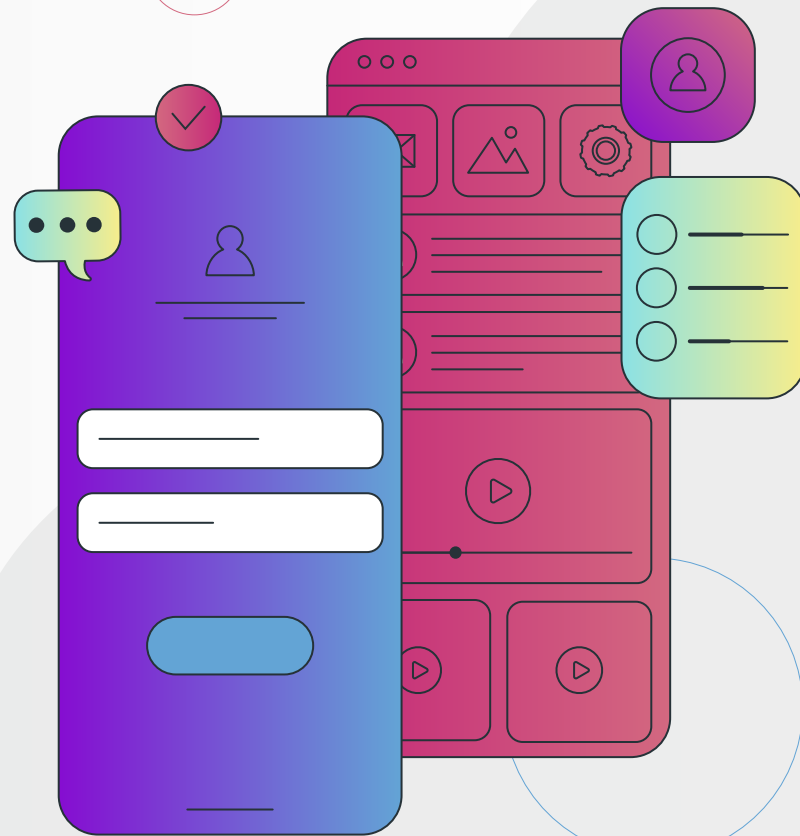


Helpers

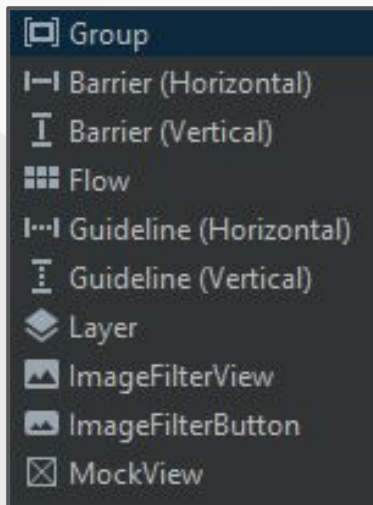
Acadêmicos: Hevandro,
Sergio e Tiago Vanelli.



Introdução

- O que são *Helpers* ?

São classes ou bibliotecas que oferecem funcionalidades adicionais para a criação de layouts, como o uso de grades (Grids), sistemas de posicionamento flexíveis ou regras específicas para o posicionamento de elementos na tela.



Componentes

01 Group

02 Barrier

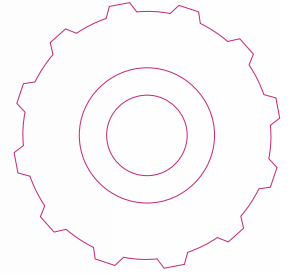
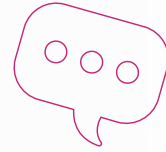
03 Flow

04 Layer

05 MockView



GROUP



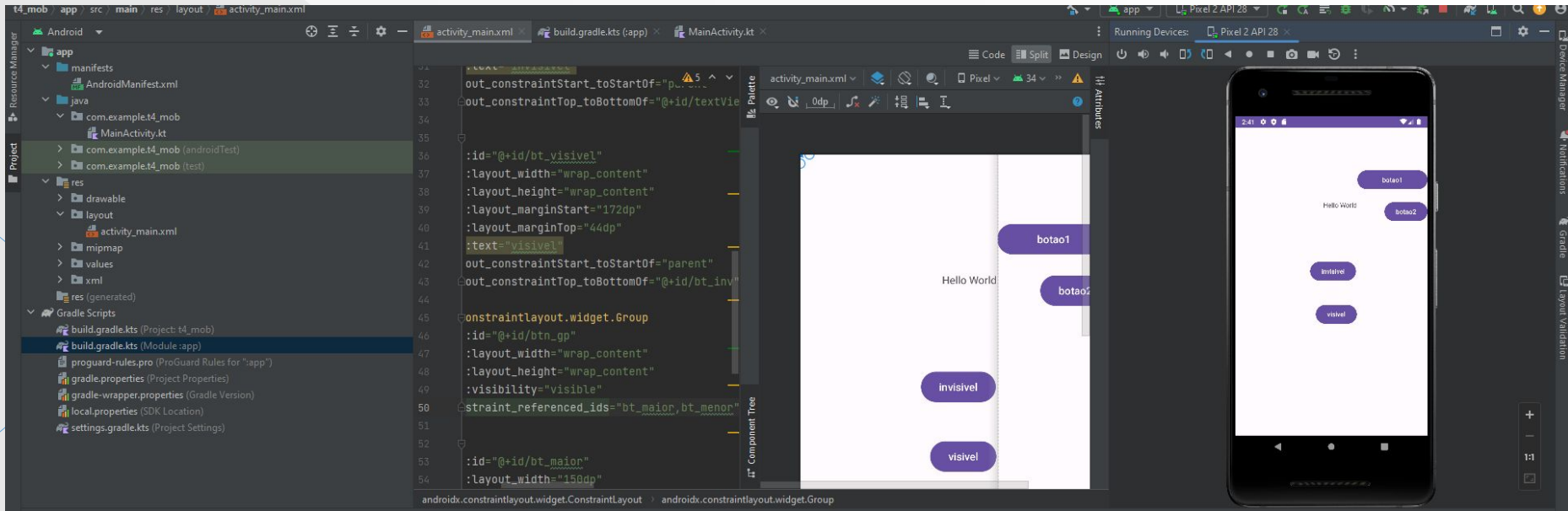


Group

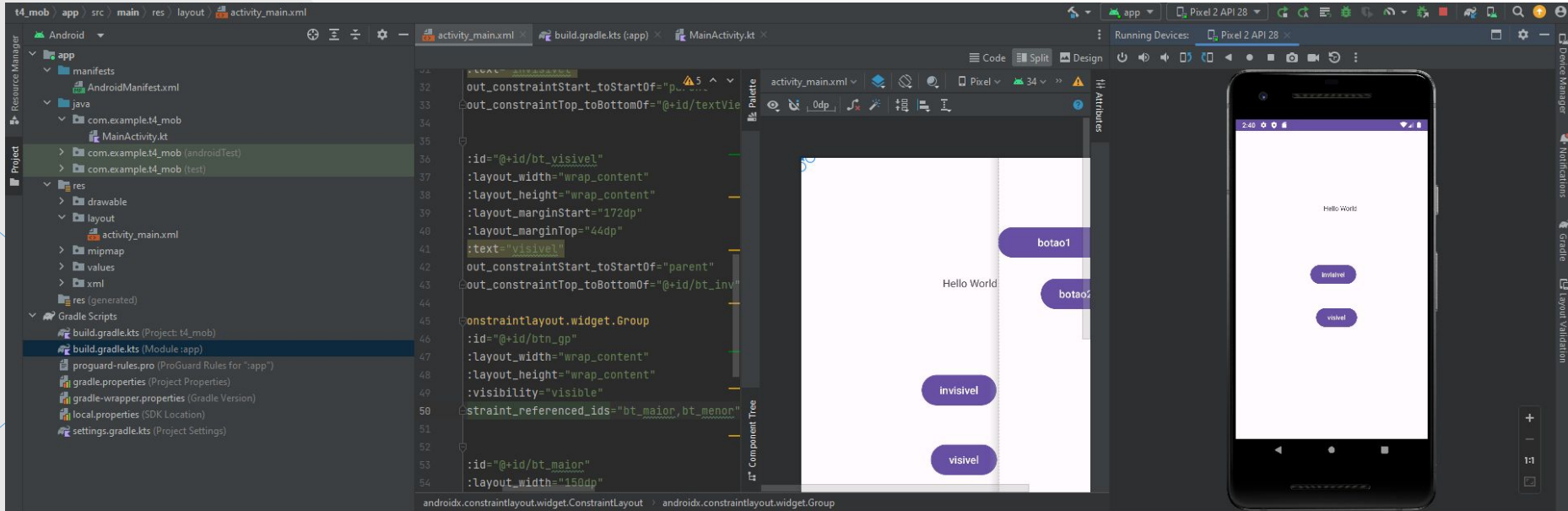
- O que faz o *Group* ?

Esse helper tem como função agrupar widgets (através de referências) para poder controlar a visibilidade deles, deixando o código mais limpo.

Exemplo de group



Exemplo de group





Barrier

Enter a subtitle here if you need it



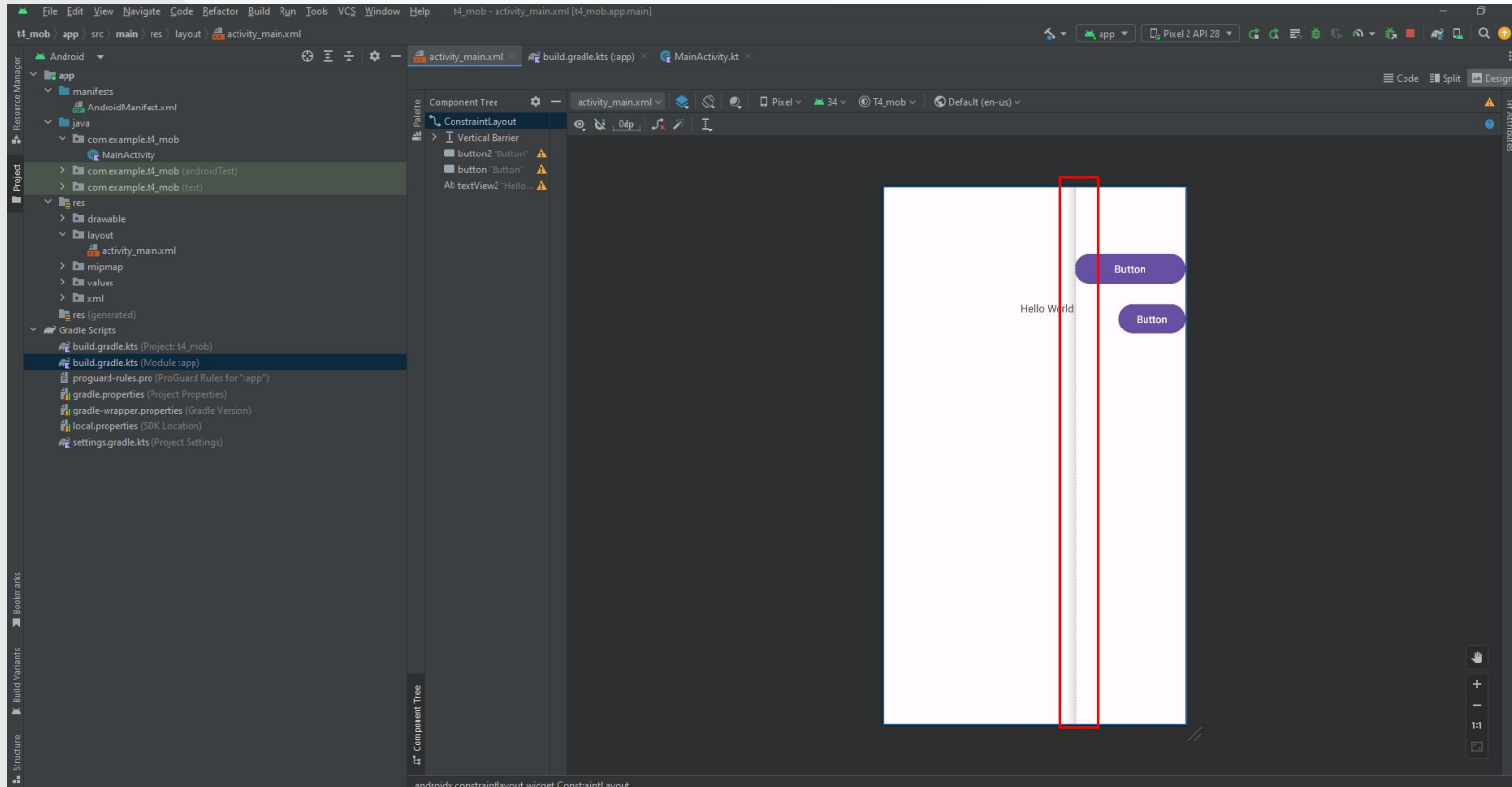


Barrier

- O que fazem o *Barrier*?

A barreira é um helper que faz referência a vários widgets como entrada, criando uma guideline baseada no widget mais extremo do lado especificado. A barrier é geralmente usada como uma ferramenta visual para criar layouts

Exemplo de barrier





MOCKVIEW

O que é o MockView?

MockView é um elemento de interface que serve para criar uma espécie de "rascunho" ou esboço visual do design que planejamos implementar em nosso aplicativo.

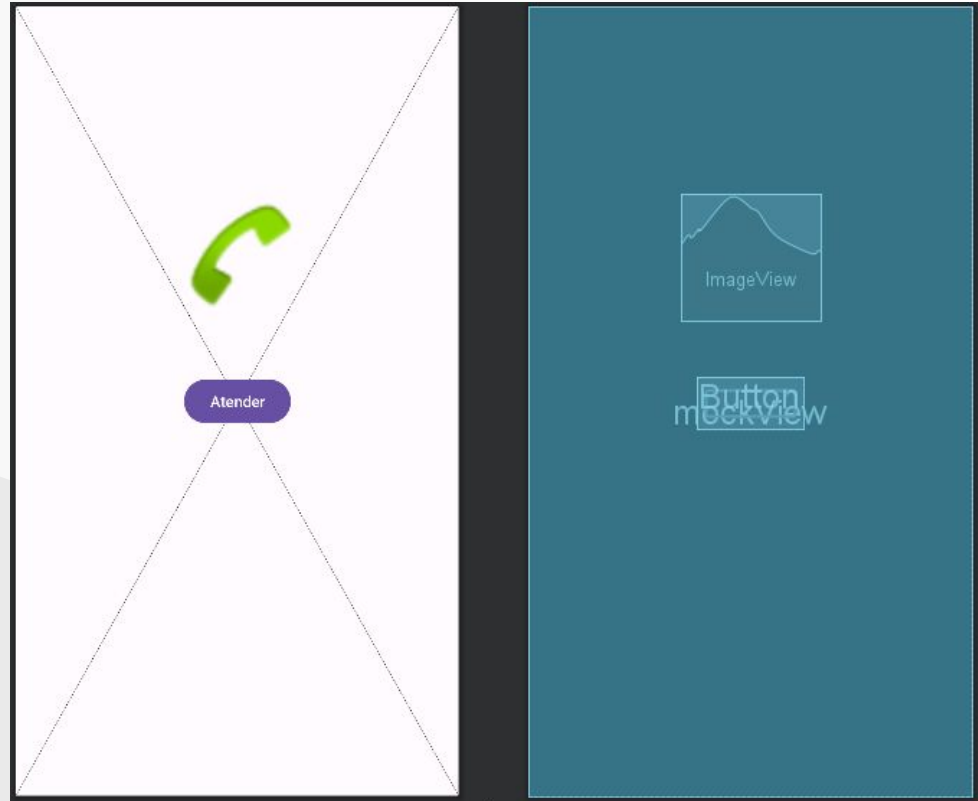
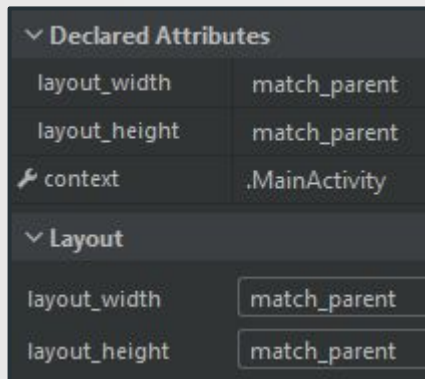
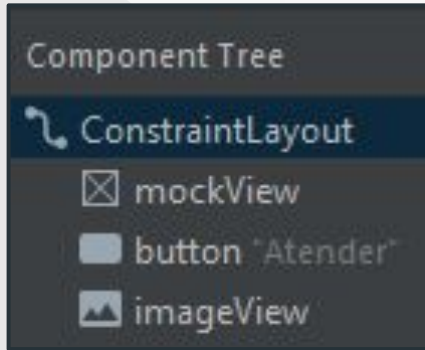
Em termos simples, imagine que você não deseja criar um design completo e original para o seu aplicativo, mas, em vez disso, deseja esboçar um design simplificado e experimentar diferentes opções de cores.

Nesse cenário, o MockView pode ser usado para criar esse esboço visual e testar várias combinações de cores.

Um pouco mais sobre MockView

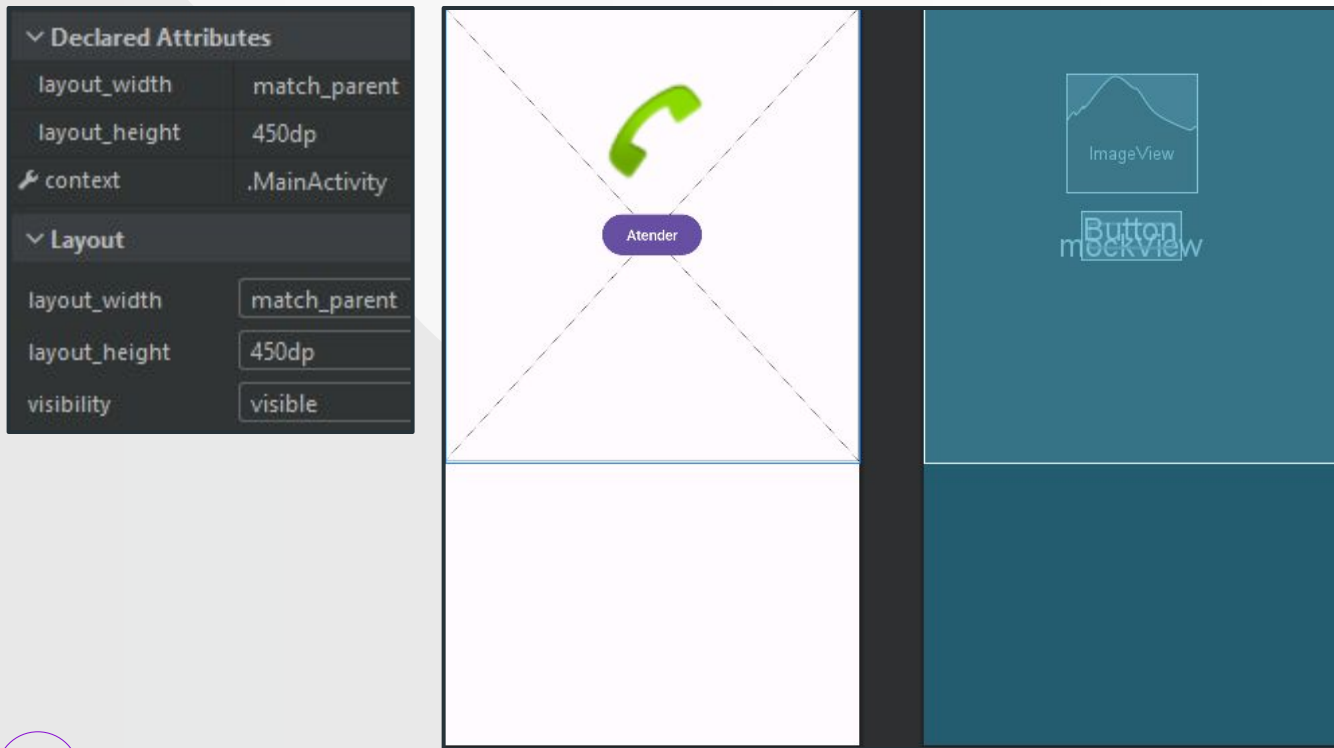
- Definir ID do MockView: Muitas vezes, precisamos do id da visualização para acessá-la no arquivo kotlin ou criar uma interface do usuário relativa a essa visualização no arquivo xml.
- Definir largura (Width) e Altura (Height) do MockView: Atributos para definir a largura e altura do MockView, ambos podem ser “**MATCH_PARENT**” ou “**WRAP_CONTENT**” ou qualquer **valor fixo** (como 20dp, 30dp etc.).
- Outras atributos e configurações como Padding (preenchimento), Margin (Margem) e Visibility (Visibilidade) também podem ser usados aqui.

Exemplo de Uso 1



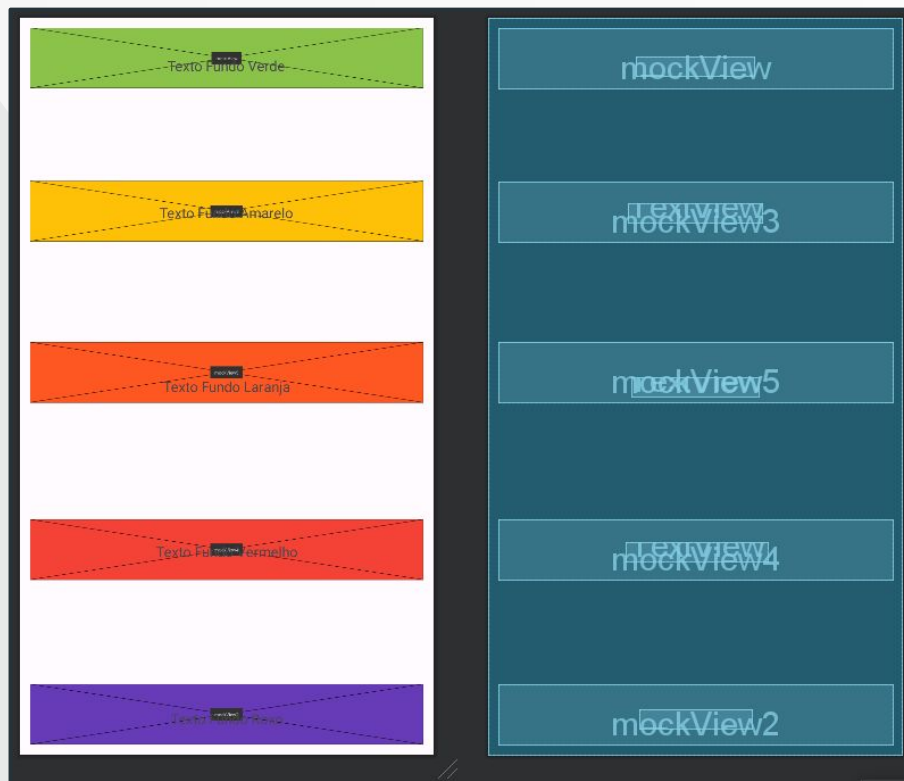
Exemplo de Uso 1

Ajustando um novo tamanho ao MockView, podemos notar que todo o seu conteúdo passa a ser alterado de mesmo modo.



Exemplo de Uso 2

Um segundo exemplo de uso para o MockView, pode ser utilizar o mesmo como molde de teste para variações de tamanhos, cores, fontes e layouts do seu projeto.





LAYERS

Layers

O Layer foi adicionado aos Helpers na versão **2.0**. Ele tem a capacidade de mover e girar um grupo de visualizações como se estivessem contidas em um `viewGroup`.

A classe Layer é uma extensão do **ConstraintHelper** que permite adicionar uma camada visual para manipular a posição e a rotação de um grupo de “interações”.



O que é Layer ?

Inicialmente podemos falar de dois principais eventos proporcionados e o método de uso da classe, que são:

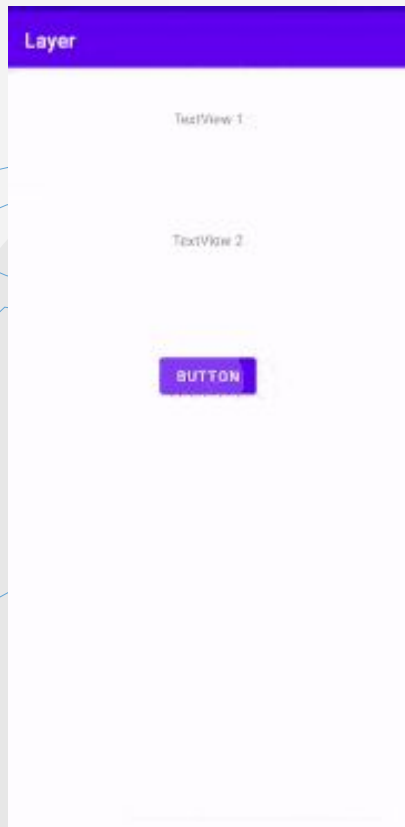
- **Extensão de ConstraintHelper:** é usada em juntamente com o ConstraintLayout para definir restrições entre as “interações”. Ela adiciona recursos específicos para manipular interações em grupo, como movimento e rotação.
- **Manipulação de Rotação:** Permite que você aplique rotações a um grupo de interações. Você pode usar o método `setRotation(float)` para girar todas as visualizações no grupo em torno de um centro comum.
- **Manipulação de Visibilidade Simples:** Podemos também usar para controle de visibilidade, semelhante a um `ViewGroup`, para mostrar ou ocultar um grupo de “visualizações”.

Exemplo de Aplicação e Atributos

Sobre Atributos:

- Sempre devemos definir um id de visualização para acessá-la no arquivo kotlin.
- Largura (Width): este atributo serve para definir a largura do Layer(camada). No código `“android:layout_width=” ””`.
- Altura (Height): este atributo serve para definir a altura do Layer. No código `“android:layout_height=” ””`.

Exemplo de Aplicação e Atributos



Aplicação:

Basicamente criamos layers virtuais a partir de visualizações referenciadas para que possamos construir animações / movimentos e aplicá-las ao mesmo tempo.

Aqui conseguimos com apenas um click no botão fazer o movimento/rotação de dois TextView, apenas com a interação do usuário.



05

FLOW

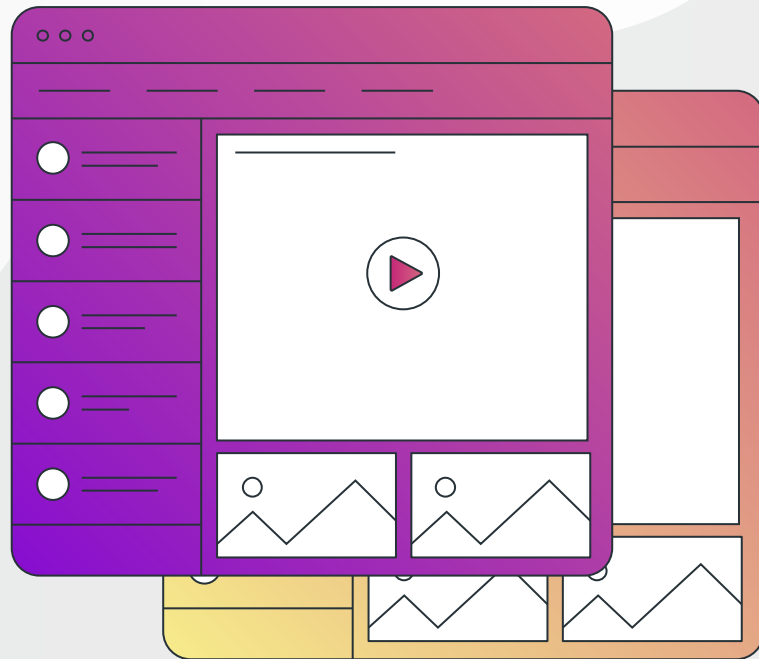





Flow

O Fluxo Virtual Layout é um recurso introduzido na versão 2.0 do Android Studio que permite o posicionamento flexível de widgets na interface do usuário.

Ele se baseia no conceito de uma "Cadeia" (Chain) virtual, que permite organizar widgets horizontal ou verticalmente com base em regras específicas.



Principais Eventos, Atributos e Configurações do Fluxo Virtual Layout

O Fluxo Virtual Layout oferece algumas maneiras de controlar como os widgets são organizados na tela:

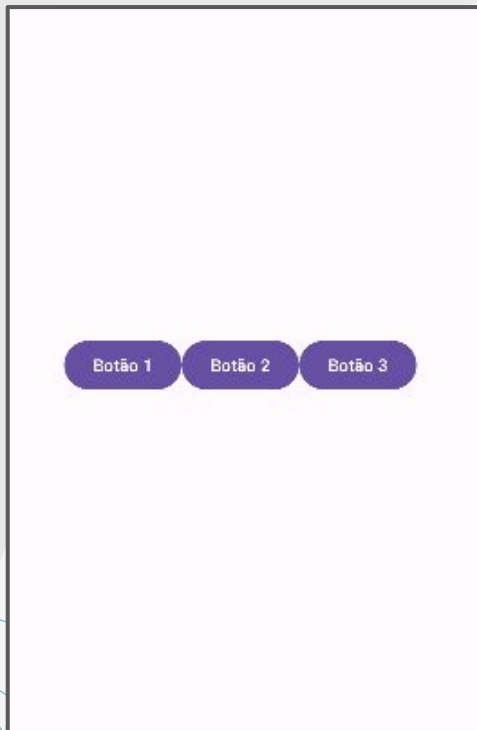
- Estilo da Cadeia (Chain Style): Isso define como os widgets na cadeia são distribuídos. Você pode escolher entre:
 - "spread": Distribui igualmente o espaço entre os widgets.
 - "spread_inside": Semelhante ao "spread," mas exclui as margens externas.
 - "packed": Os widgets ficam juntos, sem espaço extra entre eles.
 - "packed_inside": Semelhante ao "packed," mas exclui as margens externas.
- Compactar (Packed): Você pode definir se os widgets na cadeia devem ser compactados (agrupados) ou não, isso afeta como o espaço é distribuído.
- Largura (Width): Atribuir largura aos widgets ou seja a quantidade de espaço que cada widget na cadeia ocupa.
- Altura (Height): Atribuir altura aos widgets ou seja a quantidade de espaço que cada widget na cadeia ocupa.

Principais Eventos, Atributos e Configurações do Fluxo Virtual Layout

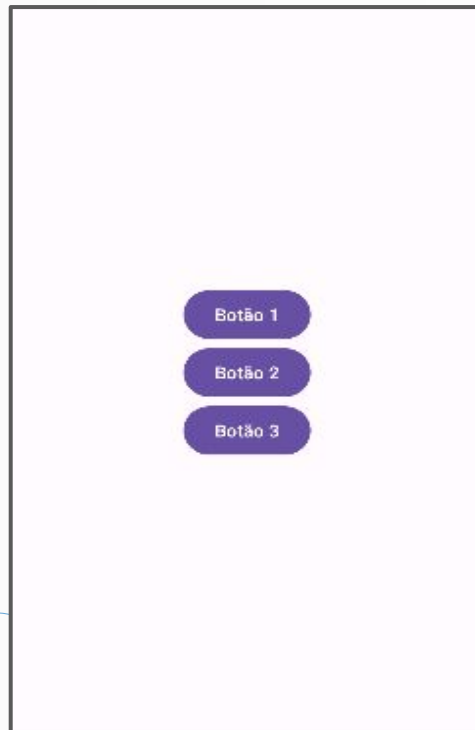
▼ Declared Attributes		+ -
layout_width	wrap_content	▼
layout_height	wrap_content	▼
layout_constraintHorizontal_chainStyle	spread	▼
layout_constraintVertical_chainStyle	spread	▼
layout_constraintBottom_toBottomOf	parent	▼
layout_constraintEnd_toEndOf	parent	▼
layout_constraintStart_toStartOf	parent	▼
layout_constraintTop_toTopOf	parent	▼
constraint_referenced_ids	button3,btn	
flow_wrapMode	chain	▼
id	id_Flow	
orientation	horizontal	▼

Exemplo de Uso

flow_wrapMode	chain
id	id_Flow
orientation	horizontal



flow_wrapMode	chain
id	id_Flow
orientation	vertical





**OBRIGADO PELA SUA
ATENÇÃO!**