentar o tamanho de uma imagem de bitmap de forma significativa, ela poderá parecer desfocada, porque o Android precisará adivinhar e preencher as informações de pixels ausentes.

**Observação**: para evitar que o ícone do app fique desfocado, forneça diferentes imagens de bitmap para cada bucket de densidade (mdpi, hdpi, xhdpi etc.). As densidades de tela do dispositivo não serão exatamente de 160 dpi, 240 dpi, 320 dpi etc. Com base na densidade da tela atual, o Android selecionará o recurso no bucket de maior densidade mais próximo e depois diminuirá o ícone.

## Agora que você tem algum conhecimento sobre ícones na tela de início, aprenderá sobre os ícones adaptativos. [4. Ícones adaptáveis](https://developer.android.com/codelabs/basic-android-kotlin-training-change-app-icon?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-2&hl=pt-br#3)

Camadas de primeiro e segundo plano

Desde a [versão 8.0 do Android](https://developer.android.com/about/versions/oreo?hl=pt-br) (nível 26 da API), a compatibilidade com ícones adaptáveis na tela de início foi adicionada, o que oferece mais flexibilidade e efeitos visuais interessantes para ícones de apps. Para os desenvolvedores, isso significa que o ícone do app é composto de duas camadas: uma de primeiro plano e outra de segundo.

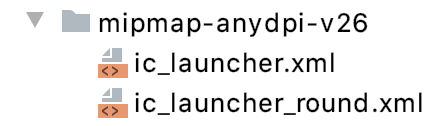


No exemplo acima, o ícone branco do Android está na camada de primeiro plano, enquanto a grade azul e branca está na camada de segundo plano. A camada em primeiro plano será colocada sobre a camada de segundo plano. Dessa forma, uma máscara (circular, neste caso) será aplicada sobre as camadas para produzir um ícone circular do app.

Explorar arquivos de ícones adaptativos

Veja os arquivos padrão de ícones adaptativos fornecidos pelo modelo de projeto no Android Studio.

1. Na **janela Project** do Android Studio, procure e expanda o diretório de recursos **res > mipmap-anydpi-v26**.

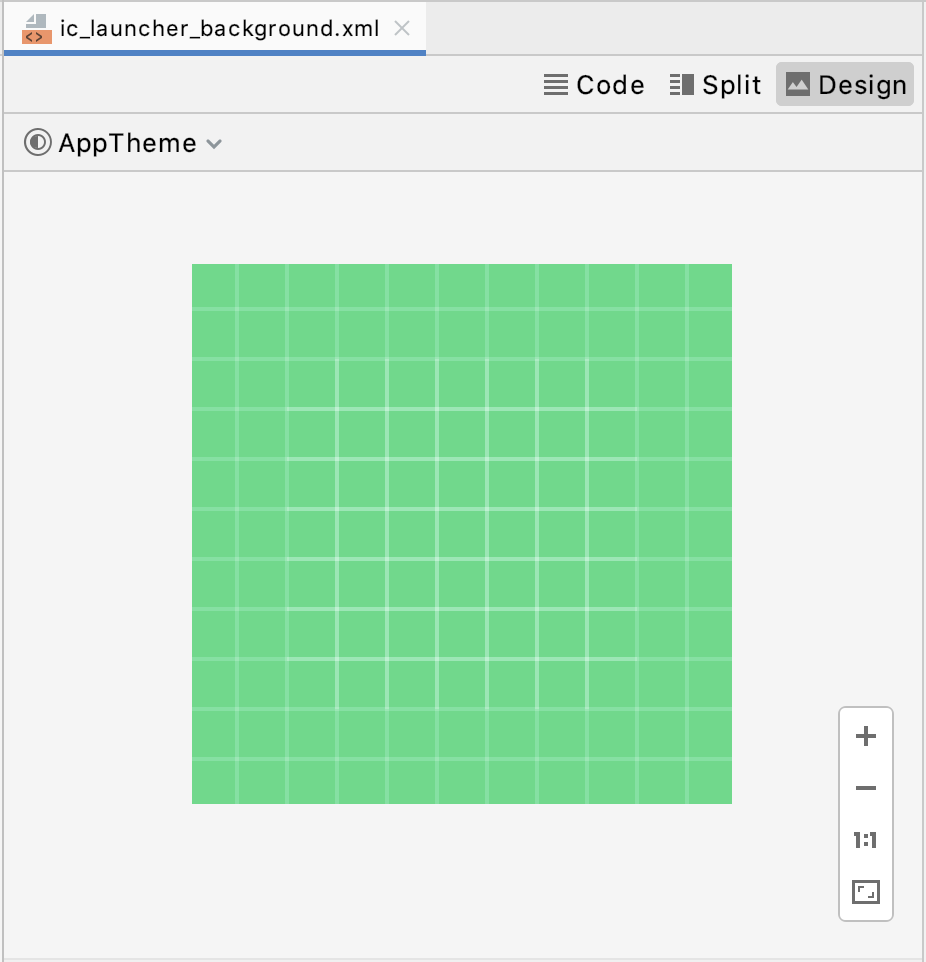
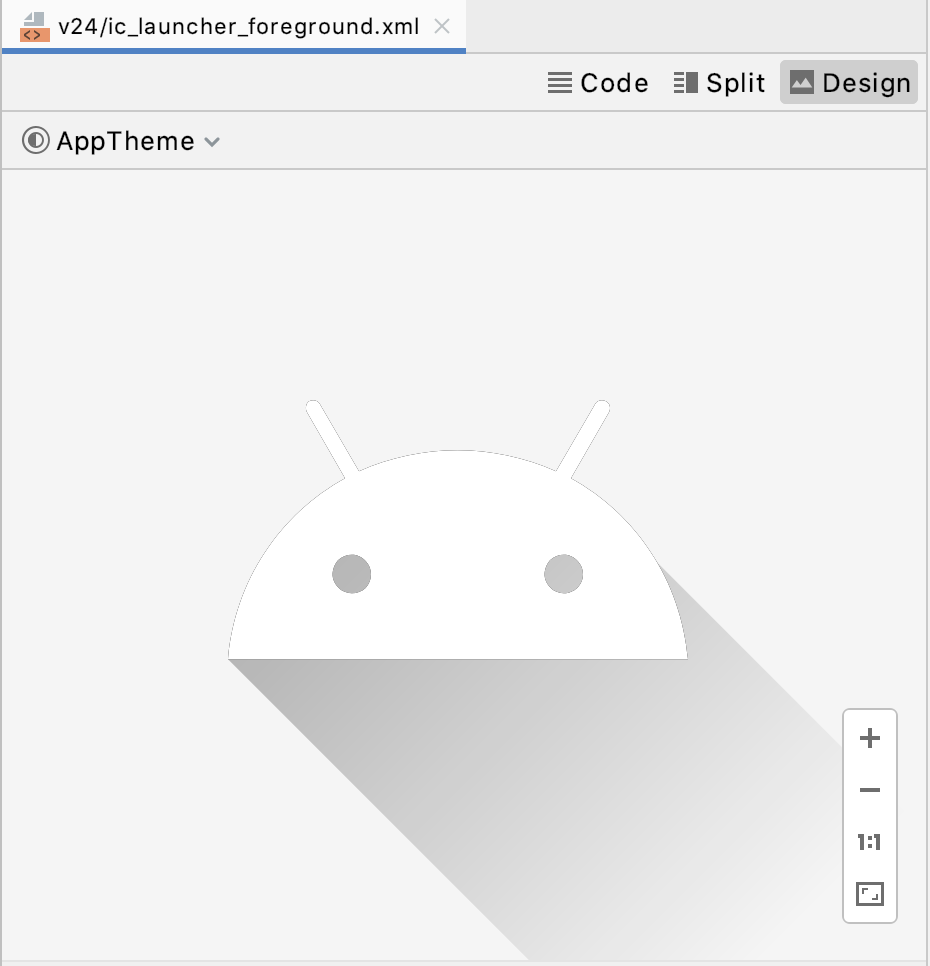


**Observação**: ícones adaptativos foram adicionados no nível 26 da API da plataforma. Os ícones adaptativos precisam ser declarados em um diretório de recursos mipmap com o qualificador de recurso -v26. Isso significa que os recursos desse diretório só serão aplicados aos dispositivos que estiverem executando o nível 26 da API (Android 8.0) ou mais recente. Os arquivos de recursos desse diretório são ignorados em dispositivos com versões mais antigas da plataforma.

1. Abra um dos arquivos XML, por exemplo, ic\_launcher.xml. Você verá isto:

<?xml version="1.0" encoding="utf-8"?>  
<adaptive-icon xmlns:android="http://schemas.android.com/apk/res/android">  
    <background android:drawable="@drawable/ic\_launcher\_background" />  
    <foreground android:drawable="@drawable/ic\_launcher\_foreground" />  
</adaptive-icon>

1. Observe como o elemento <adaptive-icon> é usado para declarar as camadas <background> e <foreground> do ícone do app fornecendo drawables de recurso para cada um.
2. Volte para a visualização **Project** e procure pelo local onde os drawables são declarados: **drawable > ic\_launcher\_background.xml** e **drawable-v24 > ic\_launcher\_foreground.xml**.
3. Alterne para a visualização **Design** para ver uma prévia de cada drawable (de segundo plano à esquerda, de primeiro plano à direita).

1. Ambos os arquivos são drawables vetoriais. Eles não têm um tamanho fixo em pixels. Se você alternar para a visualização de **Code**, poderá ver a declaração XML para o drawable vetorial usando o elemento <vector>.

Embora um drawable vetorial e uma imagem de bitmap descrevam um gráfico, há diferenças importantes entre eles.

Uma imagem de bitmap não tem muitas informações sobre a imagem que ela contém, exceto pelas informações de cor em cada pixel. Por outro lado, um gráfico vetorial sabe como desenhar as formas que definem uma imagem. Essas instruções são compostas por um conjunto de pontos, linhas e curvas, junto com as informações de cor. A vantagem é que um gráfico vetorial pode ser dimensionado para qualquer tamanho e densidade de tela sem perder a qualidade.

Um **[drawable vetorial](https://developer.android.com/guide/topics/graphics/vector-drawable-resources?hl=pt-br" \t "_blank)** é a implementação de gráficos vetoriais do Android, projetada para ser flexível o suficiente em dispositivos móveis. É possível defini-lo em XML com estes [elementos possíveis](https://developer.android.com/reference/kotlin/android/graphics/drawable/VectorDrawable?hl=pt-br). Em vez de fornecer versões de um recurso de bitmap para todos os buckets de densidade, você só precisa definir a imagem uma vez. Dessa forma, você reduz o tamanho do app e facilita a manutenção.

**Observação**: existem [vantagens](https://medium.com/androiddevelopers/understanding-androids-vector-image-format-vectordrawable-ab09e41d5c68) (link em inglês) em usar um drawable vetorial em relação a uma imagem de bitmap. Por exemplo, os ícones podem ser ideais como drawables vetoriais por serem compostos por forma simples, enquanto uma fotografia seria mais difícil de descrever como um conjunto de formas. Nesse caso, seria mais eficiente usar um recurso de bitmap.

Agora, vamos mudar o ícone do app.

[5. Fazer o download de novos recursos](https://developer.android.com/codelabs/basic-android-kotlin-training-change-app-icon?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-2&hl=pt-br#4)

Em seguida, faça o download desses dois novos recursos que permitirão a criação de um ícone adaptativo para um app de cálculo de gorjetas. Você não precisa se preocupar em entender cada detalhe dos arquivos de drawables vetoriais. O conteúdo pode ser gerado automaticamente para você com base nas ferramentas de design.

1. Faça o download do [ic\_launcher\_background.xml](https://raw.githubusercontent.com/google-developer-training/android-basics-kotlin-tip-calculator-app-solution/main/app/src/main/res/drawable-anydpi-v26/ic_launcher_background.xml) (https://developer.android.com/codelabs/basic-android-kotlin-training-change-app-icon?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-2&hl=pt-br#4:~:text=ic\_launcher\_background.xml), o drawable vetorial da camada de primeiro plano. Se o navegador mostrar o arquivo em vez de fazer o download, selecione **Arquivo > Salvar página como...** para salvá-lo no computador.
2. Faça o download do [ic\_launcher\_foreground.xml](https://raw.githubusercontent.com/google-developer-training/android-basics-kotlin-tip-calculator-app-solution/main/app/src/main/res/drawable-anydpi-v26/ic_launcher_foreground.xml) (https://developer.android.com/codelabs/basic-android-kotlin-training-change-app-icon?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-2&hl=pt-br#4:~:text=ic\_launcher\_foreground.xml), o drawable vetorial da camada de segundo plano.

Há alguns requisitos nesses recursos da camada de primeiro e segundo plano. Por exemplo, os dois precisam ter 108 dp x 108 dp. Veja mais detalhes sobre os [requisitos aqui](https://developer.android.com/reference/kotlin/android/graphics/drawable/AdaptiveIconDrawable?hl=pt-br) ou consulte as [orientações de design sobre ícones do Android](https://material.io/design/platform-guidance/android-icons.html) (link em inglês) no site do Material Design.

Como as bordas do ícone podem ser cortadas, dependendo da forma da máscara do fabricante do dispositivo, é importante colocar as principais informações do ícone em uma "[zona segura](https://medium.com/google-design/designing-adaptive-icons-515af294c783)" (link em inglês), que é um círculo com 66 dp de diâmetro no centro da camada. O conteúdo fora dessa zona segura não deve ser essencial. Por exemplo, uma cor de fundo, que pode ser cortada.

[6. Alterar o ícone do app](https://developer.android.com/codelabs/basic-android-kotlin-training-change-app-icon?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-2&hl=pt-br#5)

Volte ao Android Studio para usar os novos recursos.

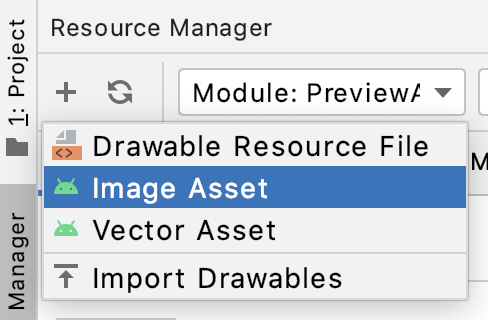
1. Primeiro, exclua os recursos desenháveis antigos que têm o ícone do Android e o plano de fundo de grade verde. Na **visualização Project**, clique com o botão direito do mouse no arquivo e escolha **Delete**.

Exclua:

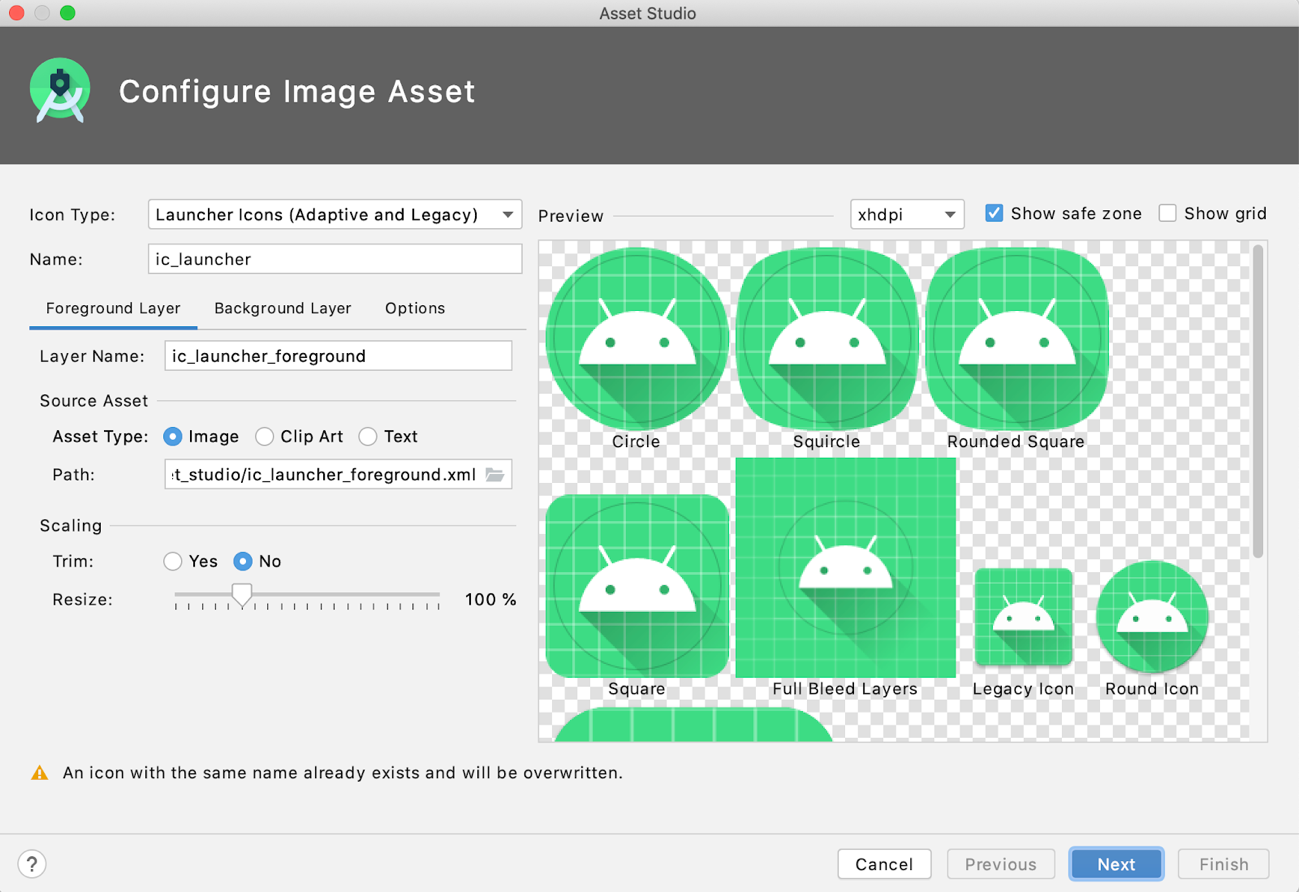
drawable/ic\_launcher\_background.xml  
drawable-v24/ic\_launcher\_foreground.xml

Desmarque a caixa **Safe delete (with usage search)** e clique em **OK**.

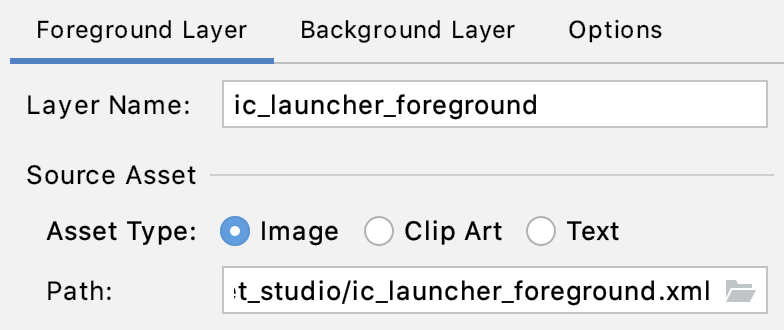
1. Crie um novo **Image Asset**. Clique com o botão direito do mouse no diretório **res** e escolha **New > Image Asset**. Também é possível clicar na guia **Resource Manager**, depois no ícone **+** e selecionar **Image Asset**.



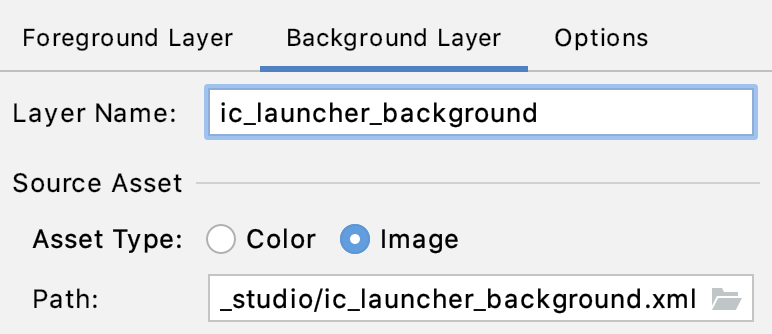
1. A ferramenta **Image Asset Studio** do Android Studio será aberta.
2. Use as configurações padrão:
3. Icon Type: Launcher Icons (Adaptive and Legacy)
4. Name: ic\_launcher



1. Com a **guia Foreground Layer** selecionada, vá até a subseção **Source Asset**. No campo **Path**, clique no ícone de pasta.
2. Será exibida uma solicitação para procurar no seu computador e selecionar um arquivo. Encontre o local do novo arquivo ic\_launcher\_foreground.xml que você acabou de transferir por download para o computador. Ele pode estar na pasta de downloads do computador. Depois de encontrá-lo, clique em **Open**.
3. O campo **Path** será atualizado com o local do novo drawable vetorial de primeiro plano. Deixe o campo **Layer Name** como **ic\_launcher\_foreground** e o campo **Asset Type** como **Image**.



1. Em seguida, alterne para a **guia Background Layer** da interface. Não mude as configurações padrão. Clique no ícone de pasta no campo **Path**.
2. Encontre o local do arquivo ic\_launcher\_background.xml que você acabou de transferir por download. Clique em **Open**.

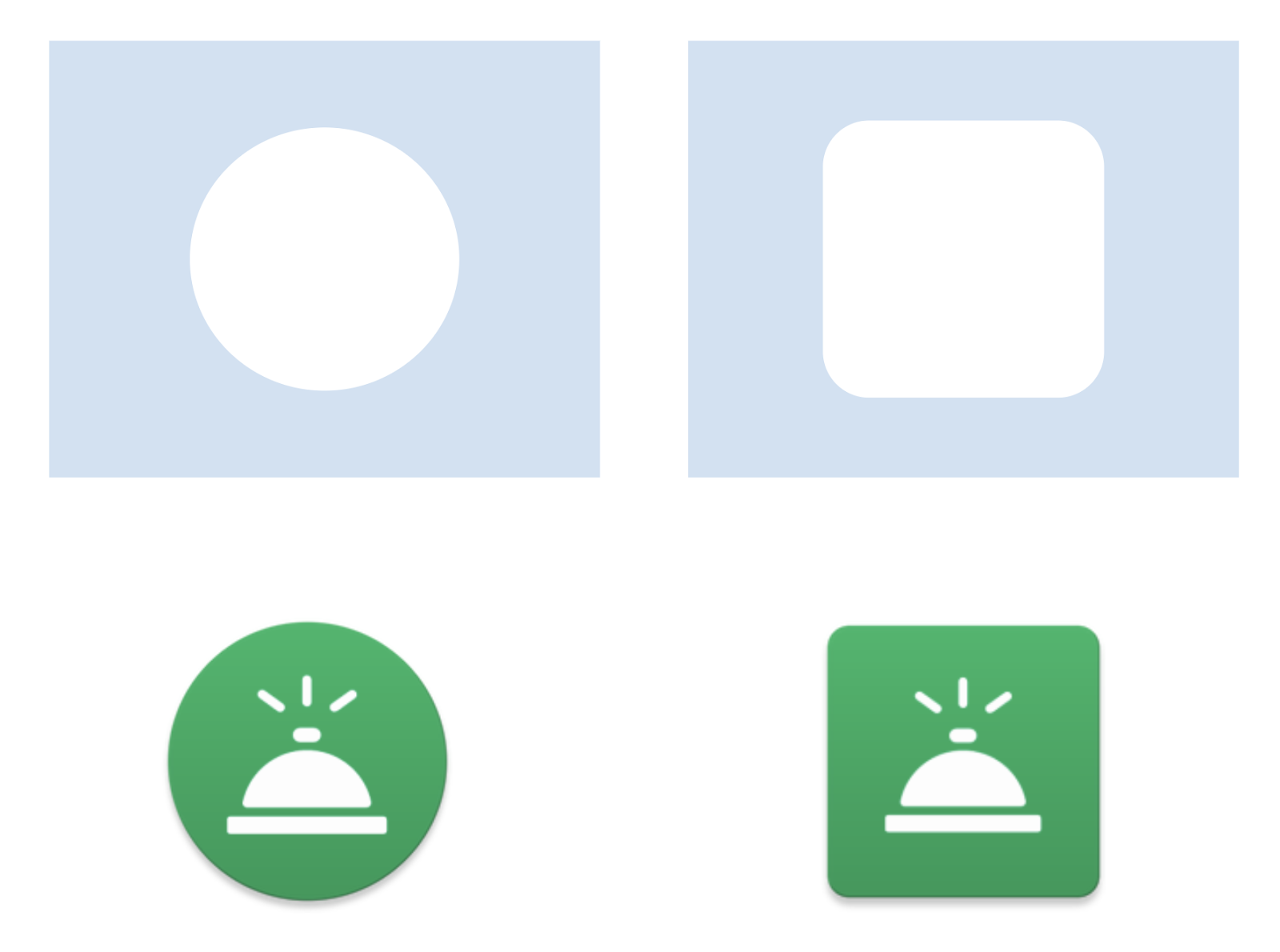


1. A visualização será atualizada conforme você selecionar os novos arquivos de recursos. As novas camadas de primeiro e segundo plano serão exibidas dessa forma.



Ao representar o ícone do app em duas camadas, os fabricantes de dispositivos (chamados de fabricantes de equipamento original, ou OEMs) poderão criar formas diferentes, dependendo do dispositivo Android, como mostrado na visualização acima. O OEM fornece uma máscara que é aplicada a todos os ícones de apps no dispositivo.

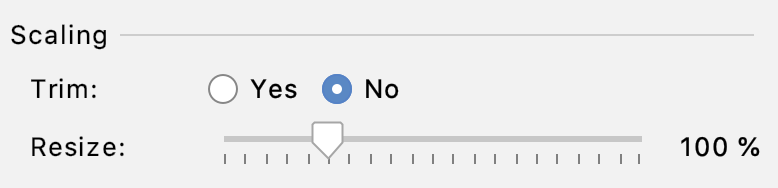
Essa máscara é aplicada sobre as camadas em primeiro e segundo plano do ícone do app. Veja um exemplo de máscara circular e quadrada abaixo.



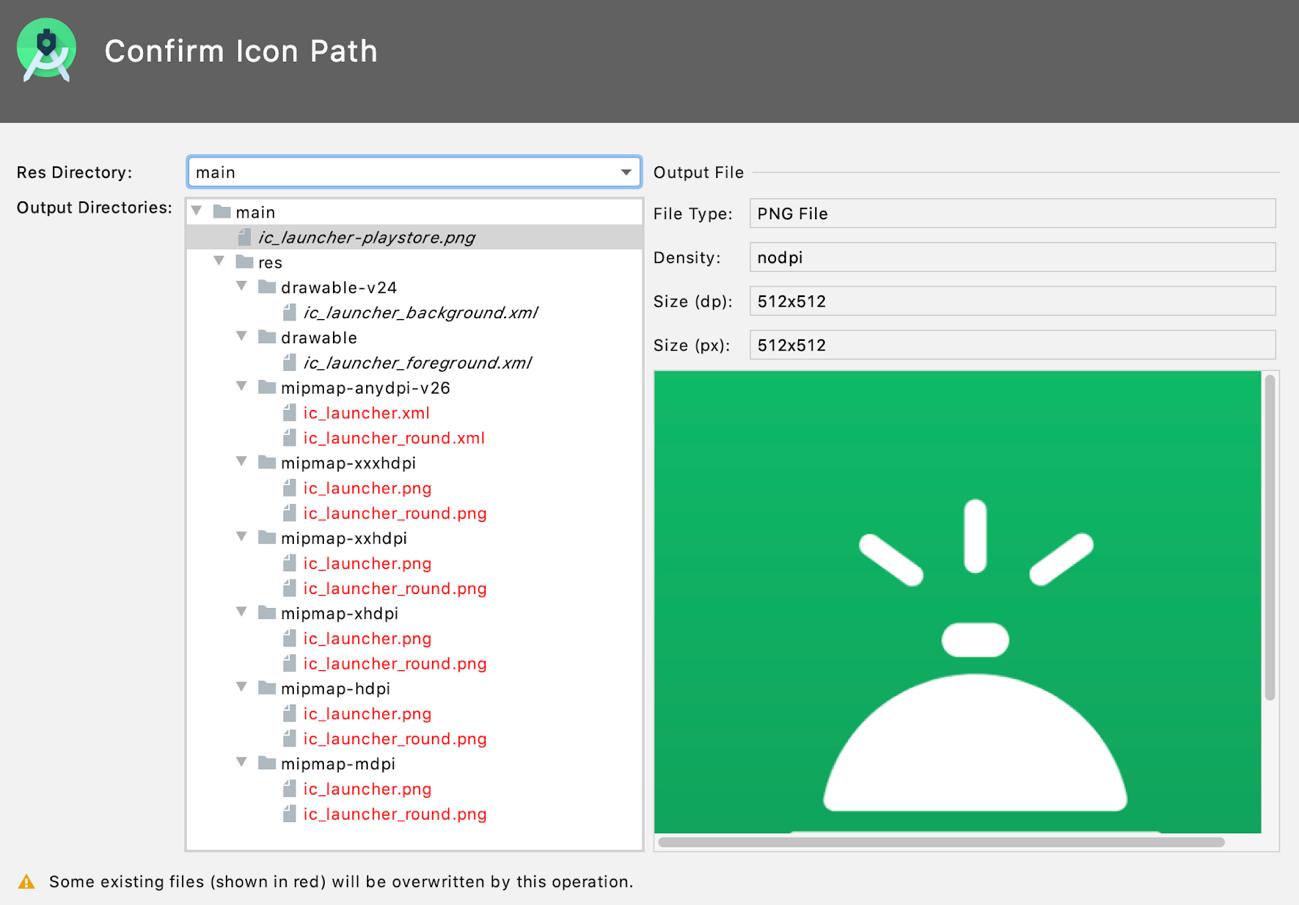
Quando uma máscara circular é aplicada sobre as duas camadas do ícone do app, o resultado é um ícone circular com um plano de fundo de grade azul e o símbolo do Android nela (imagem à esquerda e acima). Como alternativa, você pode aplicar uma máscara quadrada para produzir o ícone do app (imagem acima à direita).

O uso de duas camadas também produz efeitos visuais interessantes, porque ambas as camadas podem se mover de forma independente ou ser dimensionadas. Para ver alguns exemplos divertidos de como os efeitos visuais podem ficar, confira esta [postagem do blog](https://medium.com/google-design/designing-adaptive-icons-515af294c783) (em inglês), na seção "Design Considerations". Como não é possível saber com antecedência qual será o dispositivo do usuário ou qual máscara o OEM aplicará no ícone, é preciso configurar seu ícone adaptativo para que as informações importantes não sejam cortadas.

1. Verifique se o conteúdo principal da camada de primeiro plano (o ícone de sino de atendimento, neste caso) está dentro da zona segura e não foi cortado pelas diferentes formas de máscara. Se um conteúdo importante for cortado ou aparecer muito pequeno, use o controle deslizante **Resize** na seção **Scaling** de cada camada. Nesse caso, não é necessário redimensionar a imagem, então você pode mantê-la em 100%.



1. Clique em **Next**.
2. Esta etapa é **Confirm Icon Path**. É possível clicar nos arquivos individuais para acessar a visualização. Na parte inferior, há um aviso de que alguns arquivos serão substituídos (em vermelho). Isso não é um problema porque esses arquivos antigos eram do ícone do app anterior.



1. Como os diretórios padrão são bons, clique em **Finish**.
2. Verifique se todos os recursos gerados são exibidos corretamente nas pastas mipmap. Exemplos:

Bom trabalho! Agora você fará mais uma mudança.

Mover arquivos de drawables vetoriais para o diretório -v26

Dependendo do SDK mínimo do app, você pode perceber que o recurso em primeiro plano está localizado na pasta drawable-v24, enquanto o recurso em segundo plano está na pasta drawable. O motivo é que o recurso em primeiro plano contém um gradiente, disponível a partir do Android 7.0 também conhecido como a versão 24 da API, portanto, com o qualificador de recurso -v24. O recurso em segundo plano não contém um gradiente, então pode ser colocado na pasta drawable base.

Em vez de colocar os recursos de primeiro e segundo plano em duas pastas drawable separadas, mova os dois arquivos de drawable vetoriais para um diretório de recursos -v26. Como esses recursos são usados apenas para ícones adaptativos, esses dois drawables são necessários somente no nível 26 da API e mais recentes. Essa estrutura de pasta facilita a localização e o gerenciamento dos arquivos de ícone adaptativos.

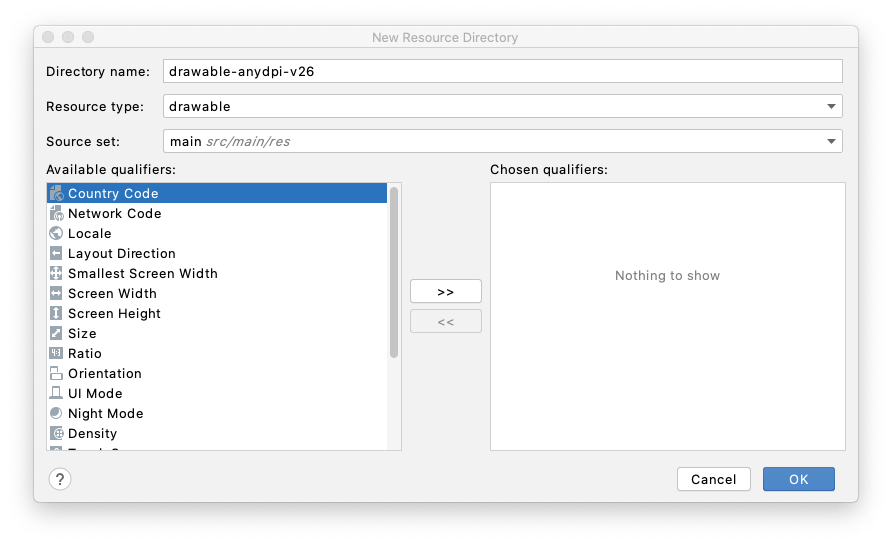
drawable-anydpi-v26  
   ic\_launcher\_background.xml  
   ic\_launcher\_foreground.xml  
mipmap-anydpi-v26  
   ic\_launcher.xml  
   ic\_launcher\_round.xml

1. Primeiro, crie o diretório drawable-anydpi-v26. Clique com o botão direito do mouse na pasta **res**. Selecione **New > Android Resource Directory**.
2. A caixa de diálogo **New Resource Directory** será exibida. Selecione estas opções:

**Directory name**: drawable-anydpi-v26

**Resource type:** drawable (selecione na lista suspensa)

**Source set:** main (não mude o valor padrão)

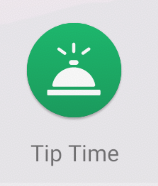


Clique em **OK**. Na visualização **Project**, verifique se o novo diretório de recursos **res > drawable-anydpi-v26** foi criado.

1. Clique com o botão esquerdo do mouse no arquivo ic\_launcher\_foreground.xml e arraste-o da pasta **drawable** para a pasta **drawable-anydpi-v26**. Lembre-se que colocar um recurso em um diretório para "any dpi" indica que ele é um recurso que pode ser dimensionado para qualquer densidade de tela.
2. Clique com o botão esquerdo do mouse em ic\_launcher\_background.xml e arraste-o da pasta **drawable-v24** para a pasta **drawable-anydpi-v26**.
3. Exclua a pasta drawable-v24 se ela estiver vazia. Clique com o botão direito na pasta e selecione **Delete**.
4. Clique em todos os arquivos drawable e mipmap do projeto. Verifique se a visualização desses ícones está correta.

Testar o app

1. Teste se o novo ícone do app é exibido. Execute o app no dispositivo (emulador ou dispositivo físico).
2. Pressione o botão home do dispositivo.
3. Deslize para cima para ver a lista "Todos os apps".
4. Procure o app que você acabou de atualizar. Você verá o novo ícone do app.



Observação: dependendo do modelo do dispositivo, você poderá ver um ícone na tela de início com uma forma diferente. Porém, a camada de primeiro plano ainda será exibida sobre a camada de segundo plano com algum tipo de máscara aplicado a ela.

Muito bem! O novo ícone do app está ótimo.

Ícones adaptáveis e legados na tela de início

Agora que seu ícone adaptativo funciona bem, você pode estar se perguntando por que não pode excluir todas as imagens de bitmap do ícone do app. Esses arquivos ainda são necessários para que o ícone do app seja exibido em alta qualidade em versões mais antigas do Android. Isso é chamado de compatibilidade com versões anteriores.

Em dispositivos que executam o Android 8.0 ou mais recente (versão 26 da API e mais recente):

**Ícones adaptativos** podem ser usados (uma combinação de um drawable vetorial em primeiro plano e um drawable vetorial em segundo plano, com uma máscara OEM aplicada sobre eles). Estes são os arquivos relevantes para seu projeto:

res/drawable-anydpi-v26/ic\_lancher\_background.xml  
res/drawable-anydpi-v26/ic\_launcher\_foreground.xml  
res/mipmap-anydpi-v26/ic\_launcher.xml  
res/mipmap-anydpi-v26/ic\_launcher\_round.xml

Em dispositivos com uma versão anterior ao Android 8.0 (mas acima do nível mínimo da API exigido para nosso app):

**Ícones legados na tela de início** serão usados (as imagens de bitmap nas pastas mipmap de diferentes buckets de densidade). Estes são os arquivos relevantes para seu projeto:

res/mipmap-mdpi/ic\_launcher.png  
res/mipmap-mdpi/ic\_launcher\_round.png  
res/mipmap-hdpi/ic\_launcher.png  
res/mipmap-hdpi/ic\_launcher\_round.png  
res/mipmap-xhdpi/ic\_launcher.png  
res/mipmap-xhdpi/ic\_launcher\_round.png  
res/mipmap-xxdpi/ic\_launcher.png  
res/mipmap-xxdpi/ic\_launcher\_round.png  
res/mipmap-xxxdpi/ic\_launcher.png  
res/mipmap-xxxdpi/ic\_launcher\_round.png

Basicamente, o Android recorrerá às imagens de bitmap em dispositivos mais antigos sem compatibilidade com ícones adaptativos.

Parabéns! Você concluiu todas as etapas para mudar o ícone de um app.

**TEMA**

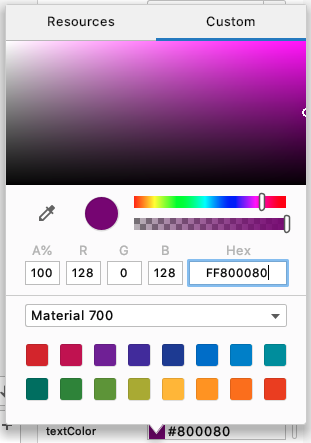
## [2. Design e cor](https://developer.android.com/codelabs/basic-android-kotlin-training-change-app-theme?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-2&hl=pt-br#1)

## Material Design

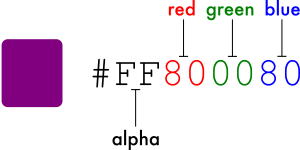
O [Material Design](https://material.io/design/introduction) (link em inglês) é inspirado no mundo físico e nas texturas dele, incluindo a forma como os objetos refletem a luz e projetam sombras. Ele apresenta diretrizes sobre como criar a IU do app de uma forma legível, atraente e consistente. Com o [Material Theming](https://material.io/design/material-theming/overview.html#material-theming) (link em inglês), você pode adaptar o Material Design ao app seguindo as orientações de personalização de cores, tipografia e formas. O Material Design vem com um tema de referência, que pode ser usado sem mudanças. Você pode personalizá-lo o quanto quiser para que o Material Design se adapte ao seu app.

## Um pouco sobre cores

As cores estão por toda parte ao nosso redor, no mundo real e no mundo digital. A primeira coisa que você precisa saber é que nem todos os usuários veem as cores da mesma maneira. Por isso, é importante escolher cores de modo que seu app possa ser usado facilmente pelos usuários. Definir um contraste suficiente é apenas uma etapa para [criar apps mais acessíveis](https://developer.android.com/guide/topics/ui/accessibility?hl=pt-br).



Uma [Cor](https://developer.android.com/reference/kotlin/android/graphics/Color?hl=pt-br) pode ser representada por três números hexadecimais, #00-#FF (0-255), representando os componentes vermelho, verde e azul (RGB) dessa cor. Quanto maior o número, maior a proporção da cor específica.



Uma cor também pode ser definida incluindo um valor alfa #00-#FF, que representa a transparência (#00 = 0% = totalmente transparente, #FF = 100% = totalmente opaco). Quando incluído, o valor alfa é o primeiro de quatro números hexadecimais (ARGB). Se não houver um valor alfa, ele será considerado #FF = 100% opaco.

No entanto, não é necessário gerar os números hexadecimais manualmente. Há ferramentas disponíveis para ajudar você a escolher as cores que gerarão os números.

Alguns exemplos que você pode ter visto no arquivo colors.xml do seu app Android incluem 100% preto (R=#00, G=#00, B=#00) e 100% branco (R=#FF, G=#FF, B=#FF):

<color name="black">#FF000000</color>  
<color name="white">#FFFFFFFF</color>

[3. Temas](https://developer.android.com/codelabs/basic-android-kotlin-training-change-app-theme?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-2&hl=pt-br#2)

Um *estilo* pode especificar atributos para uma View, como cor e tamanho da fonte, cor do segundo plano e muito mais.

O *tema* é uma coleção de estilos aplicados a um app, atividade ou hierarquia de visualização inteiros, não apenas a uma View individual. Quando você aplica um tema a um app, atividade, visualização individual ou em grupo, o tema é aplicado a esse elemento e a todos os filhos deles. Os temas também podem aplicar estilos a elementos que não são de visualização, como a barra de status e o segundo plano da janela.

Criar um projeto Empty Activity

Se você estiver usando seu próprio app, pule esta seção. Se precisar de um app para usar como exemplo, siga estas etapas para criar um app Empty Activity.

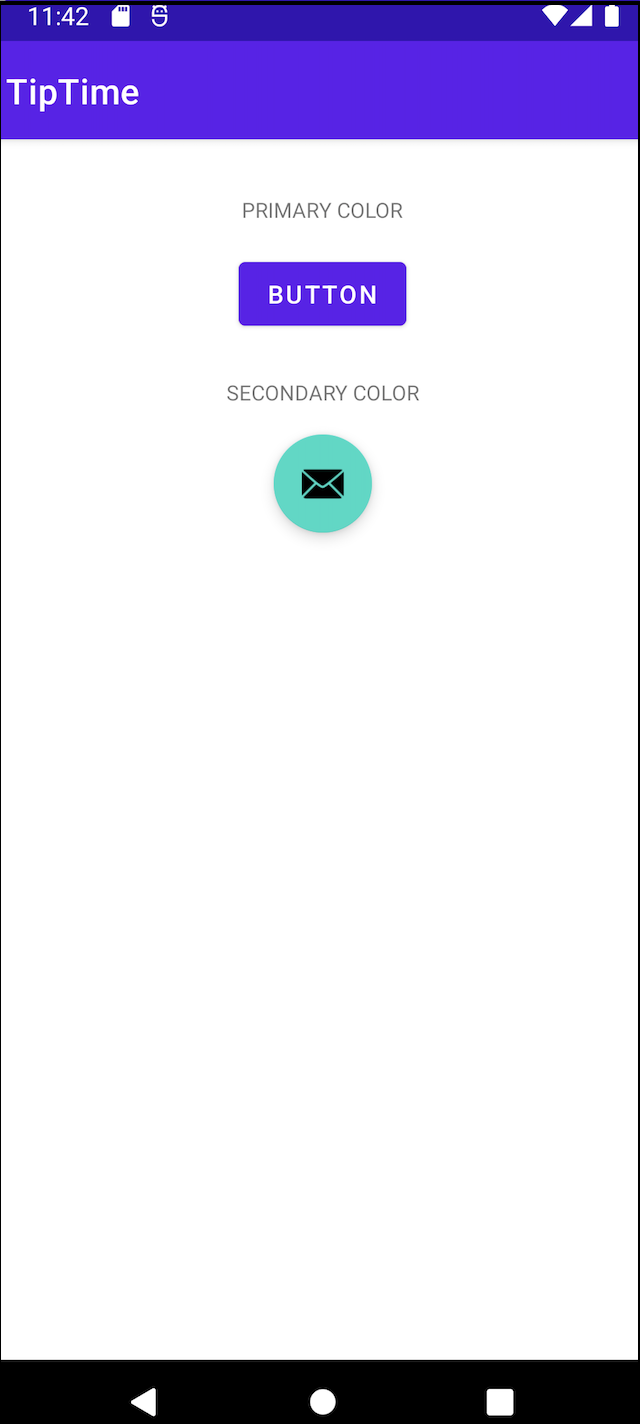
1. Abra o Android Studio.
2. Crie um novo projeto Kotlin usando o modelo **Empty Activity**.
3. Use o nome "TipTime". Como alternativa, você pode manter o nome padrão, "My Application", se não estiver participando de outros codelabs.
4. Selecione uma API de nível 19 no mínimo (KitKat).
5. Depois que o Android Studio concluir a criação do app, abra o arquivo activity\_main.xml (**app > res > layout > activity\_main.xml**).
6. Se necessário, alterne para a visualização **Code**.
7. Substitua todo o texto por este XML:

<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout\_width="match\_parent"  
    android:layout\_height="match\_parent"  
    android:layout\_margin="16dp"  
    android:orientation="vertical"  
    tools:context=".MainActivity">  
  
    <TextView  
        android:layout\_width="wrap\_content"  
        android:layout\_height="48dp"  
        android:layout\_gravity="center\_horizontal"  
        android:gravity="center\_vertical"  
        android:text="@string/primary\_color"  
        android:textAllCaps="true"  
        android:textSize="12sp" />  
  
    <Button  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:layout\_gravity="center\_horizontal"  
        android:text="@string/button" />  
  
    <TextView  
        android:layout\_width="wrap\_content"  
        android:layout\_height="48dp"  
        android:layout\_gravity="center\_horizontal"  
        android:layout\_marginTop="8dp"  
        android:gravity="center\_vertical"  
        android:text="@string/secondary\_color"  
        android:textAllCaps="true"  
        android:textSize="12sp" />  
  
    <com.google.android.material.floatingactionbutton.FloatingActionButton  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:layout\_gravity="center\_horizontal"  
        android:contentDescription="@string/email\_icon"  
        app:srcCompat="@android:drawable/ic\_dialog\_email" />  
  
</LinearLayout>

1. Abra o arquivo strings.xml (**app > res > values > strings.xml**).
2. Substitua todo o texto por este XML:

<resources>  
    <string name="app\_name">TipTime</string>  
    <string name="primary\_color">Primary color</string>  
    <string name="button">Button</string>  
    <string name="secondary\_color">Secondary color</string>  
    <string name="email\_icon">email icon</string>  
</resources>

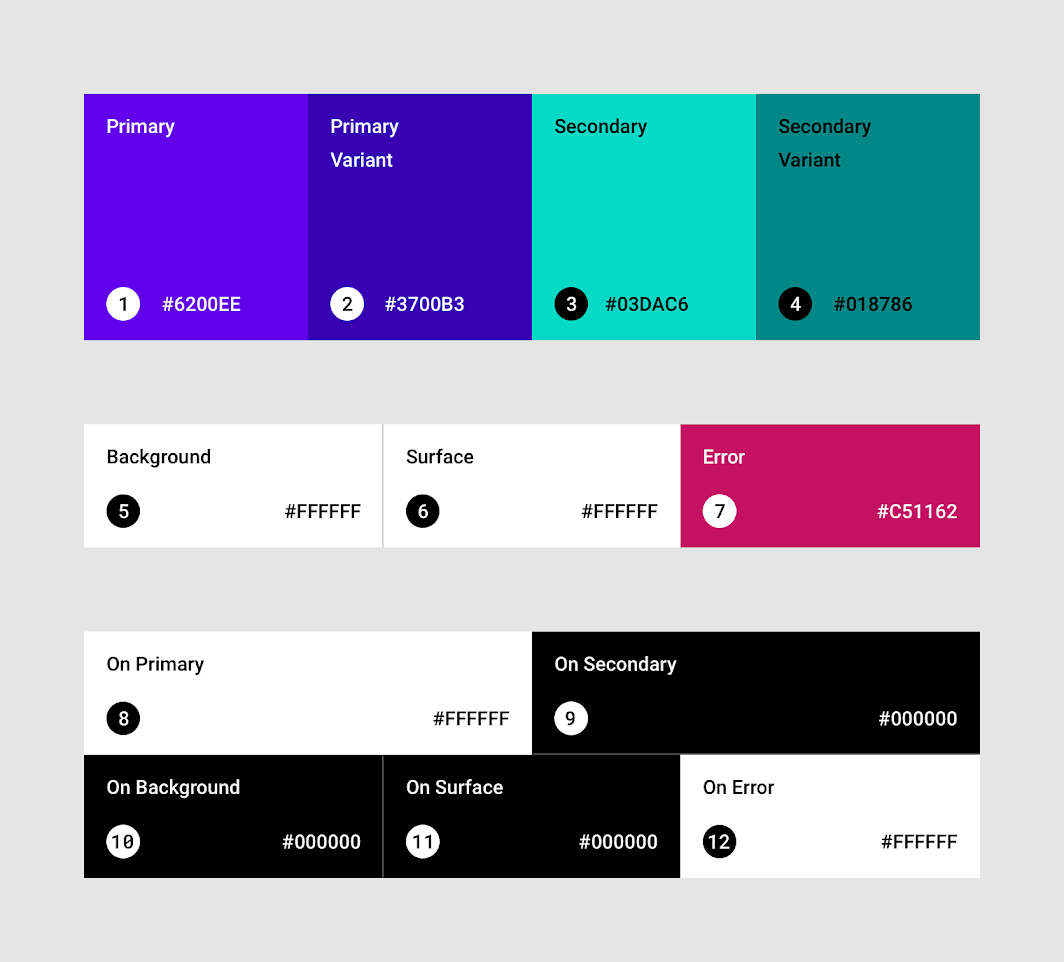
1. Execute o app. Ele ficará parecido com a captura de tela abaixo.



O app inclui uma TextView e um Button para permitir que você confira as opções de cor em um app Android real. A cor do botão mudará para a cor principal do tema nas próximas etapas.

Saiba mais sobre cores de tema

Cada parte da IU dos apps Android usa cores diferentes. Para ajudar você a usar as cores de forma significativa no app e aplicá-las consistentemente, o sistema de tema agrupa as cores em [12 atributos nomeados](https://material.io/design/material-theming/implementing-your-theme.html#color) relacionados à cor que será usada pelo texto, pelos ícones e muito mais. Seu tema não precisa especificar todos eles. Você escolherá as cores primárias e secundárias, além das cores do texto e dos ícones desenhados com essas cores.



As cores "ativadas" são usadas para textos e ícones desenhados nas diferentes superfícies.

|  |  |  |
| --- | --- | --- |
| **Nº** | **Nome** | **Atributo do tema** |
| 1 | Primary | colorPrimary |
| 2 | Primary Variant | colorPrimaryVariant |
| 3 | Secondary | colorSecondary |
| 4 | Secondary Variant | colorSecondaryVariant |
| 5 | Background | colorBackground |
| 6 | Surface | colorSurface |
| 7 | Error | colorError |
| 8 | On Primary | colorOnPrimary |
| 9 | On Secondary | colorOnSecondary |
| 10 | On Background | colorOnBackground |
| 11 | On Surface | colorOnSurface |
| 12 | On Error | colorOnError |

Confira as cores definidas no tema padrão.

1. No Android Studio, abra themes.xml (**app > res > values > topics > topics.xml**).
2. O nome do tema, Theme.TipTime, é baseado no nome do app.

<style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">

1. Observe que a linha do XML também especifica um tema pai, Theme.MaterialComponents.DayNight.DarkActionBar. DayNight é um tema predefinido na biblioteca de componentes do Material Design. DarkActionBar significa que a barra de ações usa uma cor escura. Assim como uma classe herda os atributos da classe mãe, um tema herda os atributos do tema pai.

**Observação**: os atributos de cor de tema que não estiverem definidos em um tema usarão a cor do tema pai.

1. Analise os itens no arquivo e observe que os nomes são parecidos com os do diagrama acima: colorPrimary, colorSecondary e assim por diante.

themes.xml

<resources xmlns:tools="http://schemas.android.com/tools">  
    <!-- Base application theme. -->  
    <style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">  
        <!-- Primary brand color. -->  
        <item name="colorPrimary">@color/purple\_500</item>  
        <item name="colorPrimaryVariant">@color/purple\_700</item>  
        <item name="colorOnPrimary">@color/white</item>  
        <!-- Secondary brand color. -->  
        <item name="colorSecondary">@color/teal\_200</item>  
        <item name="colorSecondaryVariant">@color/teal\_700</item>  
        <item name="colorOnSecondary">@color/black</item>  
        <!-- Status bar color. -->  
        <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>  
        <!-- Customize your theme here. -->  
    </style>  
</resources>

Nem todos os atributos de tema de cores são definidos. As cores que não forem definidas herdarão a cor do tema pai.

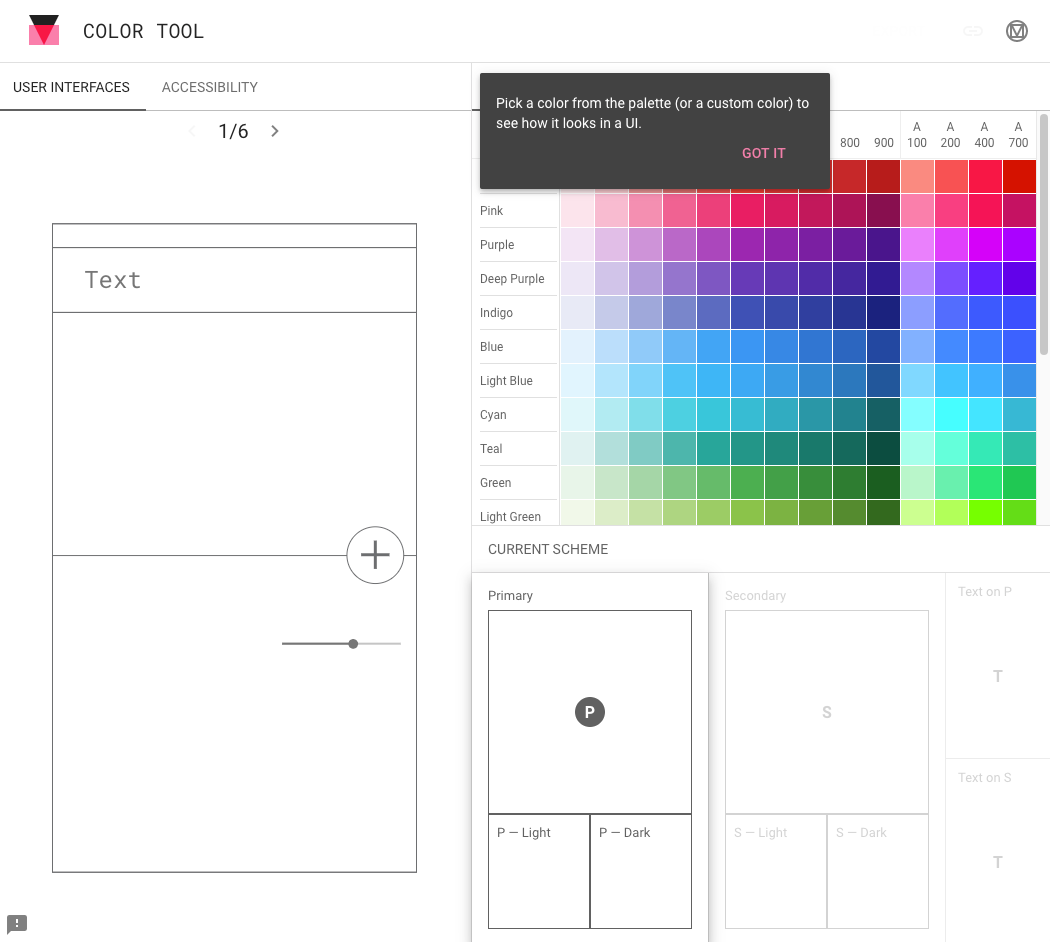
1. O Android Studio mostra uma cor pequena de exemplo na margem esquerda. 
2. Por fim, as cores são especificadas como recursos de cor, por exemplo, @color/purple\_500, em vez de usar um valor RGB diretamente.
3. Abra o arquivo colors.xml (**app > res > values > colors.xml**). Você verá os valores hexadecimais de cada recurso de cor. Lembre-se de que o #FF à esquerda é o valor alfa, o que significa que a cor é 100% opaca.

colors.xml

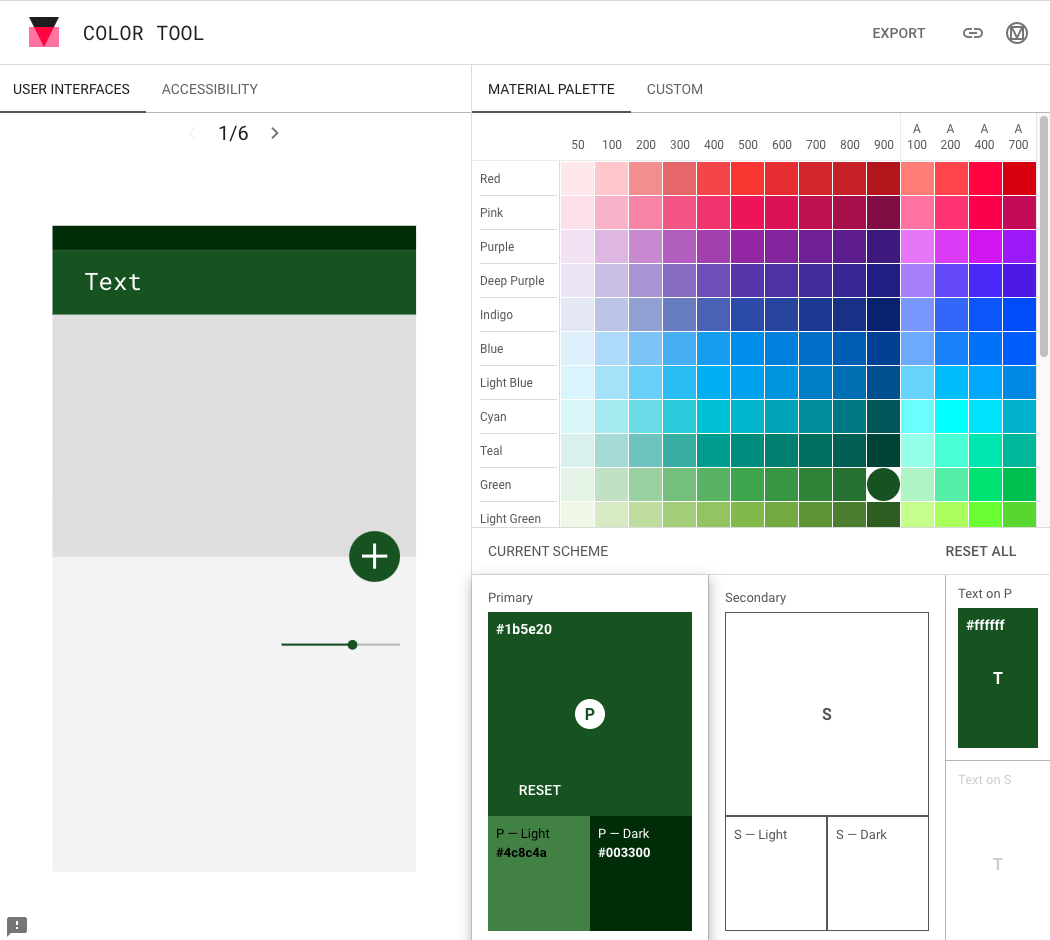
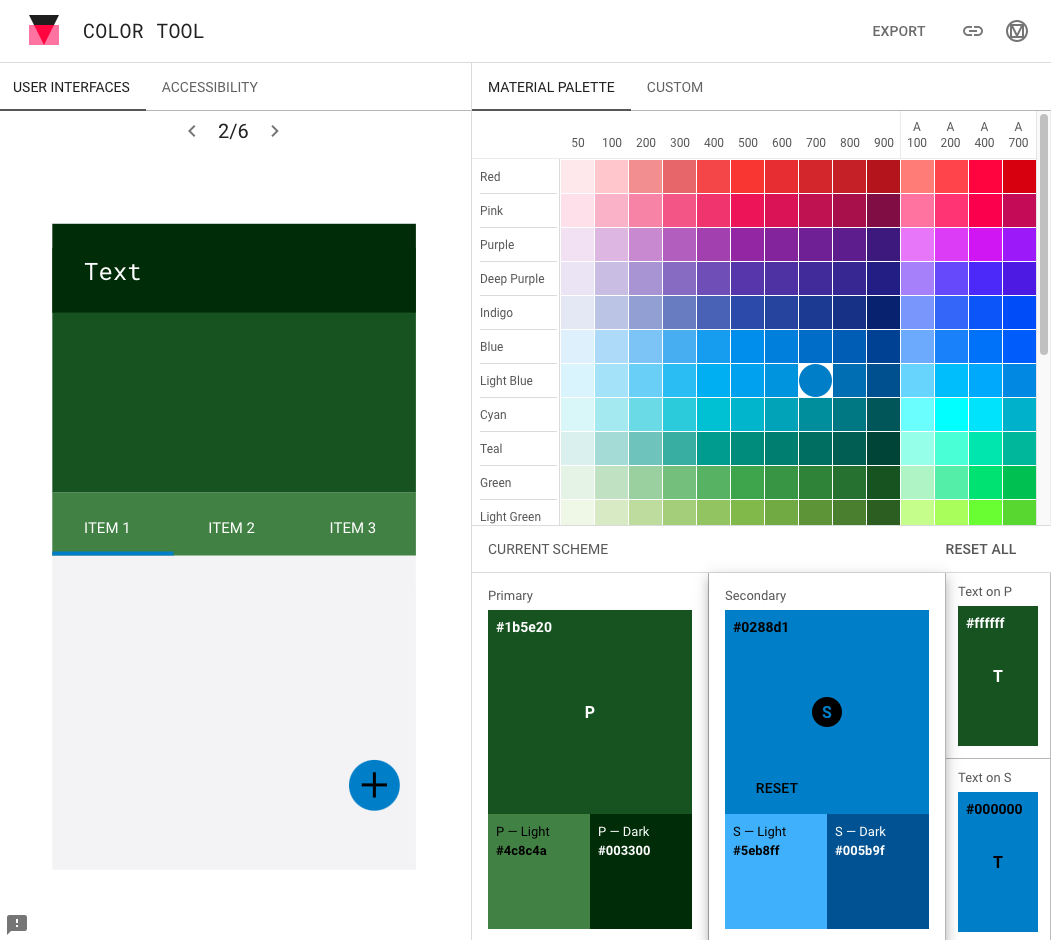
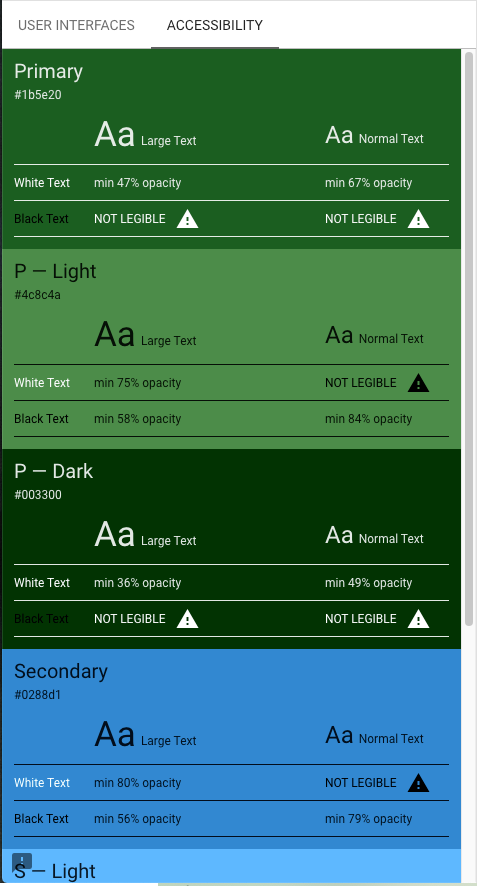
<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <color name="purple\_200">#FFBB86FC</color>  
    <color name="purple\_500">#FF6200EE</color>  
    <color name="purple\_700">#FF3700B3</color>  
    <color name="teal\_200">#FF03DAC5</color>  
    <color name="teal\_700">#FF018786</color>  
    <color name="black">#FF000000</color>  
    <color name="white">#FFFFFFFF</color>  
</resources>

[4. Escolher cores para o tema do app](https://developer.android.com/codelabs/basic-android-kotlin-training-change-app-theme?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-2&hl=pt-br#3)

Agora que você já conhece um pouco sobre os atributos do tema, é hora de escolher algumas cores. A maneira mais fácil de fazer isso é com o [Color Tool](https://material.io/resources/color/#!/?view.left=0&view.right=0), um app baseado na Web fornecido pela equipe do Material Design. A ferramenta oferece uma paleta de cores predefinidas e permite conferir facilmente como elas serão mostradas quando usadas em diferentes elementos da interface.



Escolher algumas cores

1. Comece selecionando uma cor principal na seção **Primary** (colorPrimary), como **Green 900**. A ferramenta de cor mostrará a aparência em um modelo do app, além de selecionar as variantes **Light** e **Dark**. 
2. Toque na seção **Secondary** e escolha uma cor secundária (colorSecondary) de sua preferência, por exemplo, **Light Blue 700**. A cor mostra o que será mostrado no modelo do app e seleciona novamente as variantes **Light** e **Dark**.
3. O modelo do app inclui seis simulações de "telas". Confira como suas opções de cor serão mostradas nas diferentes telas tocando nas setas acima do modelo. 
4. A ferramenta de cores também fornece a guia **Accessibility** para informar se as cores têm contraste suficiente para serem lidas quando usadas com texto preto ou branco. Para tornar seu app mais acessível, é preciso garantir que o contraste de cores seja alto o suficiente: 4,5:1 ou maior para textos pequenos e 3,0:1 ou maior para textos grandes. Saiba mais sobre [contraste de cores](https://support.google.com/accessibility/android/answer/7158390?hl=pt-br). 
5. Para primaryColorVariant e secondaryColorVariant, você pode escolher a variante clara ou escura sugerida.

**Observação:** também é possível usar o [gerador de paletas do Material Design](https://material.io/design/color/the-color-system.html#tools-for-picking-colors) para escolher uma cor secundária. Quando você escolher uma cor principal, a ferramenta sugerirá cores que sejam [complementares](https://en.wikipedia.org/wiki/Complementary_colors), [análogas](https://en.wikipedia.org/wiki/Analogous_colors) ou [triádicas](https://en.wikipedia.org/wiki/Color_scheme#Triadic).

Adicionar as cores ao app

A definição de recursos de cores facilita a reutilização consistente das mesmas cores em partes diferentes do app.

1. No Android Studio, abra colors.xml (**app > res > values > colors.xml**).
2. Após as cores existentes, defina um recurso de cor chamado green usando o valor selecionado acima, #1B5E20.

<color name="green">#1B5E20</color>

1. Continue definindo recursos para as outras cores. A maioria dos valores é gerada pela ferramenta de cores. Observe que os valores para green\_light e blue\_light são diferentes daqueles mostrados pela ferramenta. Você vai usá-los em uma etapa futura.

|  |  |
| --- | --- |
| green | #1B5E20 |
| green\_dark | #003300 |
| green\_light | #A5D6A7 |
| blue | #0288D1 |
| blue\_dark | #005B9F |
| blue\_light | #81D4FA |

Como já existem recursos de cor definidos para preto e branco, você não precisa defini-los.

O arquivo colors.xml do app agora terá esta aparência:

<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <color name="purple\_200">#FFBB86FC</color>  
    <color name="purple\_500">#FF6200EE</color>  
    <color name="purple\_700">#FF3700B3</color>  
    <color name="teal\_200">#FF03DAC5</color>  
    <color name="teal\_700">#FF018786</color>  
    <color name="black">#FF000000</color>  
    <color name="white">#FFFFFFFF</color>  
  
    <color name="green">#1B5E20</color>  
    <color name="green\_dark">#003300</color>  
    <color name="green\_light">#A5D6A7</color>  
    <color name="blue">#0288D1</color>  
    <color name="blue\_dark">#005B9F</color>  
    <color name="blue\_light">#81D4FA</color>  
</resources>

Usar as cores no tema

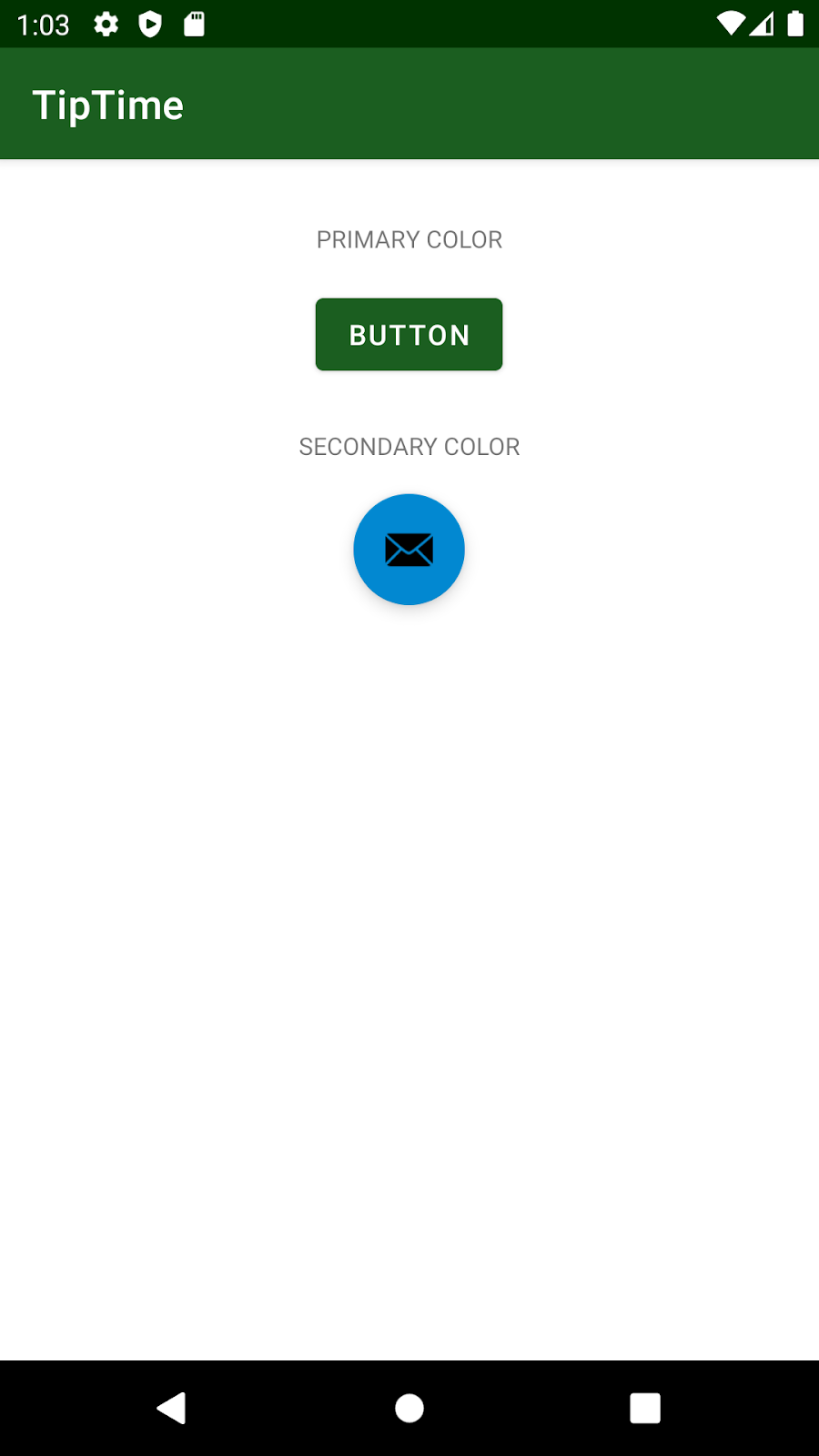
Agora que você definiu nomes para as cores selecionadas, é hora de usá-las no seu tema.

1. Abra themes.xml (**app > res > values > themes > themes.xml**).
2. Mude a colorPrimary para a cor principal que você selecionou, @color/green.
3. Mude colorPrimaryVariant para @color/green\_dark.
4. Mude colorSecondary para @color/blue.
5. Mude colorSecondaryVariant para @color/blue\_dark.
6. Verifique se as opções **Text on P** e **Text on S** ainda estão definidas como branco (#FFFFFF) e preto (#000000). Se você estiver usando a ferramenta de cores por conta própria e selecionar outras cores, talvez seja necessário definir mais recursos de cor.

Quando terminar, o tema ficará assim:

<resources xmlns:tools="http://schemas.android.com/tools">  
    <!-- Base application theme. -->  
    <style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">  
        <!-- Primary brand color. -->  
        <item name="colorPrimary">@color/green</item>  
        <item name="colorPrimaryVariant">@color/green\_dark</item>  
        <item name="colorOnPrimary">@color/white</item>  
        <!-- Secondary brand color. -->  
        <item name="colorSecondary">@color/blue</item>  
        <item name="colorSecondaryVariant">@color/blue\_dark</item>  
        <item name="colorOnSecondary">@color/black</item>  
        <!-- Status bar color. -->  
        <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>  
        <!-- Customize your theme here. -->  
    </style>  
</resources>

1. Execute seu app no emulador ou em um dispositivo e veja o visual do novo tema no app.



## [5. Tema escuro](https://developer.android.com/codelabs/basic-android-kotlin-training-change-app-theme?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-2&hl=pt-br#4)

O modelo de app incluiu um tema claro padrão e também uma variante de [tema escuro](https://developer.android.com/guide/topics/ui/look-and-feel/darktheme?hl=pt-br). Um tema escuro usa cores mais escuras e discretas e:

* Pode reduzir significativamente o consumo de energia (dependendo da tecnologia da tela do dispositivo).
* Melhora a visibilidade para usuários com problemas de visão e que tenham sensibilidade ao brilho da luz.
* Facilita o uso do dispositivo em um ambiente com pouca luz.

## Como escolher cores para o tema escuro

As cores de um tema escuro precisam ser legíveis. Os temas escuros usam uma cor de superfície escura com tons de cores limitados. Para garantir a legibilidade, as cores primárias geralmente são versões menos saturadas das cores primárias do tema claro.

Para oferecer mais flexibilidade e usabilidade em um tema escuro, recomendamos usar tons mais claros (200 a 50) em um tema escuro, em vez do tema de cores padrão (tons de saturação de 900 a 500). Anteriormente, você escolheu verde 200 e azul claro 200 como as cores claras. Para o app, você usará as cores claras como as principais e as cores primárias como as variantes.

## Atualizar a versão escura do tema

1. Abra themes.xml (night)**.** No painel **Project**, selecione Android e acesse **app > res > values > topics > topics.xml (night)**.

**Observação**: esse arquivo themes.xml é diferente do arquivo themes.xml anterior. Ele contém a versão do tema escuro. Os recursos nesse arquivo serão usados quando o **tema escuro** estiver ativado no dispositivo.

1. Mude a colorPrimary para a variante clara da cor primária que você selecionou, @color/green\_light.
2. Mude colorPrimaryVariant para @color/green.
3. Mude colorSecondary para @color/blue\_light.
4. Mude colorSecondaryVariant para @color/blue\_light. Observe que colorSecondaryVariant pode ter a mesma cor que colorSecondary.

Quando você terminar, seu arquivo themes.xml (night) terá esta aparência:

<resources xmlns:tools="http://schemas.android.com/tools">  
    <!-- Application theme for dark theme. -->  
    <style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">  
        <!-- Primary brand color. -->  
        <item name="colorPrimary">@color/green\_light</item>  
        <item name="colorPrimaryVariant">@color/green</item>  
        <item name="colorOnPrimary">@color/black</item>  
        <!-- Secondary brand color. -->  
        <item name="colorSecondary">@color/blue\_light</item>  
        <item name="colorSecondaryVariant">@color/blue\_light</item>  
        <item name="colorOnSecondary">@color/black</item>  
        <!-- Status bar color. -->  
        <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>  
        <!-- Customize your theme here. -->  
    </style>  
</resources>

1. Agora, as cores originais definidas no arquivo colors.xml, por exemplo, purple\_200, não são mais usadas; então você pode excluí-las.

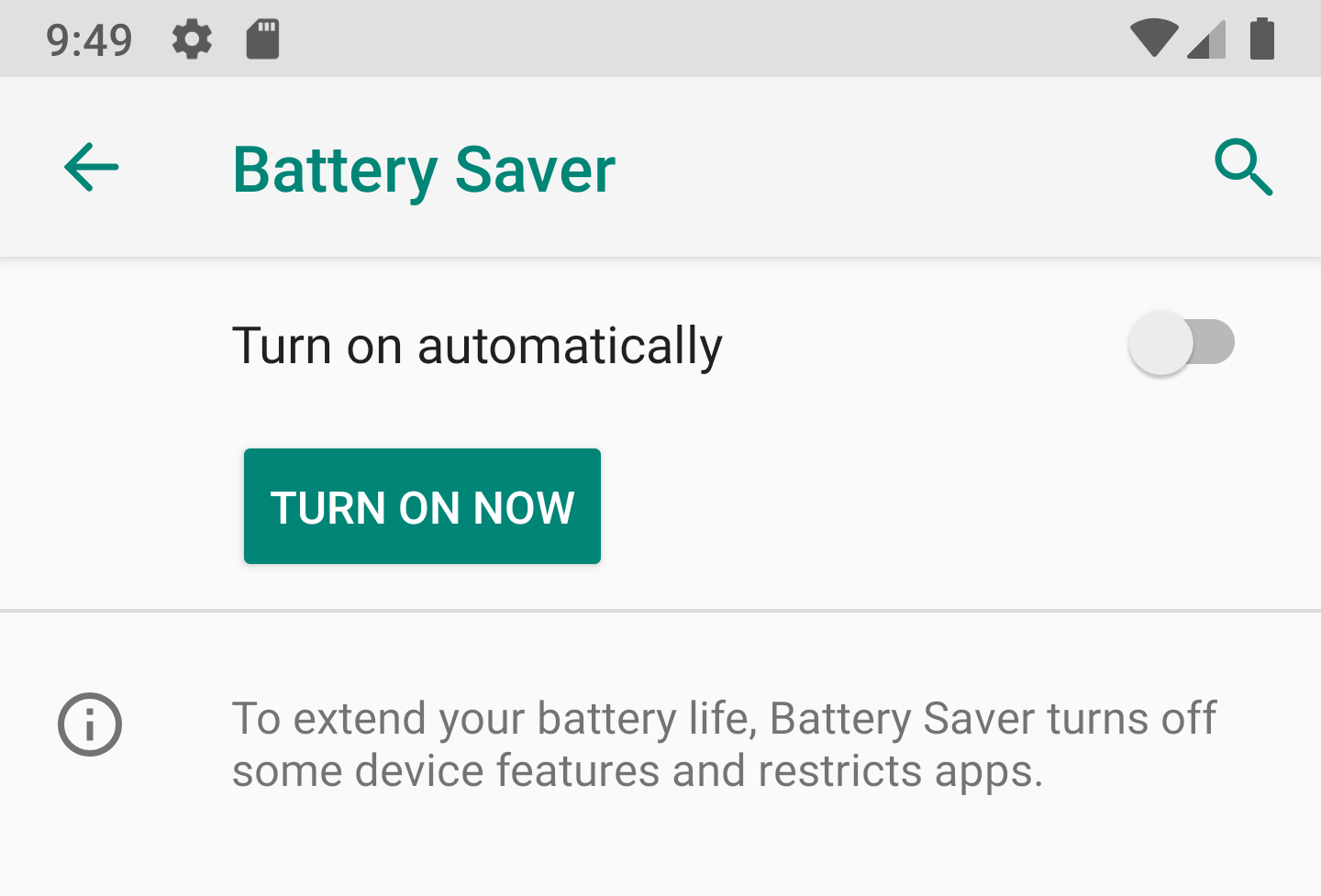
## Testar o tema escuro

Você pode ver como seu tema será exibido no modo escuro ativando-o no dispositivo.

**Observação**: o tema escuro requer um dispositivo ou emulador com a API 28 (Android 9) ou a versão 29 (Android 10) ou mais recente.

### **Para a API 28 (Android 9)**

1. Execute o app novamente.
2. Acesse o app **Config.**.
3. Na seção **Bateria**, localize **Economia de bateria**.

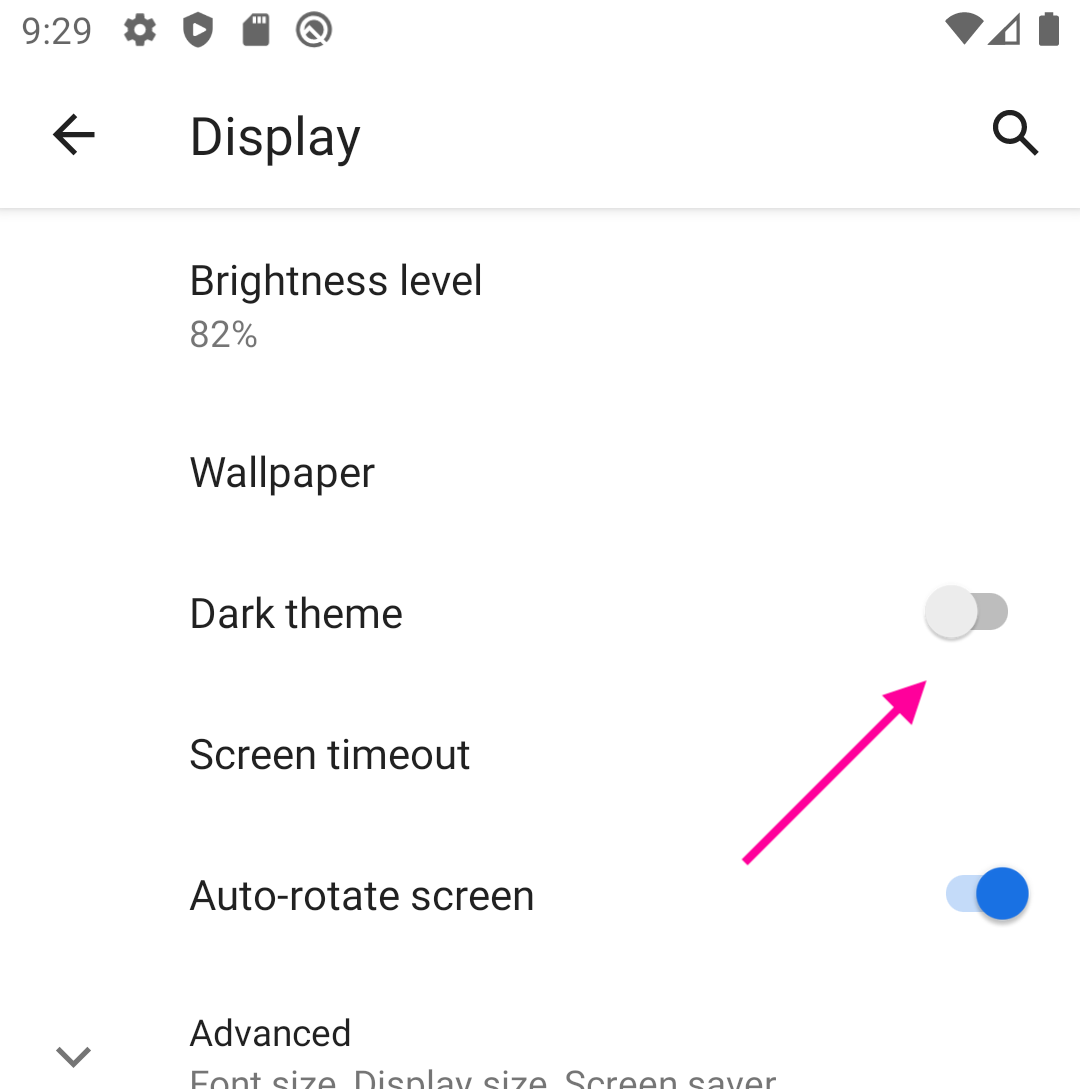


1. Pressione **Ativar agora**.

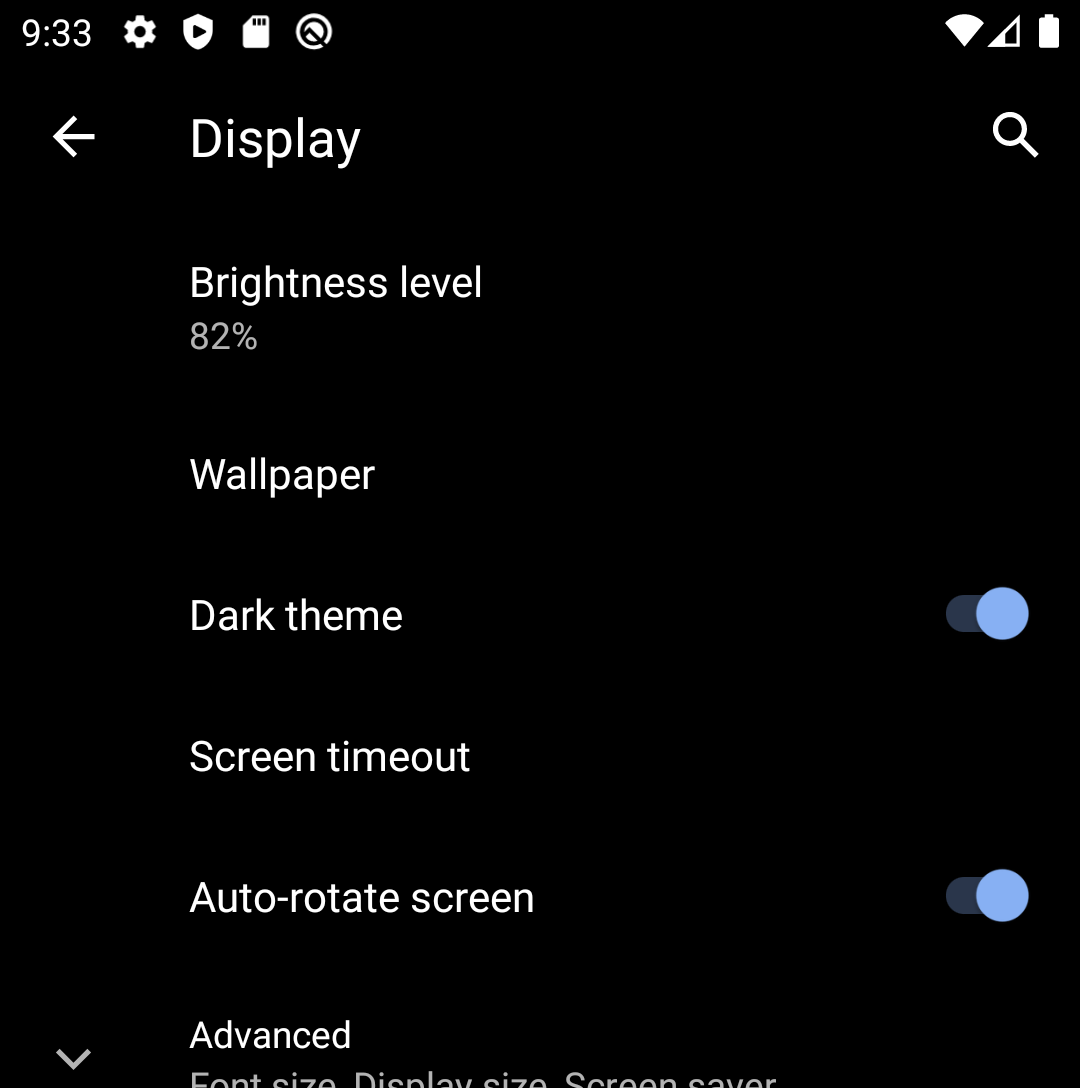
Conclua as etapas abaixo.

### **Para a API 29 (Android 10) ou mais recente**

1. Execute o app novamente.
2. Acesse o app **Config.**.
3. Na seção **Tela**, localize a chave **Tema escuro**.

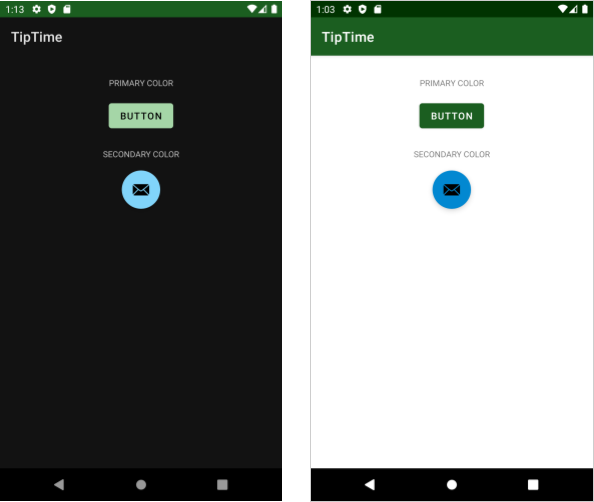


1. Mude a posição da chave **Tema escuro** para "Ativado". O dispositivo passará a usar o modo noturno.



### **Para qualquer API**

1. Volte para seu app e veja as diferenças.



A mudança mais óbvia é que o segundo plano está mais escuro e com texto claro, em vez de um segundo plano claro com texto escuro. Além disso, as cores usadas para os botões são menos vibrantes no tema escuro do que no tema claro.

Parabéns! Você criou um novo tema para o app com um tema claro e um escuro.