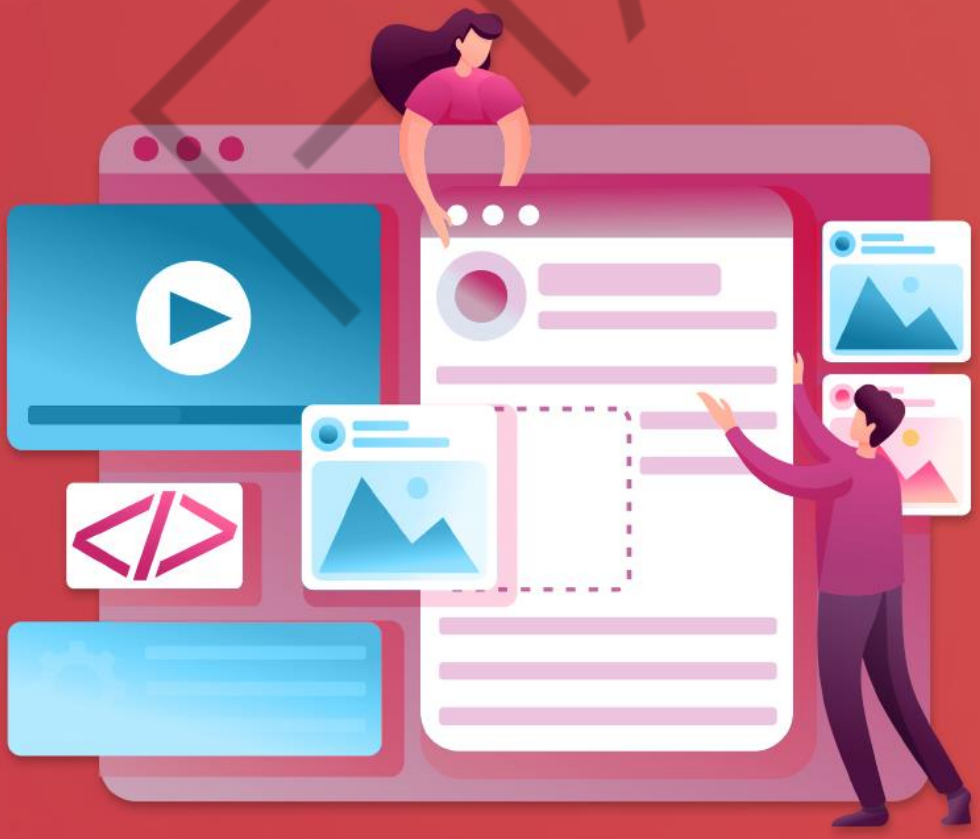
APP WORLD

LAYOUTS E COMPONENTES BÁSICOS NO JETPACK COMPOSE



5A

LISTA DE FIGURAS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

1 LAYOUTS E COMPONENTES BÁSICOS NO JETPACK

O Jetpack Compose tornou a atividade de criação das Interfaces de Usuário (IU) um trabalho extremamente fácil e produtivo. Um dos fundamentos que possibilitam isso são o sistema de layouts do Jetpack Compose. O Jetpack Compose fornece vários elementos de layout que você pode usar para criar IUs interessantes e intuitivas para o usuário.

Quando estamos construindo uma IU é necessário inserirmos vários componentes como textos, botões, caixas de seleção, etc. Se não fornecemos as instruções de como esses componentes dever ser organizados, o resultado não será como desejamos. Por exemplo, o efeito do código abaixo gera dois textos sobrepostos, já que não passamos as orientações necessárias para posicioná-los. Exemplo:

```
@Composable
fun BoxScreen() {
    Text(text = "FIAP")
    Text(text = "ON")
}
```

Código-fonte 1 – Inclusão de dois Texts em uma função Compose
Fonte: Elaborado pelo autor (2023)

O resultado do código acima deve ser parecido com o da figura “Componentes sem contêiner”:

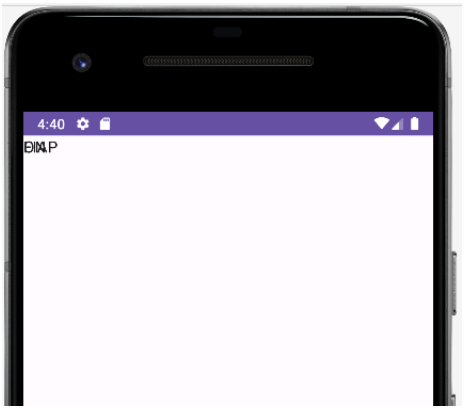


Figura 1 – Componentes sem o contêiner
Fonte: Elaborado pelo autor (2023)

No capítulo anterior, já tivemos uma amostra do sistema de layouts, com o uso dos *composables* "*Column*" e "*Row*". Neste capítulo, vamos explorá-los um pouco mais, além de outros componentes de layout que tornarão a IU muito mais profissional.

1.1 Box

O *composable* "*Box*" é utilizado para agrupar outros *composables,* como textos e botões dentro de uma área retangular. Podemos comparar o *Box* a uma DIV no HTML, ou seja, é um container de *composables*, mas que sabe como posicioná-los de acordo com as orientações do programador.

Para praticarmos o uso do *composable* "*Box*", crie um novo projeto "*Composable*" no Android Studio com o nome de "*BoxApp*". Assim que o projeto estiver concluído apague algumas instruções que não nos interessa agora. Ao concluir, o seu projeto deverá se parecer com o da figura "Estrutura inicial do projeto":

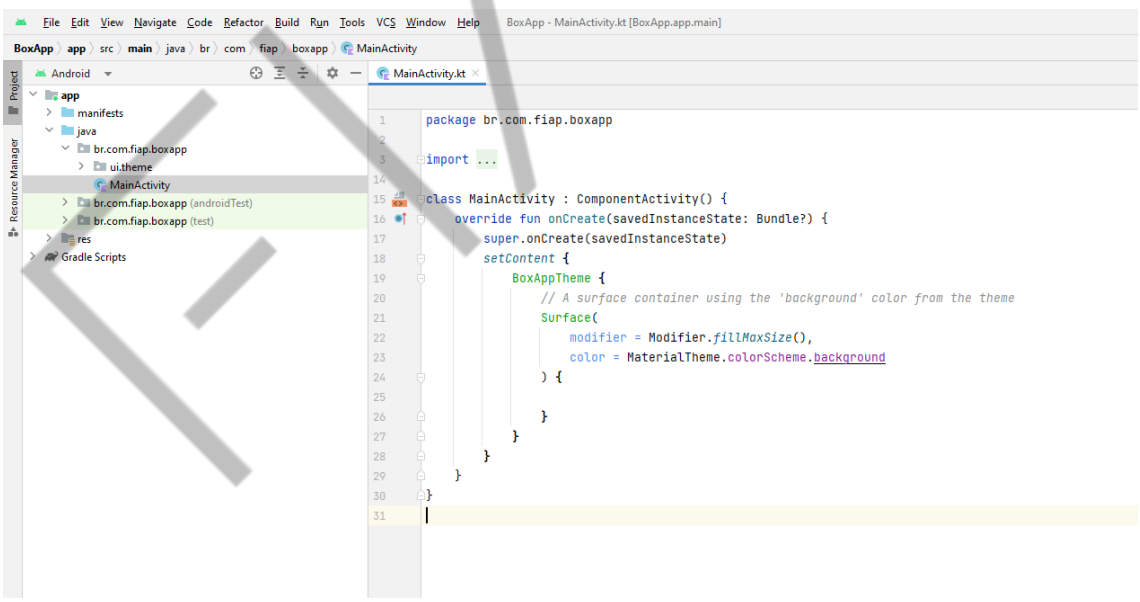


Figura 2 – Estrutura inicial do projeto
Fonte: Elaborado pelo autor (2023)

Vamos criar uma função de composição chamada “BoxScreen”. Vamos inserir o nome “FIAP” em nossa IU. A sua função deverá ser como o da listagem abaixo:

```
@Composable
fun BoxScreen() {
    Box {
        Text(text = "FIAP")
    }
}
```

Código-fonte 2 – Utilização da função Box.
Fonte: Elaborado pelo autor (2023)

Agora, no método “onCreate” da nossa classe, vamos chamar a função *composable* na função “setContent”. O seu código deverá se parecer com o da listagem abaixo:

```
package br.com.fiap.boxapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import br.com.fiap.boxapp.ui.theme.BoxAppTheme

class MainActivity : ComponentActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContent {
      BoxAppTheme {
        // A surface container using the 'background' color from the theme
        Surface(
          modifier = Modifier.fillMaxSize(),
          color = MaterialTheme.colorScheme.background
        ) {
          BoxScreen()
        }
      }
    }
  }
}

@Composable
fun BoxScreen() {
  Box {
    Text(text = "FIAP")
  }
}
```

Código-fonte 3 – Chamada para a função BoxScreen.
Fonte: Elaborado pelo autor (2023)

Execute a aplicação em um emulador, o resultado obtido deverá ser como o da figura “Função Box padrão”:

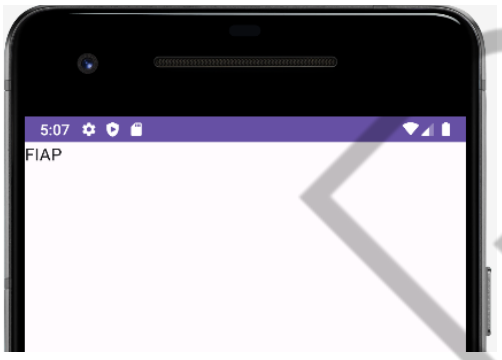


Figura 3 – Função Box padrão
Fonte: Elaborado pelo autor (2023)

Nosso texto ficou posicionado no canto superior esquerdo, ou “***TopStart***”, que significa “no topo ao início’, que é o padrão. Agora, imagine se quisermos posicionar o texto no centro da tela? Fácil, o *composable* “***Box***” possui um parâmetro “***contentAlignment***”, que nos permite alinhar o conteúdo em qualquer lado do “***Box***”. Portanto, vamos acrescentar esse parâmetro ao “***Box***” e ver o resultado. O código da função “***BoxScreen***” deverá se parecer com o código da listagem abaixo:

```
@Composable
fun BoxScreen() {
  Box(contentAlignment = Alignment.Center) {
    Text(text = "FIAP")
  }
}
```

Código-fonte 4 – Utilização do contentAlignment da função Box
Fonte: Elaborado pelo autor (2023)

O “***contentAlignment***” é um parâmetro da função “***Box***”, então, devemos passá-lo dentro dos parênteses da função. Para esse exemplo passamos o valor “***Alignment.Center***”, que é uma constante da Classe “***Alignment’*** que representa o alinhamento central dos componentes.

Rode a aplicação novamente, o resultado deve ser como o da figura “ContentAlignment”:

Figura 4 – ContentAlignment
Fonte: Elaborado pelo autor (2023)

Altere o valor do parâmetro “contentAlignment” para “***Alignment.TopEnd***”.

Rode a aplicação novamente, o resultado deverá ser parecido com o da figura “Alignment.TopEnd”:

Figura 5 – Alignment.TopEnd
Fonte: Elaborado pelo autor (2023)

Agora nosso texto está alinhado ao topo e do lado direito da nossa Box.

Como podemos perceber, o parâmetro “contentAlignment” permite alinhar o conteúdo em qualquer um dos lados da “Box”. Mas, e se tivermos outro componente? Vamos testar!

Vamos adicionar um botão à nossa Box e observar o resultado. O código da função “BoxScreen” deverá ser como o da listagem abaixo:

```
@Composable
fun BoxScreen() {
  Box(contentAlignment = Alignment.TopEnd) {
    Text(text = "FIAP")
    Button(onClick = { /*TODO*/ }) {
      Text(text = "Clique aqui")
    }
  }
}
```

Código-fonte 5 – Adição do componente Button.
Fonte: Elaborado pelo autor (2023)

Execute a aplicação no emulador e observe o resultado. Onde está o nosso texto?

Os componentes são empilhados no interior da Box, então o primeiro componente ficará na primeira “Camada”, o segundo na segunda camada, e assim por diante. O Resultado da aplicação deve ser parecido com o da figura “Componentes sobrepostos na Box”:

Figura 6 – Componentes sobrepostos na Box
Fonte: Elaborado pelo autor (2023)

Se observarmos, poderemos ver um pedacinho do texto atrás do botão. Vamos inverter a ordem de inclusão dos componentes, o código da função “BoxScreen” deverá ficar como na listagem abaixo:

```
@Composable
fun BoxScreen() {
  Box(contentAlignment = Alignment.TopEnd) {
    Button(onClick = { /*TODO*/ }) {
      Text(text = "Clique aqui")
    }
    Text(text = "FIAP")
  }
}
```

Código-fonte 6 – Invertendo a posição dos componentes
Fonte: Elaborado pelo autor (2023)

Rode a aplicação novamente. O resultado deve ser parecido com a figura **“**Ordem dos componentes invertidos na Box**”**:

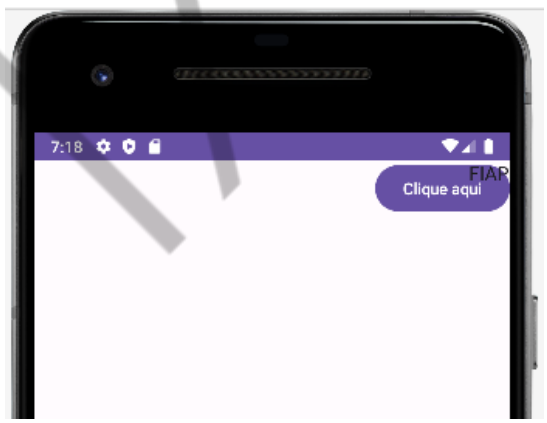


Figura 7 – Ordem dos componentes invertidos na Box
Fonte: Elaborado pelo autor (2023)

Note que agora o texto ficou na frente do botão.

1.2 Alinhando vários componentes na Box

Aprendemos que podemos alinhar o conteúdo de uma *Box* utilizando o parâmetro “***contentAlignment***”, mas todos os componentes dentro da *Box* serão

alinhados no mesmo lugar, um sobre o outro. E se quisermos que o texto fique no topo a esquerda e o botão na parte de baixo a direita?

Os *composables* dentro de uma *Box* estão sobre o escopo da *Box*, então, obedecem a este escopo. Mas, os *composables* possuem um parâmetro chamado "***modifier***", com este parâmetro podemos alterar diversas características do componente, inclusive a sua posição dentro da *Box*. E, quando alteramos, no *composable,* um parâmetro herdado do composable pai, este tem prioridade. Assim, podemos alinhar cada *composable* dentro de uma *Box* individualmente utilizando o seu modificador. Vamos lá!

Altere o código da função de composição "***BoxScreen***" para que fique de acordo com a listagem abaixo:

```
@Composable
fun BoxScreen() {
  Box(contentAlignment = Alignment.TopEnd) {
    Button(
      onClick = { /*TODO*/ },
      modifier = Modifier.align(Alignment.BottomEnd)
    ) {
      Text(text = "Clique aqui")
    }
    Text(
      text = "FIAP",
      modifier = Modifier.align(Alignment.TopStart)
    )
  }
}
```

Código-fonte 7 – Uso do modificador para alinhamento do componente.
Fonte: Elaborado pelo autor (2023)

Execute a aplicação novamente. O resultado deverá ser como o apresentado na figura **"**Alinhamento por componente**"**:

Figura 8 – Alinhamento por componente
Fonte: Elaborado pelo autor (2023)

Note que, mesmo que tenhamos mantido o "***contentAlignment***" da *Box*, o que foi utilizado, de fato, foi o "***align***" do modificador de cada *composable*. Então, neste caso, os filhos têm vontade própria.

Atenção: no Jetpack Compose o "***modifier***" é um dos principais recursos para alterar a aparência e o comportamento de um *composable*. Os modificadores nos permitem adicionar efeitos visuais, como tamanho, cor, espaçamento, dentre outras transformações nos elementos da IU.

1.3 Controle de posicionamento utilizando "offset"

Também é possível movimentar os componentes dentro da Box utilizando coordenadas x e y. Essa não é a melhor e mais eficiente forma de posicionar componentes, mas, às vezes é necessário para fazermos algum efeito visual.

Na figura **“**Diagrama de deslocamento com offset**”**, temos um botão posicionado nas coordenadas **x=30.dp** e **y=40.dp** e o seu modificador de alinhamento como “***TopStart***” dentro da Box.

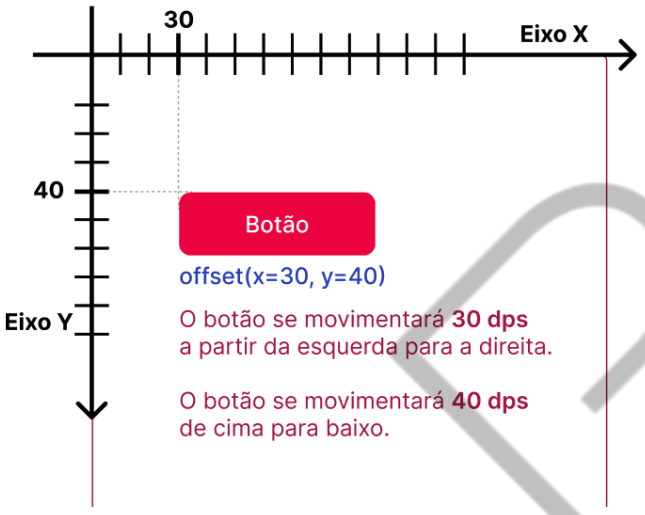


Figura 9 – Diagrama de deslocamento com offset
Fonte: Elaborado pelo autor (2023)

Vamos incluir na função “***BoxScreen***”, vamos incluir um novo botão e posicioná-lo em **x=150.dp** e **y=120.dp**. Seu código deverá se parecer com a listagem abaixo:

```
@Composable
fun BoxScreen() {
  Box(contentAlignment = Alignment.TopEnd) {
    Button(
      onClick = { /*TODO*/ },
      modifier = Modifier.align(Alignment.TopStart)
    ) {
      Text(text = "Clique aqui")
    }
    Text(
      text = "FIAP",
      modifier = Modifier.align(Alignment.TopStart)
    )
    Button(
      onClick = { /*TODO*/ },
      modifier = Modifier
        .align(Alignment.TopStart)
        .offset(x = 150.dp, y = 120.dp)
    ) {
      Text(text = "Outro botão")
    }
  }
}
```

Código-fonte 8 – Uso do offset.
Fonte: Elaborado pelo autor (2023)

Execute a aplicação no emulador. O resultado obtido deve ser parecido com o da figura **"**Deslocamento com offset**"**:

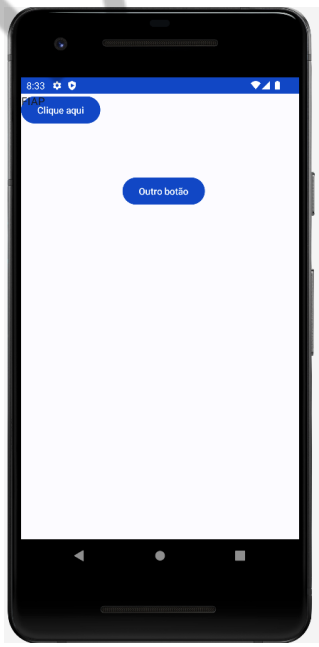


Figura 10 – Deslocamento com offset
Fonte: Elaborado pelo autor (2023)

É importante ressaltar que a posição x=0 e y=0 sempre levará em conta a posição inicial de composição do componente. Portanto, se você estiver utilizando "***BottomEnd***", a movimentação ocorrerá a partir deste ponto.

Altere o "***Modifier.align***" do segundo botão para "***BottomEnd***" e modifique o "***offset***" para x=20 e y=10. O seu código deverá se parecer como o da listagem abaixo:

```
@Composable
fun BoxScreen() {
  Box(contentAlignment = Alignment.TopEnd) {
    Button(
      onClick = { /*TODO*/ },
      modifier = Modifier
        .align(Alignment.TopStart)
    ) {
      Text(text = "Clique aqui")
    }
    Text(
      text = "FIAP",
      modifier = Modifier.align(Alignment.TopStart)
    )
    Button(
      onClick = { /*TODO*/ },
      modifier = Modifier
        .align(Alignment.BottomEnd)
        .offset(x = 20.dp, y = 10.dp)
    ) {
      Text(text = "Outro botão")
    }
  }
}
```

Código-fonte 9 – Ajuste de posicionamento com offset.
Fonte: Elaborado pelo autor (2023)

Execute a aplicação novamente, o resultado obtido deverá ser como o da figura **"**Offset excedendo limites da tela**"**:

Figura 11 – Offset excedendo limites da tela
Fonte: Elaborado pelo autor (2023)

Observe que o botão se movimentou em x e y levando em conta a sua posição inicial ficando parcialmente fora da tela tanto na horizontal quanto na vertical.

1.4 Column e Row

Os *composables* de layout mais utilizados são, com certeza, a “***Column***” e a “***Row***”, pois permitem que posicionemos os componentes de nossa IU lado a lado, na horizontal ou na vertical, na ordem em que são inseridos em nosso código, conforme exemplificado na figura **“**Layouts com Column e Row**”:**

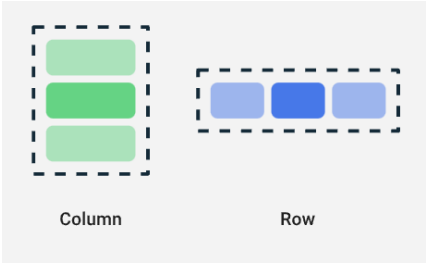


Figura 12 – *Layouts com Column e Row*
Fonte: Elaborado pelo autor (2023)

Crie um projeto *Jetpack Compose* no Android Studio chamado “***Column Row***”. Após a criação do projeto, apague todas as funções, exceto a classe “***MainActivity***”. Seu código deverá se parecer com o da figura **“**Estrutura inicial do projeto**”**:

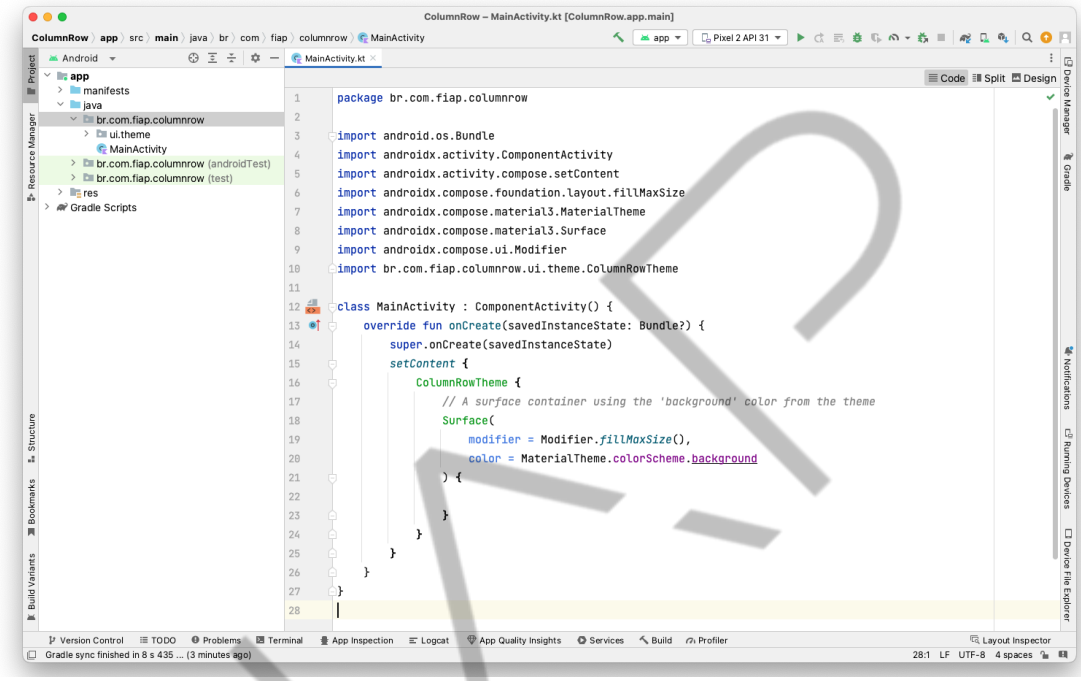


Figura 13 – Estrutura inicial do projeto
Fonte: Elaborado pelo autor (2023)

Vamos criar o layout da figura “Layout”:

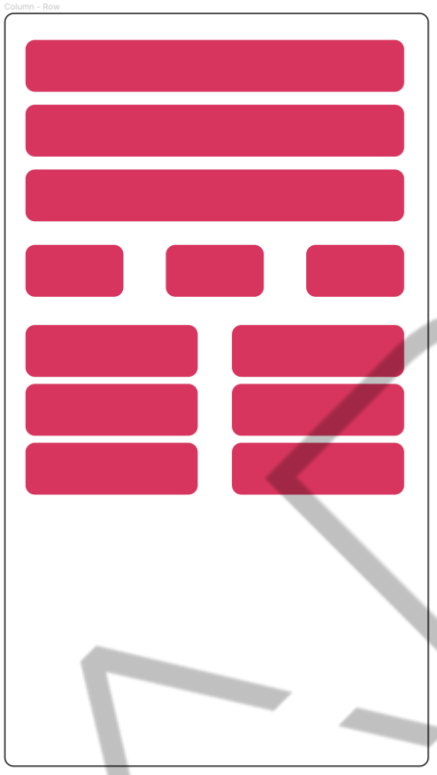


Figura 14 – Layout
Fonte: Elaborado pelo autor (2023)

Quando olhamos para um layout, precisamos analisá-lo de modo a “ver” a sua estrutura ou esqueleto. No layout da figura “Componentes estruturais”, identificamos os seguintes componentes estruturais da nossa IU representados pelas linhas pontilhadas.

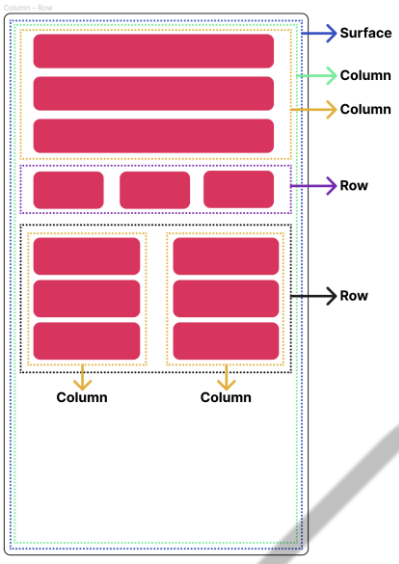


Figura 15 - Componentes estruturais
Fonte: Elaborado pelo autor (2023)

Agora podemos começar a construção da nossa IU pelos componentes estruturais e ir incluindo os outros componentes pouco a pouco. É como construir um brinquedo formado por blocos, vamos juntando os blocos e no final teremos o brinquedo montado.

Crie uma função de composição chamada "***ColomnRowScreen***". Vamos começar incluindo nossos componentes estruturais.

Em relação ao "*Surface*", que é o nosso primeiro componente estrutural, não precisaremos inseri-lo agora, pois ele já faz parte da nossa IU de forma padrão. Observe o código listado abaixo:

```
class MainActivity : ComponentActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContent {
      ColumnRowTheme {
        // A surface container using the 'background' color from the theme
        Surface(
          modifier = Modifier.fillMaxSize(),
          color = MaterialTheme.colorScheme.background
        ) {

        }
      }
    }
  }
}
```

Código-fonte 10 – Estrutura padrão o setContent com Surface.
Fonte: Elaborado pelo autor (2023)

No método “***setContent***” da função “***onCreate***”, da nossa *Activity*, o *Surface* é o primeiro *composable* e a função que desenhará o resto da IU será envolvida por ela. O *Surface* da nossa IU ocupa todo o tamanho da tela do dispositivo, isso se deve ao parâmetro modificador “***modifier = Modifier.fillMaxSize()***”, que por si só já é autoexplicativo. Esse parâmetro diz ao *Surface* que ele deve se encaixar ao tamanho máximo da tela.

Explicações dadas, vamos à construção da nossa função “***ColumnRowScreen***”.

Vamos começar pelos composable mais externos. O código da função deverá se parecer com o da listagem abaixo:

```
 @Composable
fun ColumnRowScreen() {
  // Column principal
  Column(modifier = Modifier.fillMaxSize()) {
    Column(modifier = Modifier.fillMaxWidth()) {

    }
    Row(modifier = Modifier.fillMaxWidth()) {

    }
    Row(modifier = Modifier.fillMaxWidth()) {

    }
  }
}
```

Código-fonte 11 – Estrutura básica com Column e Row
Fonte: Elaborado pelo autor (2023)

A primeira *Column* ocupará todo o tamanho da tela, por isso o seu modificador recebe o valor “***fillMaxSize***”.

Dentro desta primeira *Column* estamos inserindo uma *Column* que ocupará toda a largura do componente pai, que é a primeira *Column*. Sua altura será dinâmica, ou seja, vai expandir de acordo com o conteúdo.

Em seguida, inserimos duas “***Rows”***, que ficarão uma abaixo da outra.

A representação gráfica da estrutura da nossa IU até aqui, está conforme a figura “Estrutura do projeto”:

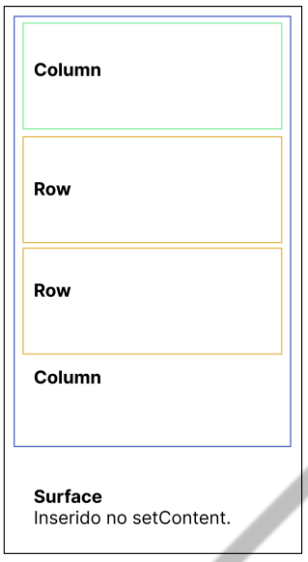


Figura 16 – Estrutura do projeto
Fonte: Elaborado pelo autor (2023)

Agora, no interior da última "***Row***", vamos inserir duas "***Columns***", que ficarão lado a lado, já que o seu componente pai é uma "***Row***". Nosso código final dever se parecer com o da listagem abaixo:

```
@Composable
fun ColumnRowScreen() {
  // Column principal
  Column(){
    Column() {
      // Aqui vai o conteúdo
    }
    Row() {
      // Aqui vai o conteúdo
    }
    Row(){
      Column() {
        // Aqui vai o conteúdo
      }
      Column() {
        // Aqui vai o conteúdo
      }

    }
  }
}
```

Código-fonte 12 – Estrutura de Column e Row aninhados.
Fonte: Elaborado pelo autor (2023)

1.5 Modifier

O “***Modifier***” é uma classe que nos permite modificar os estilos de um *composable*, como tamanho, cor, posicionamento, espaçamento, dentre outros. Com o “***Modifier***” podemos encadear vários modificadores para alterar a aparência de um *composable*, além do seu comportamento. As modificações são aplicadas na ordem em que você as declara, então, a ordem é importante.

Essa é uma pequena lista dos modificadores mais utilizados:

`padding()`: Adiciona um espaçamento interno ao redor do *composable*.

`size()`: Define o tamanho do *composable*.

`offset()`: Define a posição do composable dentro do seu contêiner pai.

`clickable()`: Torna o *composable* clicável e permite adicionar uma ação ao ser clicado.

`background()`: Define a cor de fundo do *composable*.

`fillMaxWidth()`, `fillMaxHeight()`: Faz com que o *composable* ocupe todo o espaço disponível no eixo horizontal ou vertical, respectivamente.

`fillMaxSize()`: Faz com que o *composable* ocupe todo o espaço disponível na tela.

`align()`: Alinha o *composable* dentro do seu contêiner pai.

`weight()`: Controla a distribuição do espaço disponível entre vários *composables* dentro de um contêiner.

Vamos aplicar alguns efeitos visuais em nossa IU para que possamos visualizar melhor a estrutura que criamos. Ao final, nossa IU deverá se parecer com a **figura “Efeitos visuais”**:

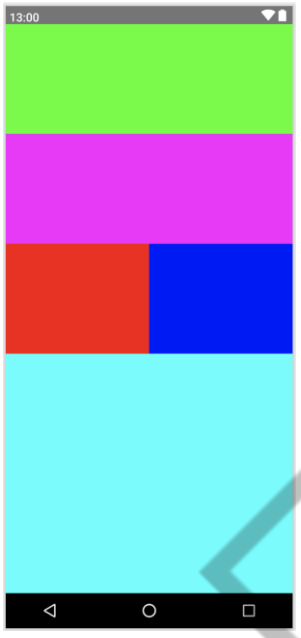


Figura 17 – Efeitos visuais
Fonte: Elaborado pelo autor (2023)

Na primeira “Column” vamos colocar uma cor de fundo (background) na cor ciano. Para isso adicione a seguinte instrução a esse composable:

```
@Composable
fun ColumnRowScreen() {
  // Column principal
  Column(modifier = Modifier
      .background(Color.Cyan)) {
    Column() {
      // Aqui vai o conteúdo
    }
    Row() {
      // Aqui vai o conteúdo
    }
    Row() {
      Column() {
        // Aqui vai o conteúdo
      }
      Column() {
        // Aqui vai o conteúdo
      }

    }
  }
}
```

Código-fonte 13 – Modificando o background de uma Column.
Fonte: Elaborado pelo autor (2023)

Antes de executarmos a aplicação, devemos chamar a função "***ColumnRowScreen***" na função "***setContent***" do método "***onCreate***" da *Activity*. Seu código deverá se parecer com o da listagem abaixo:

```
class MainActivity : ComponentActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContent {
      ColumnRowTheme {
        // A surface container using the 'background' color from the theme
        Surface(
          modifier = Modifier.fillMaxSize(),
          color = MaterialTheme.colorScheme.background
        ) {
          ColumnRowScreen()
        }
      }
    }
  }
}
```

Código-fonte 14 – Chamando a função ColumnScreen no arquivo MainActiviy.kt.
Fonte: Elaborado pelo autor (2023)

Execute a aplicação no emulador. O resultado deverá ser como o da figura **"**Preview da função ColumnScreen com background alterado**"**:

Figura 18 – Preview da função ColumnScreen com background alterado
Fonte: Elaborado pelo autor (2023)

Como você pode ver, toda a nossa tela está com o *background* ciano. Lembre-se de que a primeira *Column* tem como contêiner pai o *Surface*, que tem a seguinte declaração:

```
Surface(
  modifier = Modifier.fillMaxSize(),
  color = MaterialTheme.colorScheme.background
) {
```

Código-fonte 15 – Uso do modificador fillMaxSize do Surface padrão.
Fonte: Elaborado pelo autor (2023)

O *Surface* declara um modificador com o valor "***Modifier.fillMaxSize()***", que faz com que ele ocupe todo o tamanho da tela do dispositivo. Por padrão, os componentes filhos de um *Surface* herdam o seu tamanho, por isso que a *Column* está ocupando todo o tamanho da tela.

Vamos alterar o modificador para o seguinte:

```
Surface(
  modifier = Modifier.size(150.dp),
  color = MaterialTheme.colorScheme.background
) {
```

Código-fonte 16 – Uso do modificador size do Surface.
Fonte: Elaborado pelo autor (2023)

Rode a aplicação novamente, o resultado deverá ser como o da **figura "**Preview do Surface com tamanho alterado**"**:

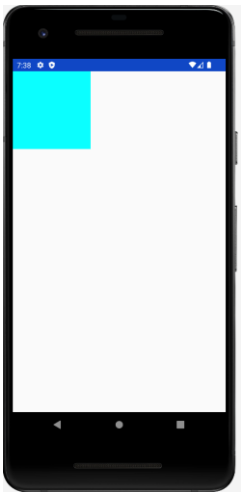


Figura 19 – Preview do Surface com tamanho alterado
Fonte: Elaborado pelo autor (2023)

O que vemos agora é um quadrado de tamanho 150x150, do lado superior esquerdo da tela. Esse quadrado é o *Surface* com a Column no seu interior. A cor ciano é da Column, já que não colocamos cor de background no Surface.

Vamos fazer outro ajuste no Surface, seu código deverá se parecer com a listagem abaixo:

```
Surface(
  modifier = Modifier.fillMaxWidth().height(200.dp),
  color = MaterialTheme.colorScheme.background
) {
```

Código-fonte 17 – Uso dos modificadores fillMaxWidth e height do Surface.
Fonte: Elaborado pelo autor (2023)

Execute a aplicação no emulador, o resultado deverá ser como o da figura **“**Surface com fillMaxWidth e height:

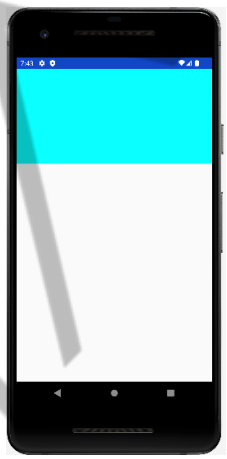


Figura 20 – Surface com fillMaxWidth e height
Fonte: Elaborado pelo autor (2023)

Agora, o Surface ocupa toda a extensão horizontal da tela. Isso foi declarado na função “***fillMaxWidth***”, ou seja, encaixar na largura (width) da tela e a altura foi declarada como 200dp na função “***height(200.dp)***”.

E os outros componentes? Onde estão?

No interior de uma *Column* ou *Row*, os componentes não possuem tamanho se não fornecemos esta configuração. Então, os componentes dentro desta *Column* ciano não estão visíveis. Vamos colocar uma cor de background na Column que está

em nossa Column de background ciano. Seu código deverá se parecer com o da listagem abaixo:

```
@Composable
fun ColumnRowScreen() {
  // Column principal
  Column(
    modifier = Modifier
      .background(Color.Cyan)
  ) {
    Column(modifier = Modifier.background(Color.Magenta)) {
      // Aqui vai o conteúdo
    }
    Row() {
      // Aqui vai o conteúdo
    }
    Row() {
      Column() {
        // Aqui vai o conteúdo
      }
      Column() {
        // Aqui vai o conteúdo
      }

    }
  }
}
```

Código-fonte 18 – Trocando a cor da Column para magenta.
Fonte: Elaborado pelo autor (2023)

Execute a aplicação novamente. Como esperado, você não deve ter notado nenhuma alteração. Sua aplicação deve se parecer com o da figura **“**Preview da Column com background magenta**”**:

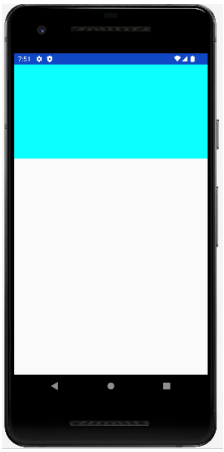


Figura 21 – Preview da Column com background magenta
Fonte: Elaborado pelo autor (2023)

Vamos acrescentar a função “size” ao Modifier. Seu código deverá se parecer com a listagem abaixo:

```
@Composable
fun ColumnRowScreen() {
  // Column principal
  Column(
    modifier = Modifier
      .background(Color.Cyan)
  ) {
    Column(modifier = Modifier
      .background(Color.Magenta)
      .size(100.dp)) {
      // Aqui vai o conteúdo
    }
    Row() {
      // Aqui vai o conteúdo
    }
    Row() {
      Column() {
        // Aqui vai o conteúdo
      }
      Column() {
        // Aqui vai o conteúdo
      }

    }
  }
}
```

Código-fonte 19 – Trocando a cor da Column magenta.
Fonte: Elaborado pelo autor (2023)

Execute a aplicação, o resultado deve ser parecido com o da figura “Visualização da Column magenta”:

Figura 22 – Visualização da Column magenta
Fonte: Elaborado pelo autor (2023)

Finalmente nossa *Column* tornou-se visível!

Agora que já conhecemos os modificadores de tamanho e cor dos nossos *composables* vamos colorir e redimensionar todos os *composables* da nossa IU.

Ao final, o código da função "***ColumnRowScreen***" deverá se parecer com a listagem abaixo:

```
@Composable
fun ColumnRowScreen() {
  // Column principal
  Column(
    modifier = Modifier
      .background(Color.Cyan)
  ) {
    Column(modifier = Modifier
      .background(Color.Magenta)
      .fillMaxWidth().height(150.dp)) {
      // Aqui vai o conteúdo
    }
    Row(modifier = Modifier
      .fillMaxWidth()
      .height(150.dp)
      .background(Color.Green)) {
      // Aqui vai o conteúdo
    }
    Row(modifier = Modifier
      .fillMaxWidth()
      .height(150.dp)
      .background(Color.Yellow)) {
      Column(modifier = Modifier
        .fillMaxWidth()
        .height(100.dp)
        .padding(8.dp)
        .background(Color.Red)
        .weight(0.3f)) {
        // Aqui vai o conteúdo
      }
      Column(modifier = Modifier
        .fillMaxWidth()
        .height(100.dp)
        .padding(8.dp)
        .background(Color.Blue)
        .weight(0.7f)) {
        // Aqui vai o conteúdo
      }
    }
  }
}
```

Código-fonte 20 – Código final da função ColumnRowScreen
Fonte: Elaborado pelo autor (2023)

Execute novamente o aplicativo no emulador. Sua IU deve se parecer com o da figura **“**Função ColumnRowScreen modificada**”**:

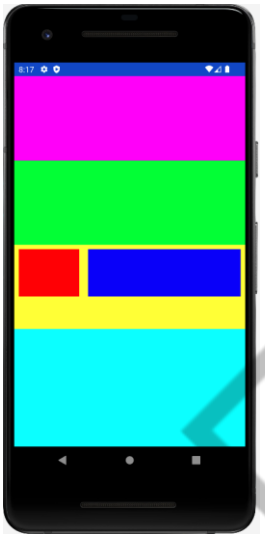


Figura 23 – Função *ColumnRowScreen* modificada
Fonte: Elaborado pelo autor (2023)

Você consegue ver todos os componentes da nossa IU?

Vamos a algumas explicações importantes:

A primeira *Column*, de fundo ciano é o nosso contêiner principal. Ele posiciona todos os componentes internos de forma empilhada na vertical.

O componente com fundo magenta é uma *Column*, ainda não temos nada dentro dela, mas se tivéssemos tudo estaria empilhado verticalmente.

O que vemos na **figura “Função ColumnRowScreen”,** com fundo verde, é uma *Row*, ainda não temos nada dentro dela, mas se tivéssemos tudo estaria alinhado lado a lado na horizontal.

O componente de fundo amarelo também é uma *Row*, e temos dentro dela duas *Columns*, uma com fundo vermelho e outra com fundo azul. Ambas estão lado a lado, já que é assim que as *Rows* posicionam seus componentes internos.

A *Column* vermelha e a *Column* azul estão com tamanhos diferentes, percebeu? A vermelha ocupa uma parte menor da *Row* amarela. Isso se deve a função “***weight(0.3f)***” que adicionamos na **linha 28** da listagem de código acima. O valor “**0.3f**” indica que a *Column* vermelha deverá ocupar 30% da *Row*. A *Column* azul

configuramos para "***weight(0.7f)***", ou seja, ocupará os outros 70% da *Row* amarela. O *Weight,* ou peso traduzindo para o português, representa a fração que o componente ocupará no interior de outro componente.

Também observamos que as Columns vermelha e azul, tem um espaço ao seu redor, isso foi feito utilizando a função "***padding(8.dp)***" nas **linhas 26** e **34** do modificador de cada uma delas. O *padding* determina um espaçamento ao redor do componente, é como se fosse uma margem.

1.6 Alinhamento e arranjo de Colunas

Quando trabalhamos com Column e Row, também é possível descrevermos o posicionamento dos componentes internos. O posicionamento pode ser feito utilizando o **Alinhamento** e o **Arranjo**. Vamos adicionar alguns botões dentro das Columns e Rows para entendermos como isso funciona. Vamos lá!

Na *Column* magenta, vamos adicionar 4 botões. As alterações para o código desta *Column* deverão se parecer com a listagem abaixo:

```
Column(modifier = Modifier
  .background(Color.Magenta)
  .fillMaxWidth()
  .height(150.dp)) {
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 01")
  }
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 01")
  }
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 01")
  }
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 01")
  }
}
```

Código-fonte 21 – Adição de botões na Column magenta
Fonte: Elaborado pelo autor (2023)

O "***Button***" é um *composable* que utilizamos para interagir com o usuário. Basicamente este componente possui um texto e uma função "***onClick***", que executará alguma instrução quando for pressionado. Para definição do texto do botão utilizamos o *composable* "***Text***" juntamente com o parâmetro "***text***" que exibirá o texto do botão.

Importante: Os *composables* podem utilizar outros *composables* para serem criados. O botão é um exemplo disso. Utilizamos o *Text* para incluir o texto no botão. Em tese, é possível compor um *composable* com qualquer outro *composable*. O limite é a criatividade e necessidade do desenvolvedor.

Execute a sua aplicação no emulador. O resultado deverá ser algo parecido com a figura "Preview dos botões na função ColumnRowScreen":

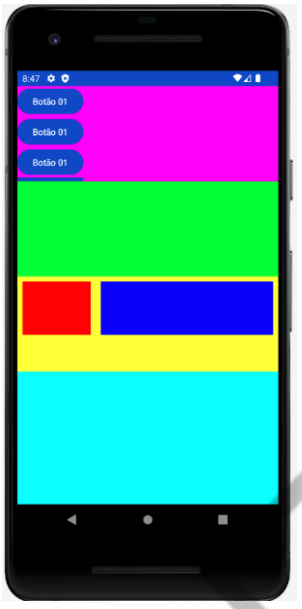


Figura 24 – Preview dos botões na função ColumnRowScreen
Fonte: Elaborado pelo autor (2023)

A Column magenta possui a largura da tela do dispositivo, que foi declarado com a função modificadora “***fillMaxWidth()***”. Possui uma altura de 150dp, que foi declarada com a função modificadora “***height(150.dp)***”. O problema disso é que o quarto botão não está sendo visível, ou apenas parte dele é. Isso se deve ao fato de a altura estar fixada e não irá expandir se adaptando ao seu conteúdo. Para resolvermos esse problema, vamos excluir a função height do modificador, assim, a altura ficará dinâmica e se ajustará ao conteúdo da Column. Sua alteração deverá se parecer com a listagem abaixo:

```
Column(
  modifier = Modifier
    .background(Color.Magenta)
    .fillMaxWidth()
) {
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 01")
  }
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 01")
  }
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 01")
  }
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 01")
  }
}
```

Código-fonte 22 – Exclusão do modificador height da Column magenta
Fonte: Elaborado pelo autor (2023)

Como resultado dessa alteração, sua IU deverá se parecer com a figura “Preview dos botões na Column magenta”:

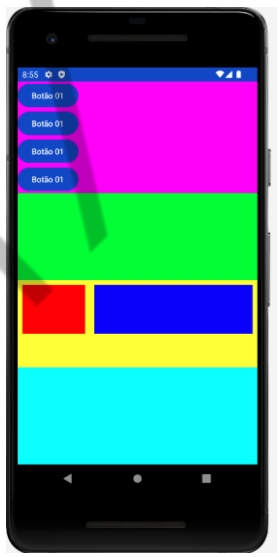


Figura 25 – Preview dos botões na Column magenta
Fonte: Elaborado pelo autor (2023)

Note que agora todos os botões estão visíveis. Se adicionarmos mais componentes nesta *Column* a sua altura sempre será ajustada para exibir todo o seu conteúdo.

Os botões estão alinhados no lado inicial da *Column*, mas e se quisermos posicioná-los ao centro ou ao final da *Column*? Podemos fazer isso utilizando o parâmetro "***horizontalAlignment***" da *Column*. Como possíveis valores para esse parâmetro podemos utilizar os seguintes:

CenterHorizontally: Alinha o conteúdo no centro horizontal da *Column*

End: Alinha o conteúdo do lado final/direito da *Column*.

Start: Alinha o conteúdo do lado inicial/esquerdo da *Column*.

Altere o código da *Column* magenta para que fique de acordo com a listagem abaixo:

```
Column(
  horizontalAlignment = Alignment.CenterHorizontally,
  modifier = Modifier
    .background(Color.Magenta)
    .fillMaxWidth()
) {
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 01")
  }
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 01")
  }
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 01")
  }
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 01")
  }
}
```

Código-fonte 23 – Implementação do alinhamento horizontal centralizado na Column magenta
Fonte: Elaborado pelo autor (2023)

Execute a aplicação. O resultado deverá se parecer com o da figura **"**Parâmetro horizontalAlignment.CenterHorizontally**"**:

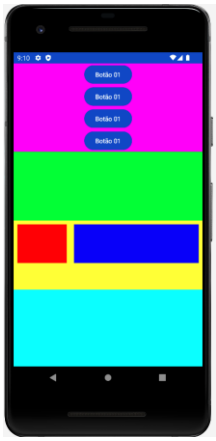


Figura 26 – Parâmetro horizontalAlignment.CenterHorizontally
Fonte: Elaborado pelo autor (2023)

Vamos alterar o valor do horizontalAlignment para “*End*”. Execute a aplicação novamente. O resultado deverá ser como o da figura **“**Parâmetro “horizontalAlignment.End”:

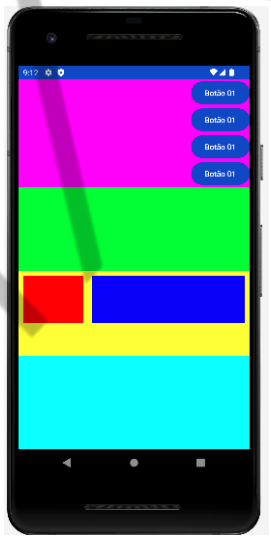


Figura 27 – Parâmetro horizontalAlignment.End
Fonte: Elaborado pelo autor (2023)

Removendo-se o parâmetro “***horizontalAlignment***”, ou trocando o valor para “***Start***”, o conteúdo será alinhado no lado inicial/esquerdo da tela. Experimente!

Além do alinhamento horizontal da *Column*, também podemos alterar o arranjo vertical do conteúdo de uma *Column* utilizando o parâmetro “***verticalArrangement***”.

O arranjo diz respeito a forma como os componentes internos de uma *Column* ficarão dispostos verticalmente.

O posicionamento do conteúdo de uma *Column* geralmente utiliza a combinação de "***horizontalAlignment***" e "***verticalArrangement***", deste modo podemos posicionar os componentes em qualquer lugar no interior da *Column*.

Alguns dos valores possíveis para o "***verticalArrangement***" são os seguintes:

Top: Alinha o conteúdo na parte superior da *Column*.

Bottom: Alinha todo o conteúdo na parte inferior da *Column*.

Center: Alinha todo o conteúdo no centro vertical da *Column*

SpaceAround: Espalha verticalmente todo o conteúdo da *Column*, com espaço uniforme entre eles e mantendo um espaço uniforme no início e no final da coluna.

SpaceBetween: Espalha verticalmente todo o conteúdo da *Column*, com espaço uniforme entre eles, mas não mantém espaço extra no início e no final da coluna.

SpaceEvenly: Espalha verticalmente todo o conteúdo da *Column*, com espaço uniforme entre eles. O mesmo espaço será colocado no início e no fim da coluna.

Para percebermos o arranjo vertical da *Column*, vamos aumentar a altura da *Column* de modo que os botões tenham mais espaço vertical. Faça as alterações necessárias no seu código de modo que fique parecido com a listagem abaixo:

```
Column(
  horizontalAlignment = Alignment.End,
  modifier = Modifier
    .background(Color.Magenta)
    .fillMaxWidth()
    .height(300.dp)
) {
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 01")
  }
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 01")
  }
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 01")
  }
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 01")
  }
}
```

Código-fonte 24 – Mudando a altura da Column magenta
Fonte: Elaborado pelo autor (2023)

Execute a aplicação no emulador. A sua IU deve se parecer com a figura **“**Column com altura modificada**”**:

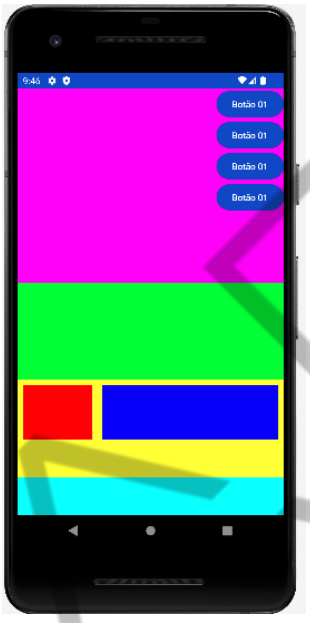


Figura 28 – Column com altura modificada
Fonte: Elaborado pelo autor (2023)

Note que a *Column* tem mais espaço vertical do que o conteúdo, assim, percebemos o espaço que sobra na parte inferior da *Column*. Vamos posicionar todos os botões na parte inferior da *Column*. Para isso o seu código deverá ficar de acordo com a listagem abaixo:

```
Column(
  horizontalAlignment = Alignment.End,
  verticalArrangement = Arrangement.Bottom,
  modifier = Modifier
    .background(Color.Magenta)
    .fillMaxWidth().height(300.dp)
) {
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 01")
  }
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 01")
  }
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 01")
  }
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 01")
  }
}
```

Código-fonte 25 – Implementação do arranjo vertical da Column magenta.
Fonte: Elaborado pelo autor (2023)

Ao executar a aplicação no emulador, sua IU deve se parecer com a figura “Posicionamento dos botões na parte inferior da Column”:

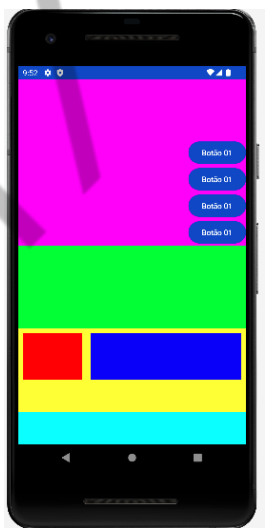


Figura 29 - Posicionamento dos botões na parte inferior da *Column*
Fonte: Elaborado pelo autor (2023)

Todo o conteúdo da Column foi alinhado horizontalmente ao lado final/direito da Column e posicionado na parte inferior.

Agora vamos posicionar todos os botões ao centro da Column e com arranjo vertical uniforme. Após as alterações o código deve se parecer com a listagem abaixo:

```
Column(
  horizontalAlignment = Alignment.CenterHorizontally,
  verticalArrangement = Arrangement.SpaceEvenly,
  modifier = Modifier
    .background(Color.Magenta)
    .fillMaxWidth().height(300.dp)
) {
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 01")
  }
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 01")
  }
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 01")
  }
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 01")
  }
}
```

Código-fonte 26 – Posicionando o conteúdo no centro da Column magenta.
Fonte: Elaborado pelo autor (2023)

O resultado esperado para a IU deve se parecer com a figura “Arranjo vertical SpaceEvenly”:

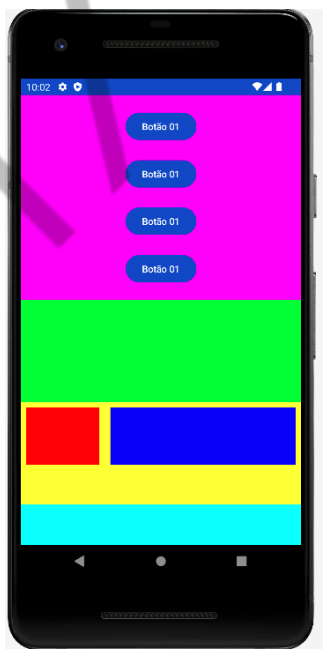


Figura 30 – Arranjo vertical SpaceEvenly
Fonte: Elaborado pelo autor (2023)

O “SpaceEvenly” distribuiu todos os botões de forma uniforme na coluna, mantendo o mesmo espaçamento entre os botões e ao início e fim da coluna.

Experimente outros valores para “***verticalArrangement***” e “***horizontalAlignment***”. Seja curioso, explore!

1.8 Alinhamento e Arranjo em Linhas

O alinhamento e arranjo de conteúdo nas Rows é bastante similar ao que aprendemos com as Columns. A diferença é que enquanto nas colunas fazemos alinhamento horizontal, nas linhas fazemos alinhamento vertical. O mesmo ocorre com o arranjo, nas colunas fazemos arranjo vertical enquanto nas linhas fazemos arranjo horizontal.

Vamos acrescentar dois botões na Row verde. Seu código deverá ficar de acordo com a listagem abaixo:

```
Row(
  modifier = Modifier
    .fillMaxWidth()
    .height(150.dp)
    .background(Color.Green)
) {
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 02")
  }
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 02")
  }
}
```

Código-fonte 27 – Adicionando botões em uma Row
Fonte: Elaborado pelo autor (2023)

Rode o aplicativo no emulador, seu código deverá se parecer com a **figura “**Alinhamento e arranjo padrão da Row**”**:

Figura 31 - Alinhamento e arranjo padrão da Row
Fonte: Elaborado pelo autor (2023)

O posicionamento padrão no interior de uma Row é no lado inicial/esquerdo e parte superior da linha.

Vamos acrescentar o parâmetro "***verticalAlignment***" com o valor "***CenterVertically***". Seu código deverá se parecer com a listagem abaixo:

```
Row(verticalAlignment = Alignment.CenterVertically,
  modifier = Modifier
    .fillMaxWidth()
    .height(150.dp)
    .background(Color.Green)
) {
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 02")
  }
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 02")
  }
}
```

Código-fonte 28 – Uso do alinhamento vertical em uma Row
Fonte: Elaborado pelo autor (2023)

Ao executar a aplicação no emulador, a IU deverá se parecer com a figura **"**Parâmetro VerticalAlignment.CenterVertically**"**:

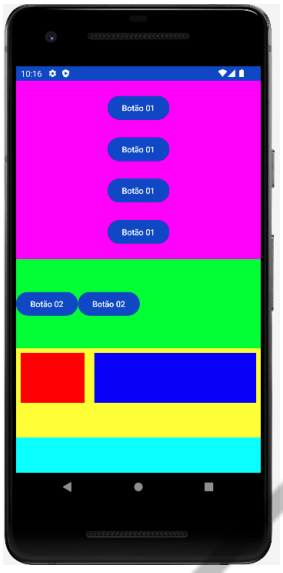


Figura 32 - Parâmetro "***verticalAlignment.CenterVertically***"
Fonte: Elaborado pelo autor (2023)

Como esperado, os botões foram posicionados no centro vertical da linha.

Vamos acrescentar o parâmetro "***horizontalArrangement***" com o valor "***SpaceBetween***". Após as alterações o seu código deve se parecer com a listagem abaixo:

```
Row(
  verticalAlignment = Alignment.CenterVertically,
  horizontalArrangement = Arrangement.SpaceBetween,
  modifier = Modifier
    .fillMaxWidth()
    .height(150.dp)
    .background(Color.Green)
) {
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 02")
  }
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 02")
  }
}
```

Código-fonte 29 – Uso do arranjo horizontal
Fonte: Elaborado pelo autor (2023)

Rode novamente o aplicativo. O resultado esperado deverá ser como o da **figura "**Preview da função SpaceBetween**"**:

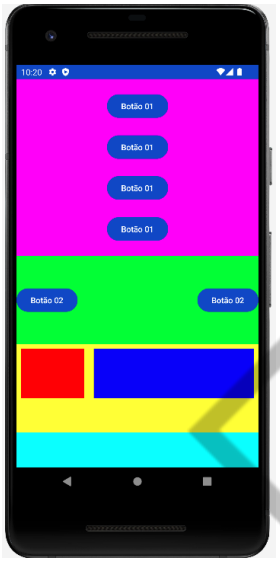


Figura 33 – Preview da função "***SpaceBetween***"
Fonte: Elaborado pelo autor (2023)

O "***SpaceBetween***" distribui os botões de modo a ocuparem todo o espaço da linha, mas sem manter os espaçamentos antes e depois da linha. Vamos substituir por "***SpaceAround***" para vermos a diferença. Seu código deverá se parecer com a listagem abaixo:

```
Row(
  verticalAlignment = Alignment.CenterVertically,
  horizontalArrangement = Arrangement.SpaceAround,
  modifier = Modifier
    .fillMaxWidth()
    .height(150.dp)
    .background(Color.Green)
) {
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 02")
  }
  Button(onClick = { /*TODO*/ }) {
    Text(text = "Botão 02")
  }
}
```

Código-fonte 30 – Implementação do arranjo SpaceAround.
Fonte: Elaborado pelo autor (2023)

Vamos executar nosso aplicativo. O resultado esperado deve ser parecido com o da figura "Preview da função SpaceAround":

Figura 34 – Preview da função *SpaceAround*”
Fonte: Elaborado pelo autor (2023)

Agora os botões foram distribuídos de modo a possuírem um espaçamento no início e no fim da linha.

Os mesmos valores utilizados na Column são válidos também para a Row, observando-se, claro, a orientação de cada componente.

CONCLUSÃO

Neste capítulo aprendemos sobre posicionamento de componentes utilizando *Columns* e *Rows*, que são componentes muito utilizados para a construção de uma IU no Android. Aprender e dominar esses dois *composables* tornarão a sua vida como desenvolvedor Android muito mais fácil.

Aprendemos também sobre o alinhamento e arranjo de componentes no interior das *Columns* e *Rows*. Essas são habilidades que permitirão um controle total sobre o posicionamento de elementos na IU e consequentemente permitirá que você crie IU cada vez mais ricas e interessantes ao usuário.

Agora que já dominamos o posicionamento de componentes na IU, estamos prontos para começar a explorar novos componentes no Android.

Estamos só começando a viagem. Até a próxima jornada! 😀

REFERÊNCIAS

ANDROID. **Conceitos básicos de layout do compose**. 2023. Disponível em: <https://developer.android.com/jetpack/compose/layouts/basics?hl=pt-br>. Acesso em: 29 jun. 2023.