PdAM-I

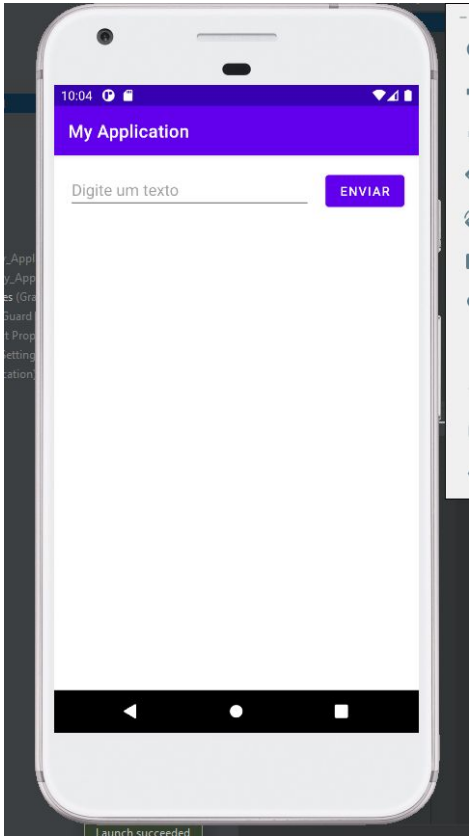
Programação de Aplicativos Mobile - I

Agenda

Criação de uma UI com o Layout Editor

Objetivo

Criar um layout simples usando o Layout editor do Android Studio



O que é uma UI?

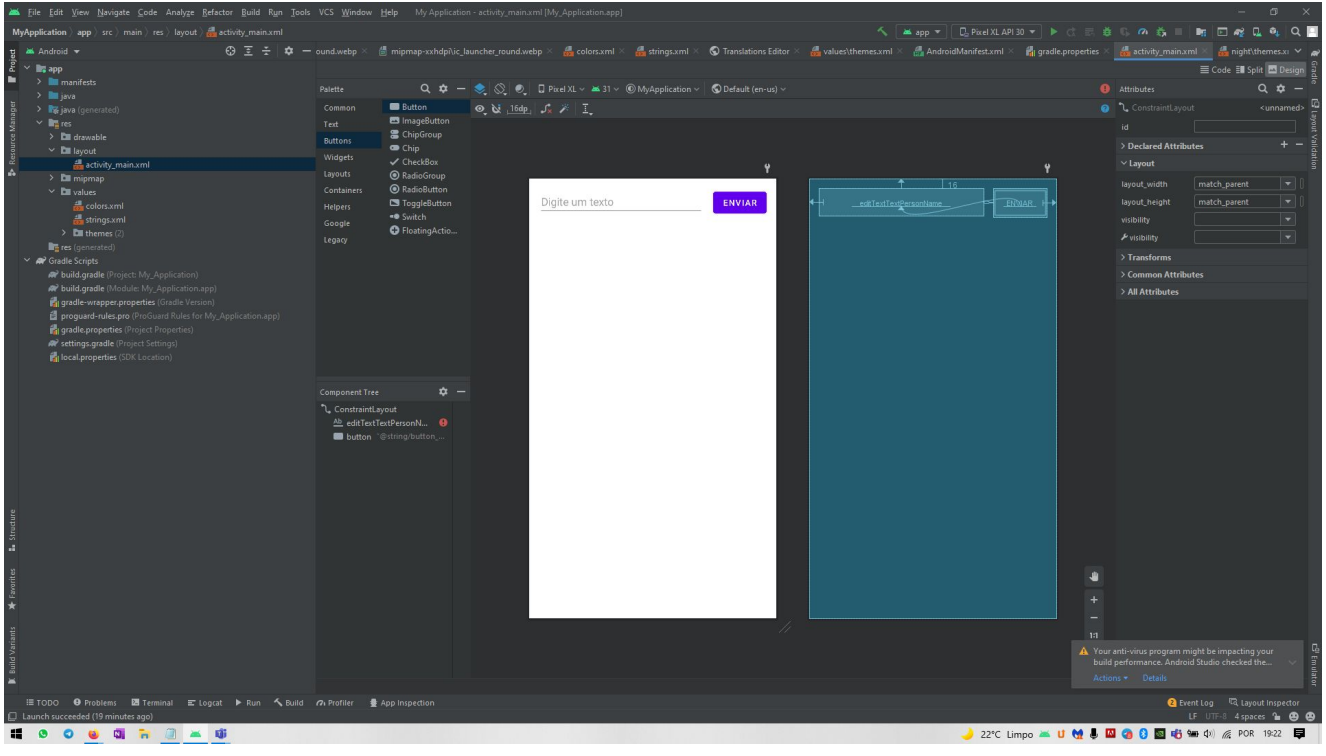
A interface do usuário (IU) de um app Android é criada usando uma hierarquia de *layouts* e *widgets*.

Os layouts são objetos de um ViewGroup.

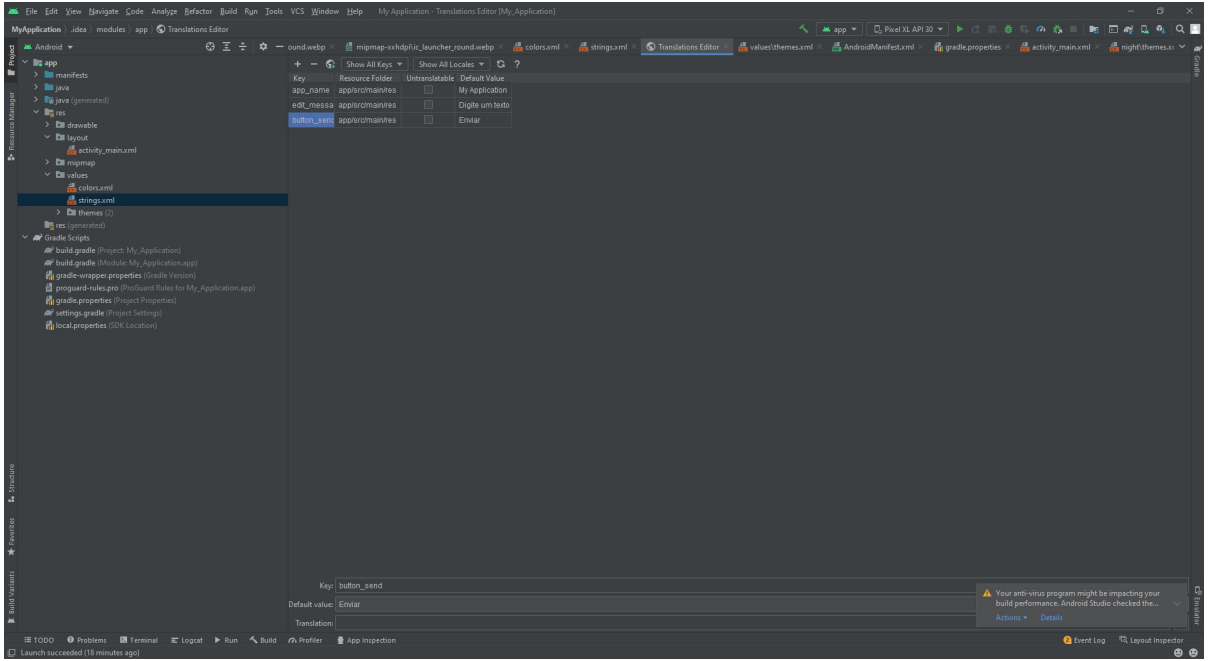
O ViewGroup é um conjunto de contêineres que controlam como as visualizações filhas são posicionadas na tela. Widgets são objetos View componentes de IU, como botões e caixas de texto.

O Android utiliza **XML** para as classes ViewGroup e View e utilizamos como ferramenta para a criação de um layout usando o **Layout Editor** do Android Studio. O Layout Editor gera o XML enquanto você arrasta e solta visualizações para criar seu layout.

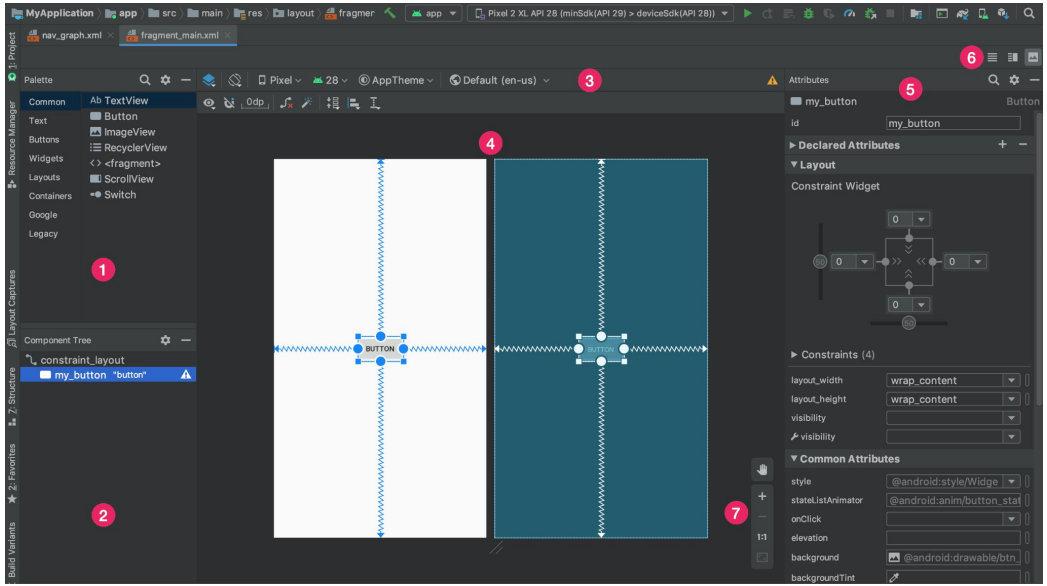
Constraint Layout

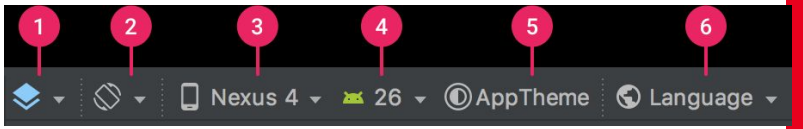


Strings


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
My Application - Translations Editor [My_Application]
MyApplication
Translations Editor
Pixel XL API 30
Android
Show All Keys
Show All Locales
Key
Resource Folder
Untranslatable
Default Value
app_name
app/src/main/res
My Application
edit_messa
Digite um texto
button_send
Enviar
manifests
java
res
drawable
layout
activity_main.xml
mipmap
values
colors.xml
strings.xml
themes
Gradle Scripts
build.gradle (Project: My_Application)
build.gradle (Module: My_Application.app)
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for My_Application.app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)
Key: button_send
Default value: Enviar
Translation:
Your anti-virus program might be impacting your build performance. Android Studio checked the...
Actions
Details
TODO
Problems
Terminal
Logcat
Run
Build
Profiler
App Inspection
Event Log
Layout Inspector
Launch succeeded (18 minutes ago)

1. **Palette:** contém várias visualizações incluindo as em grupo que você pode arrastar para o layout.
2. **Component tree**: mostra a hierarquia de componentes no seu layout.
3. **Toolbar:** clique nestes botões para configurar a aparência do layout no editor e alterar os atributos.
4. **Design editor**: edite seu layout na visualização "Design", "Blueprint" ou ambas.
5. **Attributes**: controles para os atributos da visualização selecionada.
6. **View mode:** veja seu layout no modo **Code** , **Design** ou **Split** . O modo **Split** exibe as janelas **Code** e **Design** ao mesmo tempo.
7. **Zoom and pan controls:** controle o tamanho e a posição da visualização no editor.

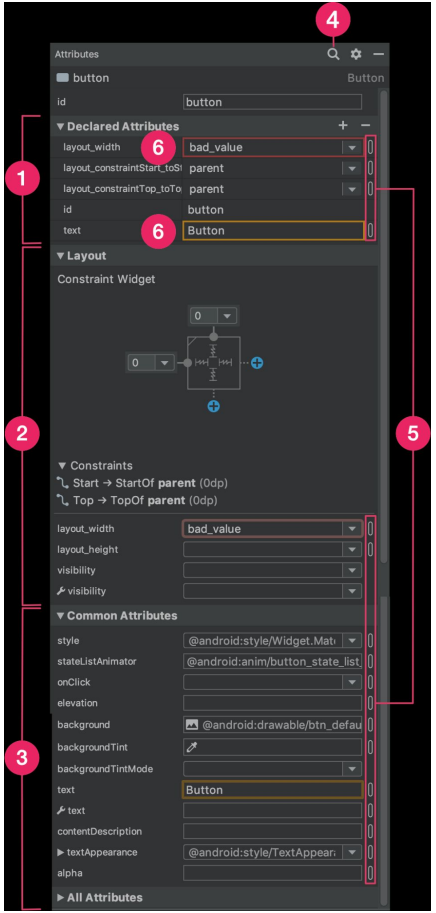




1. **Design and blueprint:** selecione como quer visualizar o layout no editor. Escolha **Design** para ter uma visualização renderizada do layout. Escolha **Blueprint** para ver apenas os contornos de cada visualização. Escolha **Design + Blueprint** para ter as duas visualizações lado a lado. Você também pode pressionar B para alternar esses tipos de visualização.
2. **Screen orientation and layout variants:** escolha entre a orientação de tela paisagem e retrato ou escolha outros modos de tela para os quais o app oferece layouts alternativos, como o modo noturno.
3. **Screen orientation and layout variants:** selecione o tipo de dispositivo (smartphone/tablet, Android TV ou Wear OS) e a configuração da tela (tamanho e densidade). Você pode escolher entre diversos tipos de dispositivos pré-configurados e suas próprias definições de AVD ou criar um novo AVD selecionando **Add Device Definition** na lista. É possível redimensionar o tamanho do dispositivo ao arrastar o canto inferior direito do layout. Você também pode pressionar D para percorrer a lista de dispositivos.
4. **API version:** selecione a versão do Android em que você quer visualizar o layout.
5. **App theme**: selecione o tema de IU a ser aplicado na visualização. Observe que isso funciona apenas para os estilos de layout compatíveis. Portanto, diversos temas dessa lista resultam em erro.
6. **Language**: selecione o idioma de exibição de strings de IU. A lista exibe apenas os idiomas disponíveis nos recursos de string. Para editar as traduções, clique em **Edit Translations** no menu suspenso.

1. A seção **Declared Attributes** lista os atributos especificados no arquivo de layout. Para adicionar um atributo, clique no botão **Add** no canto superior direito da seção.
2. A seção **Layout** contém controles para a largura e altura da visualização. Se a visualização estiver em um ConstraintLayout, esta seção também mostrará a tendência de restrição e listará as restrições que a visualização usa.
3. A seção **Common Attributes** lista atributos comuns para a visualização selecionada. Para ver todos os atributos disponíveis, abra a seção **All Attributes** na parte inferior da janela.
4. Clique no botão **Search** para procurar um atributo de visualização específico.
5. Os ícones à direita de cada valor de atributo indicam se eles são referências de recursos. Esses indicadores são sólidos quando o valor é uma referência de recurso e vazios quando o valor está codificado. Esses indicadores ajudam você a reconhecer rapidamente os valores codificados. Clicar em indicadores em qualquer um desses estados abre a janela de diálogo **Resources**, em que você pode selecionar uma referência de recurso para o atributo correspondente.
6. Um destaque vermelho ao redor de um valor de atributo indica erro com o valor. Um erro pode indicar uma entrada inválida em um atributo de definição de layout.

Um destaque laranja indica um alerta para o valor. Um aviso pode aparecer ao usar um valor codificado quando uma referência de recurso é esperada, por exemplo.



Configuração do Android para debugar

Para ativar a depuração do Android existem diferentes caminhos dependendo da versão do Android:

https://developer.android.com/studio/debug/dev-options

Para instalar o driver para cada tipo de aplicativo também podemos recorrer a essa lista:

https://developer.android.com/studio/run/oem-usb?hl=pt-br

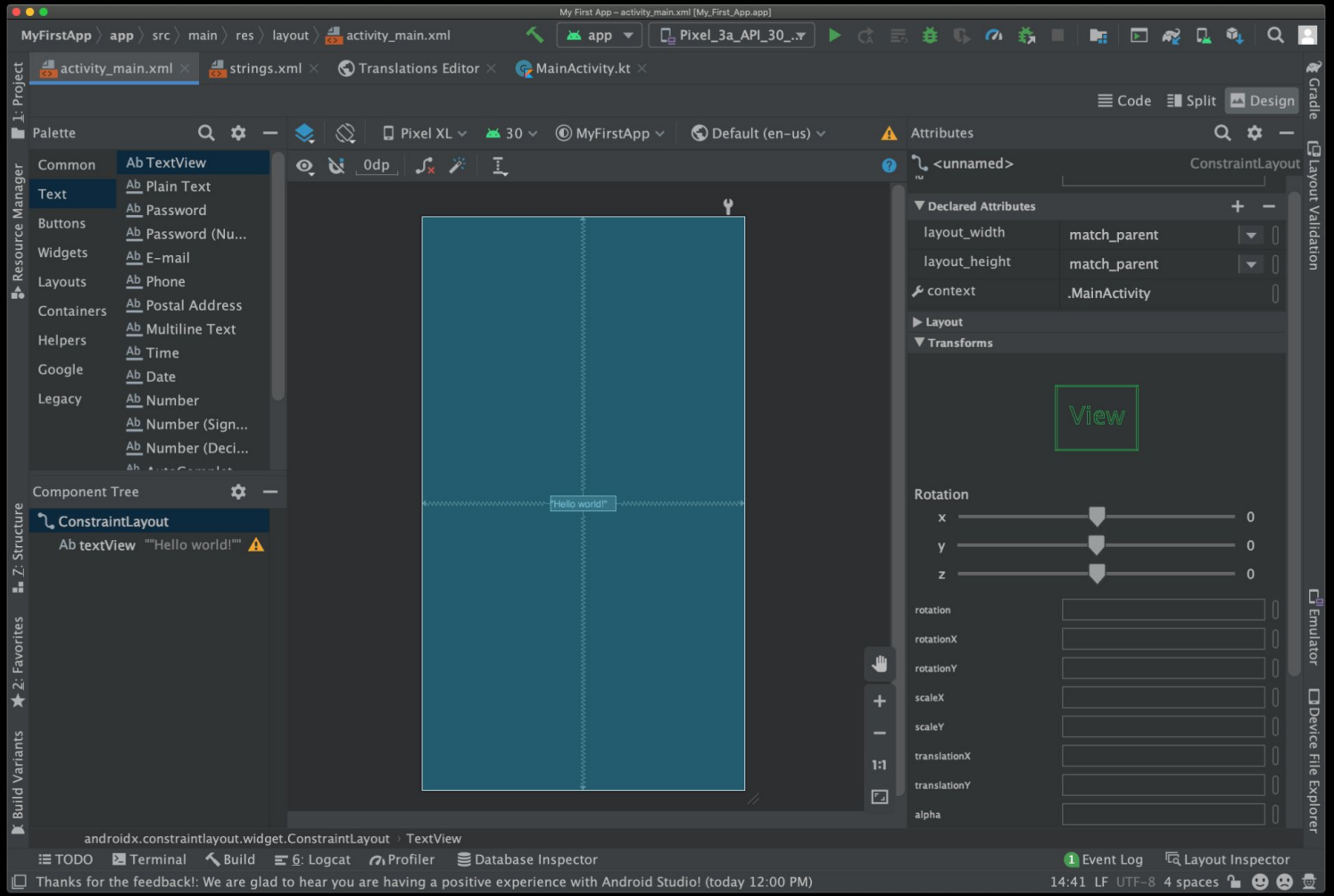
Passo a Passo

https://developer.android.com/training/basics/firstapp/building-ui

Como criar o Layout Editor

Para começar, configure o espaço de trabalho da seguinte maneira:

1. Na janela Project, abra **app > res > layout > activity_main.xml**.
2. Para liberar espaço para o Layout Editor, oculte a janela **Project**. Para fazer isso, selecione **View > Tool Windows > Project** ou clique em **Project** no lado esquerdo da tela do Android Studio.
3. Se o editor mostrar a origem XML, clique na guia **Design** no canto superior direito da janela.
4. Clique em (**Select Design Surface**) e selecione **Blueprint**.
5. Clique em (**View Options**) na barra de ferramentas do Layout Editor e verifique se a opção **Show All Constraints** está marcada.
6. Certifique-se de que a opção Autoconnect está desativada. Uma dica na barra de ferramentas exibirá (**Enable Autoconnection to Parent**) quando essa opção estiver desativada.
7. Clique em (**Default Margins**) na barra de ferramentas e selecione **16**. Se necessário, é possível ajustar as margens de cada visualização posteriormente.
8. Clique em (**Device for Preview**) na barra de ferramentas e selecione **5.5, 1440 x 2560, 560 dpi (Pixel XL)**.


My First App – activity_main.xml [My_First_App.app]
MyFirstApp app src main res layout activity_main.xml
app
Pixel_3a_API_30_..
activity_main.xml
strings.xml
Translations Editor
MainActivity.kt
Code
Split
Design
Palette
Common
Text
Buttons
Widgets
Layouts
Containers
Helpers
Google
Legacy
Ab TextView
Ab Plain Text
Ab Password
Ab Password (Nu...
Ab E-mail
Ab Phone
Ab Postal Address
Ab Multiline Text
Ab Time
Ab Date
Ab Number
Ab Number (Sign...
Ab Number (Deci...
Pixel XL
30
MyFirstApp
Default (en-us)
0dp
"Hello world!"
Component Tree
ConstraintLayout
Ab textView ""Hello world!""
Attributes
<unnamed>
ConstraintLayout
Declared Attributes
layout_width
match_parent
layout_height
match_parent
context
.MainActivity
Layout
Transforms
View
Rotation
x
y
z
rotation
rotationX
rotationY
scaleX
scaleY
translationX
translationY
alpha
androidx.constraintlayout.widget.ConstraintLayout › TextView
TODO
Terminal
Build
6: Logcat
Profiler
Database Inspector
Event Log
Layout Inspector
Thanks for the feedback!: We are glad to hear you are having a positive experience with Android Studio! (today 12:00 PM)
14:41 LF UTF-8 4 spaces
1: Project
Resource Manager
Z: Structure
2: Favorites
Build Variants
Gradle
Layout Validation
Emulator
Device File Explorer

Constraints

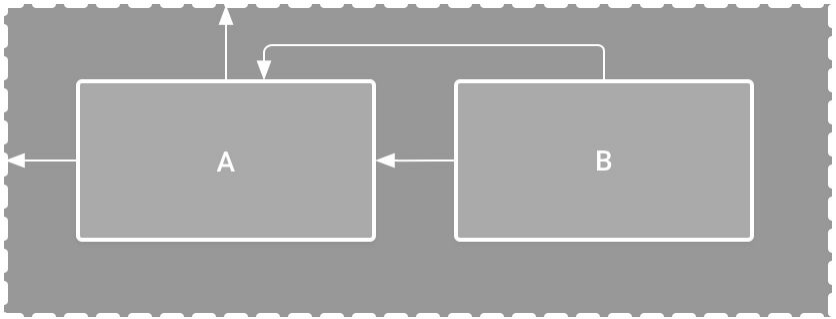
O painel **Component Tree** no canto inferior esquerdo mostra a hierarquia de visualizações do layout. Nesse caso, a visualização raiz é um ConstraintLayout, que contém apenas um objeto TextView.

ConstraintLayout é um layout que define a posição de cada visualização com base em limitações de visualizações irmãs e do layout pai. Dessa maneira, você pode criar layouts simples e complexos com uma hierarquia de visualização plana. Esse tipo de layout evita a necessidade de layouts aninhados. Um layout aninhado, que é um layout dentro de outro, conforme mostrado na Figura 2, pode aumentar o tempo necessário para desenhar a IU.

Por exemplo, você pode declarar o seguinte layout, que é mostrado na Figura 4:

- A visualização A aparece a 16 dp do topo do layout pai.
- A visualização A aparece a 16 dp da esquerda do layout pai.
- A visualização B aparece a 16 dp da direita da visualização A.
- A visualização B está alinhada com o topo da visualização A.

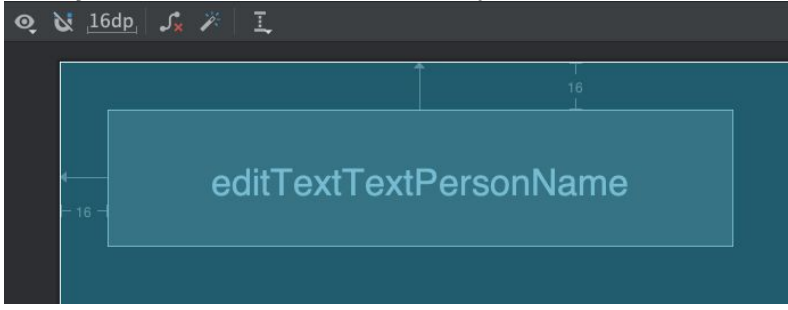
Nas seções a seguir, você criará um layout semelhante ao da figura a seguir



Adicionar uma caixa de texto

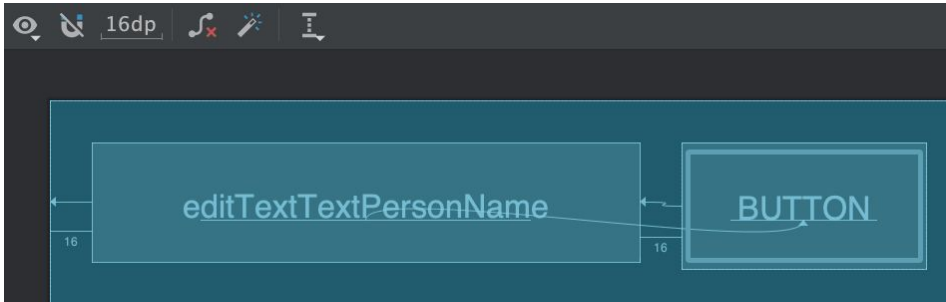
Siga estas etapas para adicionar uma caixa de texto:

1. Primeiro, você precisa remover o que já está no layout. Clique em **TextView** no painel **Component Tree** e pressione a tecla Delete.
2. No painel **Palette**, clique em **Text** para exibir os controles de texto disponíveis.
3. Arraste **Plain Text** para o editor de design e solte-o na parte superior do layout. Esse é um widget EditText que aceita entrada de texto simples.
4. Clique na visualização no editor de design. Agora você pode usar as alças quadradas para redimensionar a visualização em cada canto e as âncoras circulares de limitação em cada lado. Para ter um controle melhor, recomenda-se aumentar o zoom no editor. Para isso, use os botões **Zoom** na barra de ferramentas do Layout Editor.
5. Clique e mantenha pressionada a âncora na lateral superior, arraste-a para cima até que ela se encaixe na parte superior do layout e solte-a. Essa é uma limitação: ela limita a visualização dentro da margem padrão que foi definida. Nesse caso, você a define a 16 dp a partir do topo do layout.
6. Use o mesmo processo para criar uma limitação do lado esquerdo da visualização para o lado esquerdo do layout.



Adicionando um botão

1. No painel Palette, clique em Buttons.
2. Arraste o widget Button para o editor de design e solte-o perto do lado direito.
3. Crie uma limitação do lado esquerdo do botão para o lado direito da caixa de texto.
4. Para limitar as visualizações em um alinhamento horizontal, crie uma limitação entre as linhas de base do texto. Para fazer isso, clique com o botão direito do mouse no botão e selecione Show Baseline Mostrar ação de linha de base no Layout Editor. A âncora da linha de base aparece dentro do botão. Clique e mantenha essa âncora pressionada e arraste-a para a âncora da linha de base que aparece na caixa de texto adjacente.



Alterar as Strings da Interface

1. Abra a janela Project e o arquivo app > res > values > strings.xml.

   Esse é um arquivo de recursos de string, em que você pode especificar todas as strings de IU. Ele permite que você gerencie todas as strings da IU em um único local, o que as torna mais fáceis de encontrar, atualizar e localizar.

2. Clique em Open editor na parte superior da janela. Isso abre o Translations Editor, que fornece uma interface simples para adicionar e editar as strings padrão. Ele também ajuda a manter todas as strings traduzidas organizadas.
3. Clique em (Add Key) para criar uma nova string como o "texto de dica" da caixa de texto. Neste ponto, a janela mostrada na Figura 7 é aberta.

   Na caixa de diálogo Add Key, conclua as seguintes etapas:

   Digite "edit_message" no campo Key.

   Digite "Enter a message" no campo Default Value.

   Clique em OK.

4. Adicione outra chave chamada "button_send" com o valor "Send".

Agora você pode definir essas strings para cada visualização. Para retornar ao arquivo de layout, clique em activity_main.xml na barra de guias. Em seguida, adicione as strings da seguinte forma:

Clique na caixa de texto no layout. Se a janela Attributes ainda não estiver visível à direita, clique em Attributes na barra lateral direita.

Localize a propriedade text, atualmente definida como "Name", e exclua o valor.

Localize a propriedade hint e clique em (Pick a Resource), à direita da caixa de texto. Na caixa de diálogo exibida, clique duas vezes em edit_message na lista.

Clique no botão no layout e localize a propriedade text, que está definida como "Button". Em seguida, clique em (Pick a Resource) e selecione button_send.

Para criar um layout responsivo a diferentes tamanhos de tela, é necessário esticar a caixa de texto para preencher todo o espaço horizontal que sobre depois de considerar botão e as margens.

Antes de continuar, clique em (Select Design Surface) na barra de ferramentas e selecione Blueprint.

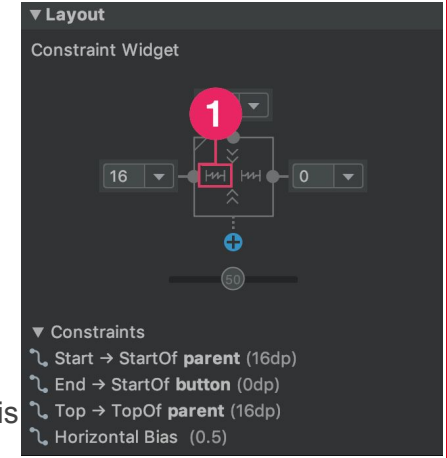
Selecione as duas visualizações. Para fazer isso, clique em uma visualização, mantenha a tecla Shift pressionada, clique na outra visualização, em seguida, clique com o botão direito do mouse em uma delas e selecione Chains > Create Horizontal Chain. O layout aparecerá como mostrado na Figura 8.

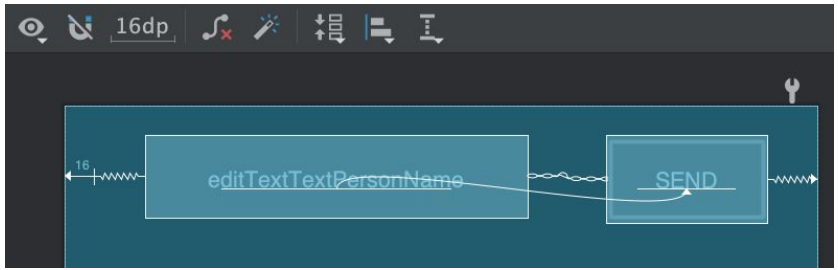
Uma cadeia é uma limitação bidirecional entre duas ou mais visualizações que permite que você disponha as visualizações encadeadas em conjunto.

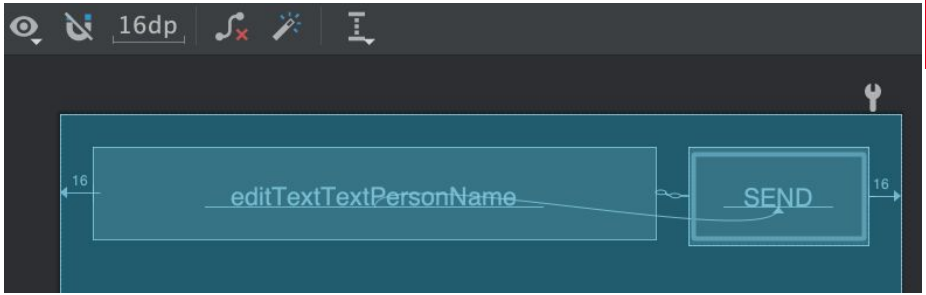
Selecione o botão e abra a janela Attributes. Em seguida, use o Constraint Widget para definir a margem direita como 16 dp.

Clique na caixa de texto para visualizar os atributos. Em seguida, clique no indicador de largura duas vezes para que ele seja definido como uma linha irregular (Match Constraints).

Match constraints significa que a largura se expande para atender à definição das limitações horizontais e das margens. Portanto, a caixa de texto será esticada para preencher o espaço horizontal que permanece depois de considerar o botão e todas as margens.







Executar a atividade

Se o app já tiver sido instalado no dispositivo na lição anterior, basta clicar em (Apply Changes) na barra de ferramentas para atualizar o app com o novo layout. Se preferir, clique em Run 'app' para instalar e executar o app.

Adicionaremos funcionalidade ao botão de enviar a seguir.

Adicionando funcionalidades

https://developer.android.com/training/basics/firstapp/starting-activity

Adicionando o botão Enviar

Siga as etapas a seguir para adicionar um método à classe MainActivity chamada ao tocar no botão **Send**:

No arquivo **app > java > com.example.myfirstapp > MainActivity**, adicione o stub de método sendMessage() a seguir:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }

    /** Essa função é chamada sempre que o usuário toca no botão enviar */
    fun sendMessage(view: View) {
        // Do something in response to button
    }
}
```

Importações

Talvez seja exibido um erro, porque o Android Studio não **pode resolver a classe View** usada como argumento do método.

Para limpar o erro, clique na declaração View, coloque o cursor sobre ela e pressione Alt+Enter ou Option+Enter no Mac, para executar uma correção rápida. Se um menu for exibido, selecione **Import class**.

Retorne ao arquivo **activity_main.xml** para chamar o método com o botão:

1. Selecione o botão no Layout Editor.
2. Na janela **Attributes**, localize a propriedade **onClick** e selecione **sendMessage [MainActivity]** na lista suspensa.

Agora, ao tocar no botão, o sistema chamará o método sendMessage().

Observe os detalhes desse método. Eles são necessários para que o sistema reconheça o método como compatível com o atributo android:onClick. Especificamente, o método tem as seguintes características:

- Acesso público.
- Um valor vazio ou, em Kotlin, um valor de retorno unit implícito.
- View como o único parâmetro. Esse é o objeto View em que você clicou no final da Etapa 1.

Intent

Criar uma intent

Uma Intent (intenção) é um objeto que fornece vínculos de tempo de execução entre componentes separados, como duas atividades. A Intent representa a intenção do app de fazer algo. Você pode usar intents para uma ampla variedade de tarefas e aqui utilizaremos o Intent para dar início a outra Activity.

Em MainActivity, adicione a constante EXTRA_MESSAGE e o código sendMessage(), conforme mostrado aqui

```
const val EXTRA_MESSAGE = "com.example.myfirstapp.MESSAGE"

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }

    /** Chamado quando o usuário apertar em enviar */
    fun sendMessage(view: View) {
        val editText = findViewById<EditText>(R.id.editTextTextPersonName)
        val message = editText.text.toString()
        val intent = Intent(this, DisplayMessageActivity::class.java).apply {
            putExtra(EXTRA_MESSAGE, message)
        }
        startActivity(intent)
    }
}
```

Importações

Para o código anterior funcionar adicione as seguintes importações.

```
import androidx.appcompat.app.AppCompatActivity
import android.content.Intent
import android.os.Bundle
import android.view.View
import android.widget.EditText
```

Ainda há um erro para DisplayMessageActivity, mas isso não é um problema. Você o corrigirá na próxima seção.

Veja o que está acontecendo em sendMessage():

- O construtor do Intent usa dois parâmetros, um Context e um Class.

  O parâmetro Context é usado primeiro porque a classe Activity é uma subclasse de Context.

  Nesse caso, o parâmetro Class do componente do app, ao qual o sistema envia o Intent, que, nesse caso, é a atividade a ser iniciada.

- O método putExtra() adiciona o valor de EditText à intent. Uma Intent pode carregar tipos de dados como pares de chave-valor chamados de *extras*.

  Sua chave é uma constante EXTRA_MESSAGE pública porque a próxima atividade usará a chave para recuperar o valor do texto. É recomendável definir chaves para intents extras com o nome do pacote do app como prefixo. Isso garante que as chaves sejam únicas caso seu app interaja com outros.

- O método startActivity() inicia uma instância da DisplayMessageActivity especificada pela Intent. Em seguida, você precisa criar essa classe.

Criar a segunda Activity

Para criar a segunda atividade, siga as etapas a seguir:

1. Na janela **Project** clique com o botão direito do mouse na pasta **app** e selecione **New > Activity > Empty Activity**.
2. Na janela **Configure Activity**, insira "DisplayMessageActivity" em **Activity Name**. Deixe todas as outras propriedades definidas como padrão e clique em **Finish**.

O Android Studio faz três coisas automaticamente:

- Cria o arquivo DisplayMessageActivity.
- Cria o arquivo de layout activity_display_message.xml, que corresponde ao arquivo DisplayMessageActivity.
- Adiciona o elemento <activity> se necessário em AndroidManifest.xml.

Se você executar o app e tocar no botão na primeira atividade, a segunda atividade será iniciada, mas estará vazia. Isso ocorre porque a segunda atividade utiliza o layout vazio fornecido pelo modelo.

Adicionando uma TextView

A nova atividade inclui um arquivo de layout vazio. Siga estas etapas para adicionar uma visualização de texto ao local em que a mensagem aparece:

Abra o arquivo app > res > layout > activity_display_message.xml.

Clique em Enable Autoconnection to Parent na barra de ferramentas. Isso habilita a conexão automática.

No painel Palette, clique em Text, arraste uma TextView para o layout e solte-a próximo ao centro da parte superior do layout para que se encaixe na linha vertical exibida. O Autoconnect adiciona limitações esquerda e direita para colocar a visualização no centro horizontal.

Crie mais uma restrição do topo da visualização de texto para o topo do layout utilizando as constraints.

Opcionalmente, faça mudanças no estilo do texto expandindo textAppearance no painel Common Attributes da janela Attributes e altere atributos, como textSize e textColor.

Exibindo a mensagem

Nesta etapa, você modificará a segunda atividade para exibir a mensagem que foi transmitida pela primeira.

Em DisplayMessageActivity, adicione o seguinte código ao método onCreate():

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_display_message)

    // Get the Intent that started this activity and extract the string
    val message = intent.getStringExtra(EXTRA_MESSAGE)

    // Capture the layout's TextView and set the string as its text
    val textView = findViewById<TextView>(R.id.textView).apply {
        text = message
    }
}
```

Adicionar opção de ‘voltar’

Todas as telas que não são o ponto de entrada principal, ou seja, que não são a tela inicial, precisam oferecer uma navegação que direcione o usuário à tela do pai lógico na hierarquia do app. Para fazer isso, adicione um botão **Up** na barra de apps.

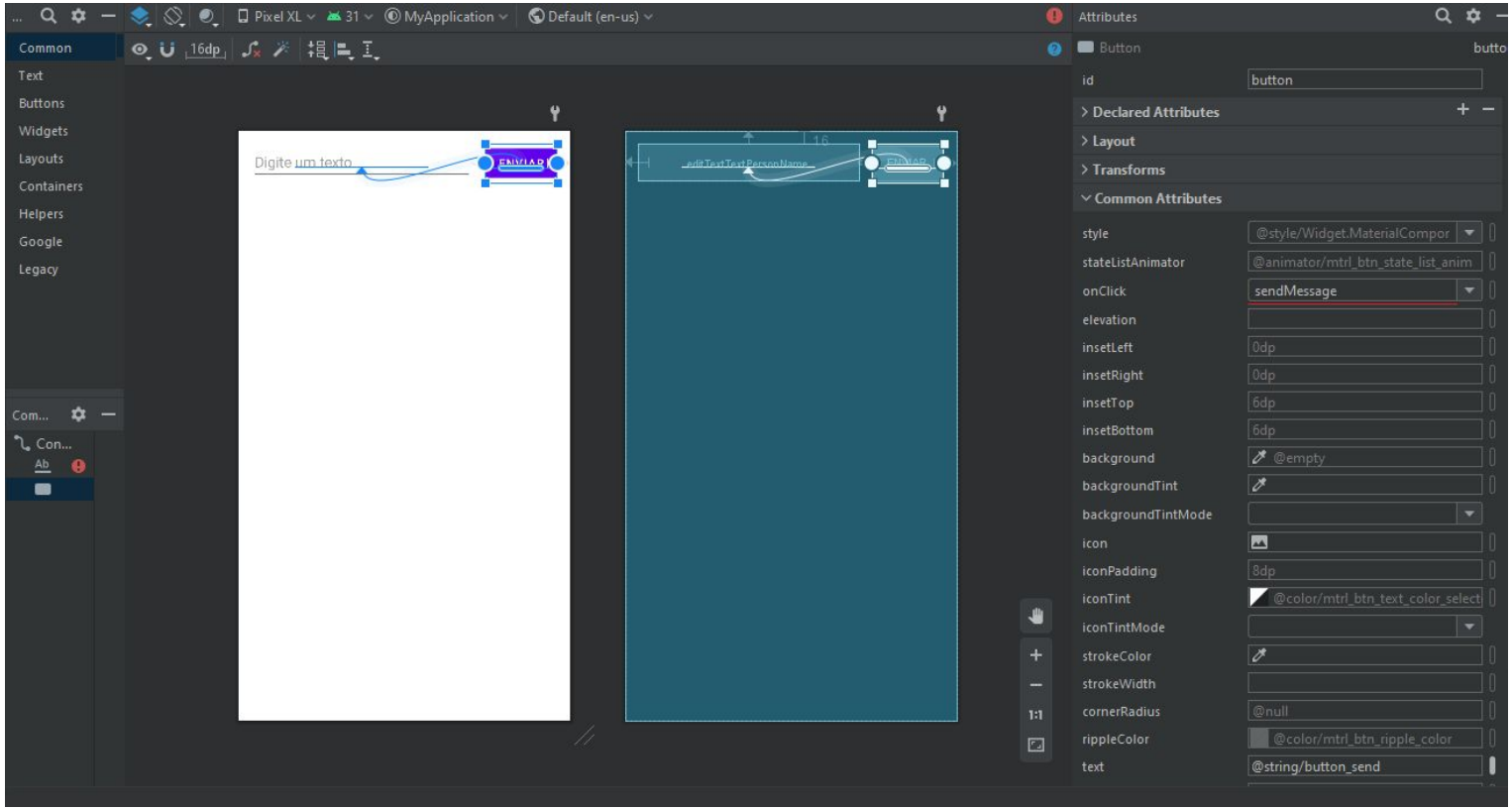
Para adicionar um botão **Up**, é necessário declarar qual atividade é o pai lógico no arquivo AndroidManifest.xml. Abra o arquivo em **app > manifests > AndroidManifest.xml**, localize a tag <activity> para DisplayMessageActivity e a substitua-a pelo seguinte:

```
<activity android:name=".DisplayMessageActivity"
          android:parentActivityName=".MainActivity">
    <!-- The meta-data tag is required if you support API level
15 and lower -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity" />
</activity>
```

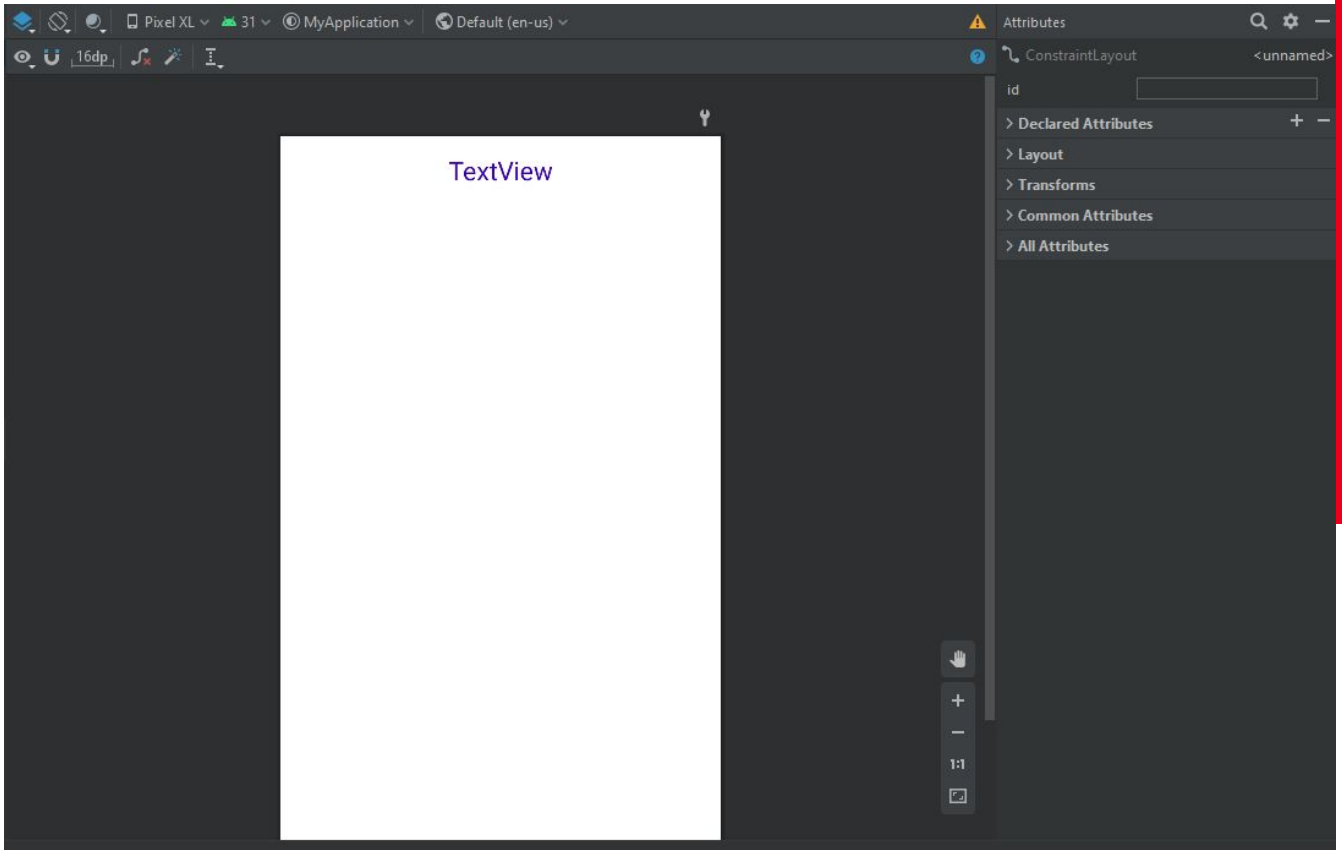
Main Activity

```
package com.example.myapplication

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.EditText

const val EXTRA_MESSAGE = "com.example.myapplication.MESSAGE"

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }

    fun sendMessage(view: View){
        //Do something in response to button press
        val editText = findViewById<EditText>(R.id.editTextTextPersonName)
        val message = editText.text.toString()
        val intent = Intent( packageContext: this, DisplayMessageActivity::class.java).apply {  this: Intent
            putExtra(EXTRA_MESSAGE, message)
        }
        startActivity(intent)
        }
}
```

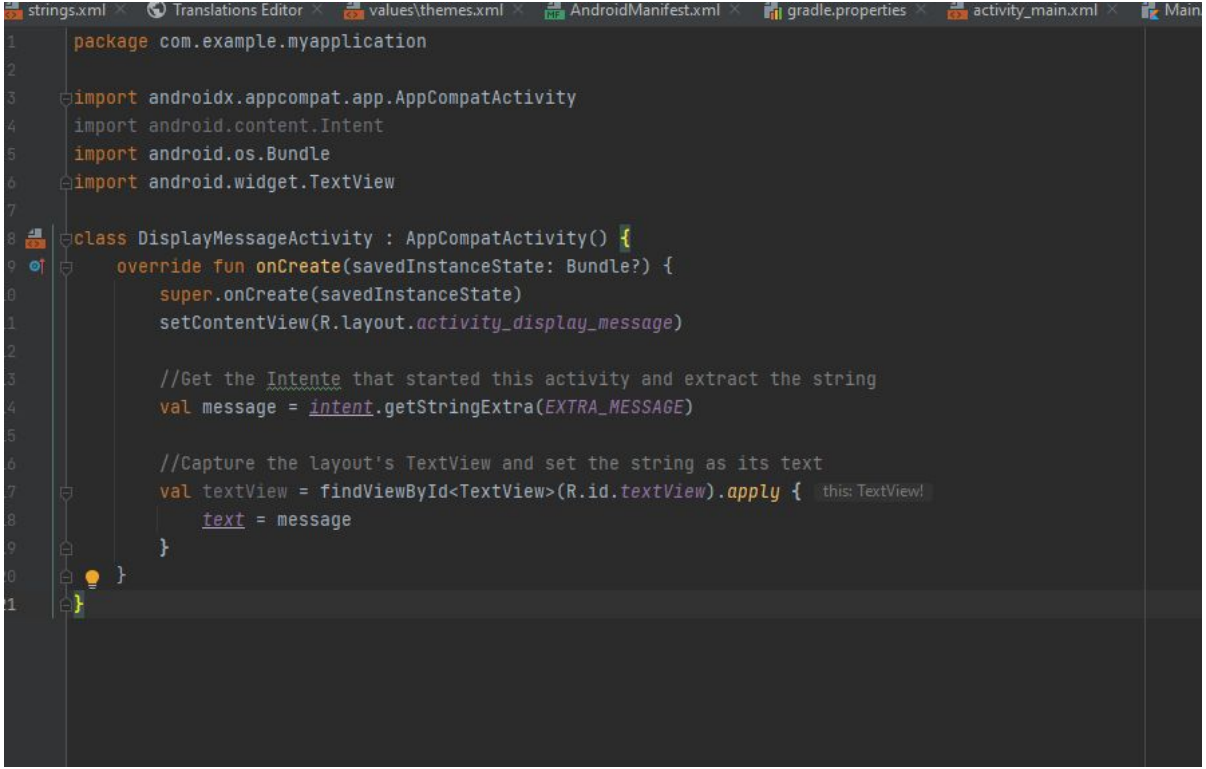
MainActivity Layout

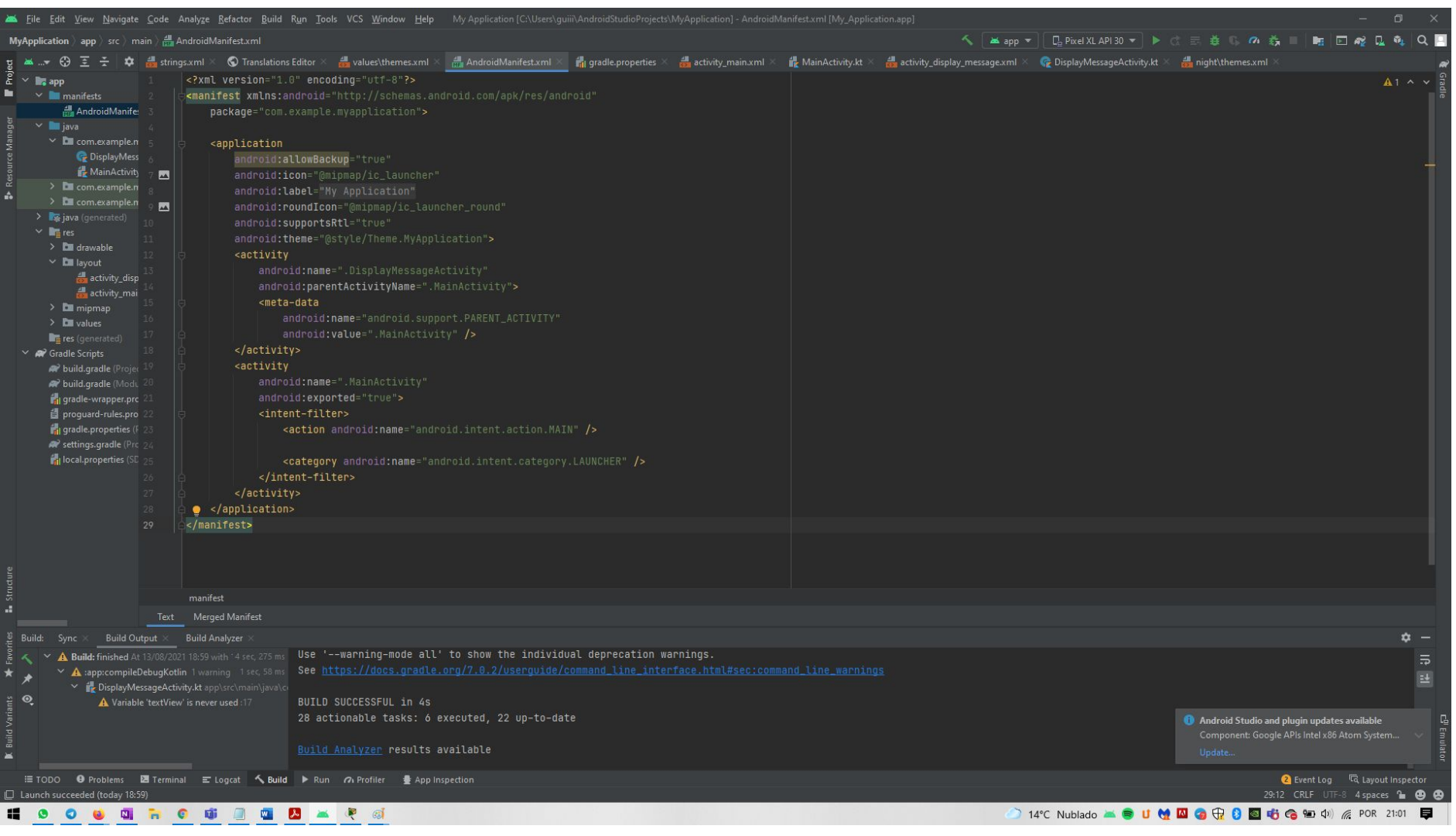


DisplayMessageLayout



DisplayMessageActivity




File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
My Application [C:\Users\guiii\AndroidStudioProjects\MyApplication] - AndroidManifest.xml [My_Application.app]
MyApplication app src main AndroidManifest.xml
app
Pixel XL API 30
strings.xml
Translations Editor
values\themes.xml
AndroidManifest.xml
gradle.properties
activity_main.xml
MainActivity.kt
activity_display_message.xml
DisplayMessageActivity.kt
night\themes.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.myapplication">
<application
android:allowBackup="true"
android:icon="@mipmap/ic_launcher"
android:label="My Application"
android:roundIcon="@mipmap/ic_launcher_round"
android:supportsRtl="true"
android:theme="@style/Theme.MyApplication">
<activity
android:name=".DisplayMessageActivity"
android:parentActivityName=".MainActivity">
<meta-data
android:name="android.support.PARENT_ACTIVITY"
android:value=".MainActivity" />
</activity>
<activity
android:name=".MainActivity"
android:exported="true">
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
</application>
</manifest>
manifest
Text
Merged Manifest
Build:
Sync
Build Output
Build Analyzer
Build: finished At 13/08/2021 18:59 with 4 sec, 275 ms
:app:compileDebugKotlin 1 warning 1 sec, 58 ms
DisplayMessageActivity.kt app\src\main\java\c
Variable 'textView' is never used :17
Use '--warning-mode all' to show the individual deprecation warnings.
See https://docs.gradle.org/7.0.2/userguide/command_line_interface.html#sec:command_line_warnings
BUILD SUCCESSFUL in 4s
28 actionable tasks: 6 executed, 22 up-to-date
Build Analyzer results available
Android Studio and plugin updates available
Component: Google APIs Intel x86 Atom System...
Update...
TODO
Problems
Terminal
Logcat
Build
Run
Profiler
App Inspection
Event Log
Layout Inspector
Launch succeeded (today 18:59)
29:12 CRLF UTF-8 4 spaces
14°C Nublado
POR 21:01

Referências

Diretrizes para desenvolvimento Android

https://developer.android.com/about?hl=pt-br