FIAP ON

APP WORLD

INTRODUÇÃO AO JETPACK COMPOSE

4A

LISTA DE FIGURAS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

1 INTRODUÇÃO AO JETPACK COMPOSE

O Jetpack Compose é o *kit* de ferramentas oficial do Google para o desenvolvimento de aplicativos de forma nativa para dispositivos Android. Segundo o próprio Google, o Jetpack Compose simplifica e acelera o desenvolvimento de Interface do Usuário (IU) no Android.

Atualmente grandes empresas já aderiram a esta nova forma de construir aplicativos Android, dentre elas podemos citar o Google Play, Airbnb, Dropbox, Twitter, Booking, Adidas, Shopee, dentre outras.

O que o Jetpack Compose tem de tão especial que chamou a atenção de tantas empresas importantes? Vamos descobrir agora!

1.1 O que é o Jetpack Compose?

O Jetpack Compose é um framework para IU declarativa, que foi apresentado pela primeira vez à comunidade de desenvolvedores Android no Google I/O 2019, que é um evento utilizado pelo Google para falar sobre os novos recursos, ferramentas, tecnologias e tendências do universo Google, que também engloba o universo Android.

O Jetpack Compose é um kit de ferramentas, também chamado de toolkit, que nos entrega todos os recursos necessários para construir aplicativos Android de forma mais rápida, mais eficiente e escrevendo menos código.

1.2 Composables

Ao construir uma interface para Android, utilizando o Jetpack Compose, nós "quebramos" essa interface em pequenos “pedaços”, que depois são combinados e reutilizados de modo a criar interfaces mais complexas. Esses pequenos componentes são chamados de "*composable*", que é uma função em Kotlin que define a IU de um componente específico da tela. O *composable* é responsável por descrever como um elemento de tela dever ser renderizado e se comportar. Um *composable* pode ser construído através de outros *composables*.

Quando criamos um *composable*, nós declaramos os elementos visuais, como textos, botões, imagens e listas, além de definir toda a lógica de interação e a aparência do componente.

Uma das grandes vantagens dos *composables* é a possibilidade de reutilizar componentes, ou seja, nós criamos os *composables* mais genéricos que serão utilizados em diferentes partes da interface da aplicação.

O Jetpack Compose possui uma vasta gama de *composables* previamente construídos, que podemos utilizar para criar a interface da nossa aplicação. Praticamente todos os componentes de interface mais comuns em uma aplicação Android já estão disponíveis e prontas para serem utilizadas. Sensacional!

1.3 State

Uma das maiores dificuldades no desenvolvimento Android tradicional era controlar o estado da IU. O estado representa os dados que podem ser modificados e que afetam a aparência ou o comportamento dos componentes da interface.

Na abordagem tradicional fazemos a manipulação direta dos componentes de tela, ou seja, se um dado é alterado e este exige que um texto mude a cor, é necessário fazermos essa mudança de forma imperativa, e isso vai ocorrer com cada componente da IU. Quanto mais componentes de IU tivermos maior será a complexidade em manter a aparência atualizada e, consequentemente, maior será a possibilidade de existência de bugs.

No Jetpack Compose o estado é declarativo e reativo. Isso quer dizer que nós definimos o estado inicial dos componentes e qualquer mudança neste estado inicial resultará em uma atualização total da IU de forma automática.

O conceito de *state* trouxe uma grande vantagem para o desenvolvedor Android, que agora foca seu trabalho no que é realmente importante, como a lógica da IU e não mais na manipulação direta dos componentes de tela.

1.4 O novo e o antigo

Antes do Jetpack Compose as *views*, que eram os componentes de tela do nosso aplicativo Android, eram construídas através de XML, era necessário um arquivo separado para “desenhar” a tela.

Para alterar o estado das *views* era preciso obter, em Kotlin ou Java, a referência de cada *view* que deveria ser alterada. Isso gerava bastante código e aumentava a complexidade.

Com o Jetpack Compose tudo fica em um único arquivo, onde você descreve, em Kotlin, o que a sua IU deve conter e o Compose faz o resto.

Para termos uma ideia melhor sobre as diferenças entre construir componentes da maneira tradicional e com Jetpack Compose, vamos observar a implementação do componente representado na figura “Componente de busca em uma IU Android” com XML e *Composables*.

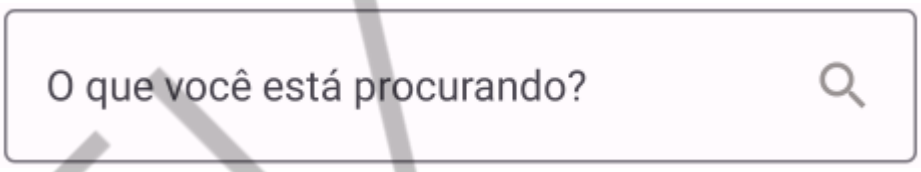

Figura 1 - Componente de busca em uma IU Android
Fonte: Elaborado pelo autor (2023)

Implementação com XML:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
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"
    tools:context=".MainActivity">

    <com.google.android.material.textfield.TextInputLayout
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="32dp"
        android:layout_marginTop="32dp"
        android:layout_marginEnd="32dp"
        android:hint="O que você está procurando?"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <com.google.android.material.textfield.TextInputEditText
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:drawableEnd="@drawable/baseline_search_24" />
    </com.google.android.material.textfield.TextInputLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Código-fonte 1 – Exemplo de código XML
Fonte: Elaborado pelo autor (2023)

Implementação com Jetpack Compose:

```
@Composable
fun SearchField() {
  Column() {
    OutlinedTextField(
      value = "",
      onValueChange = {},
      placeholder = { Text("O que você está procurando?") },
      trailingIcon = {
        Icon(
          painter = painterResource(
            id = R.drawable.baseline_search_24
        ),
        contentDescription = ""
      )
    })
  }
}
```

Código-fonte 2 – Exemplo de código utilizando Jetpack Compose
Fonte: Elaborado pelo autor (2023)

Como podemos observar, o Jetpack Compose utiliza uma quantidade de código muito menor, enquanto com XML precisamos de uma quantidade muito maior. Não se preocupe com o código neste momento, por enquanto o objetivo foi apenas demonstrar como a produtividade pode ser muito grande com o Jetpack Compose, mas não apenas isso.

1.5 Criação de um projeto Android com Jetpack Compose

Agora que já temos uma boa base sobre o que é o Jetpack Compose e quais são as vantagens dessa abordagem em relação a tradicional vamos colocar a mão na massa e construir o nosso primeiro projeto com Jetpack Compose.

Neste momento não se preocupe muito com as explicações e detalhes da implementação, elas virão nos próximos capítulos. Fique tranquilo, logo tudo fará sentido e você vai se apaixonar pelo Jetpack Compose. Vamos lá!

2 CRIAÇÃO DE UM PROJETO COM JETPACK COMPOSE

Ao abrir o Android Studio pela primeira vez, você verá uma tela como o da figura “Criar projeto”, clique no botão “*New Project*”.

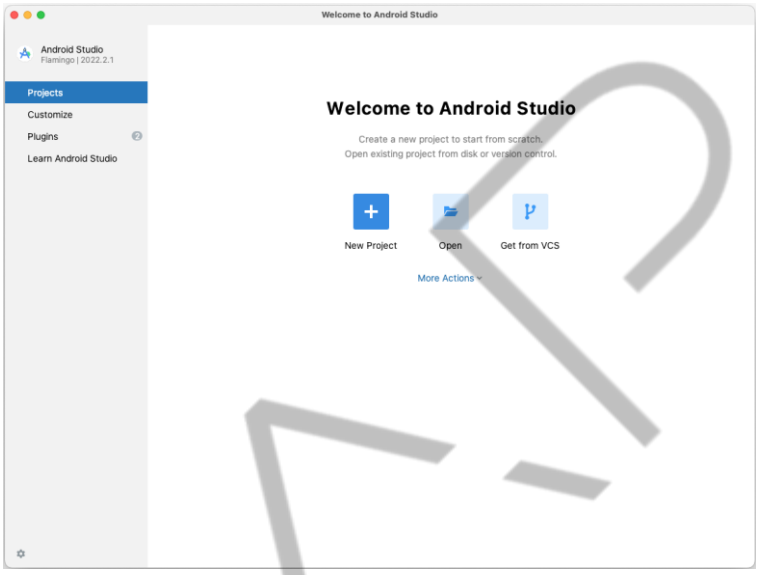

Figura 2 – Criar projeto
Fonte: Elaborado pelo autor (2023)

Na tela “*New Project*” selecione o *template* “*Empty Activity*”, conforme a Figura New Project”:

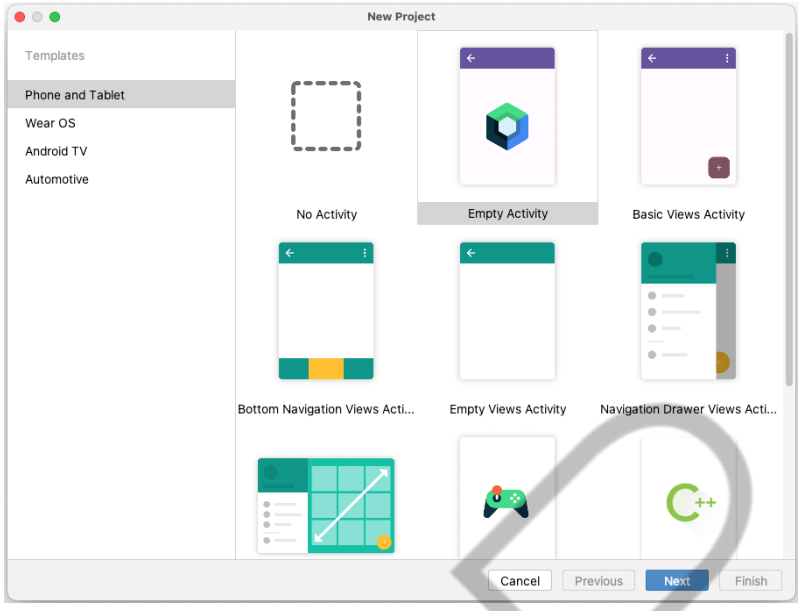

Figura 3 – New Project
Fonte: Elaborado pelo autor (2023)

Na tela “*New Project*”, preencha os dados do projeto conforme demonstrado na Figura “New Project” e clique no botão “*Finish*”:

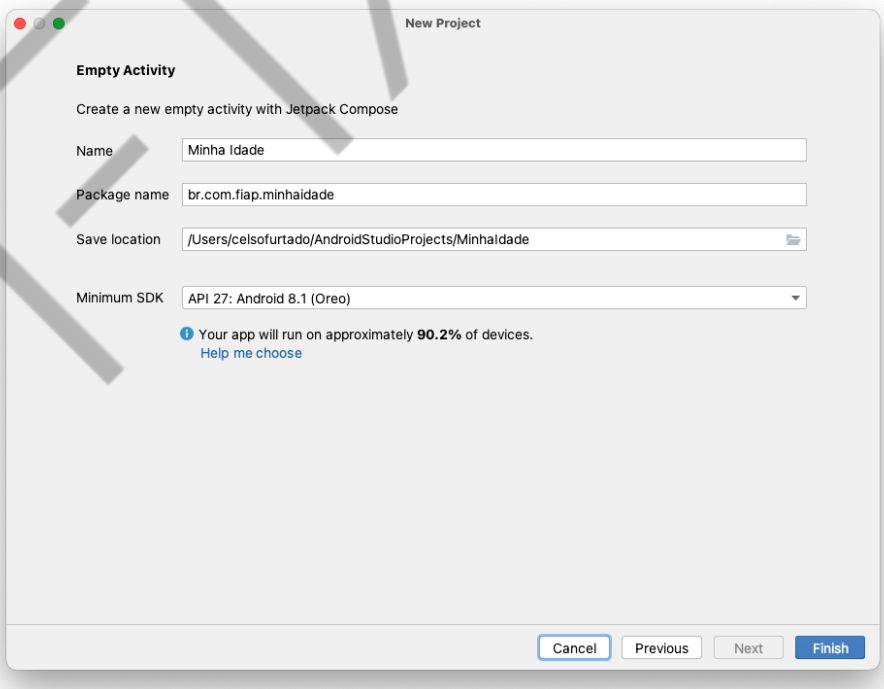

Figura 4 – New Project
Fonte: Elaborado pelo autor (2023)

Aguarde o Android Studio configurar o seu projeto. Ao finalizar você deverá ter uma tela parecida como o da figura “Configuração do projeto”:

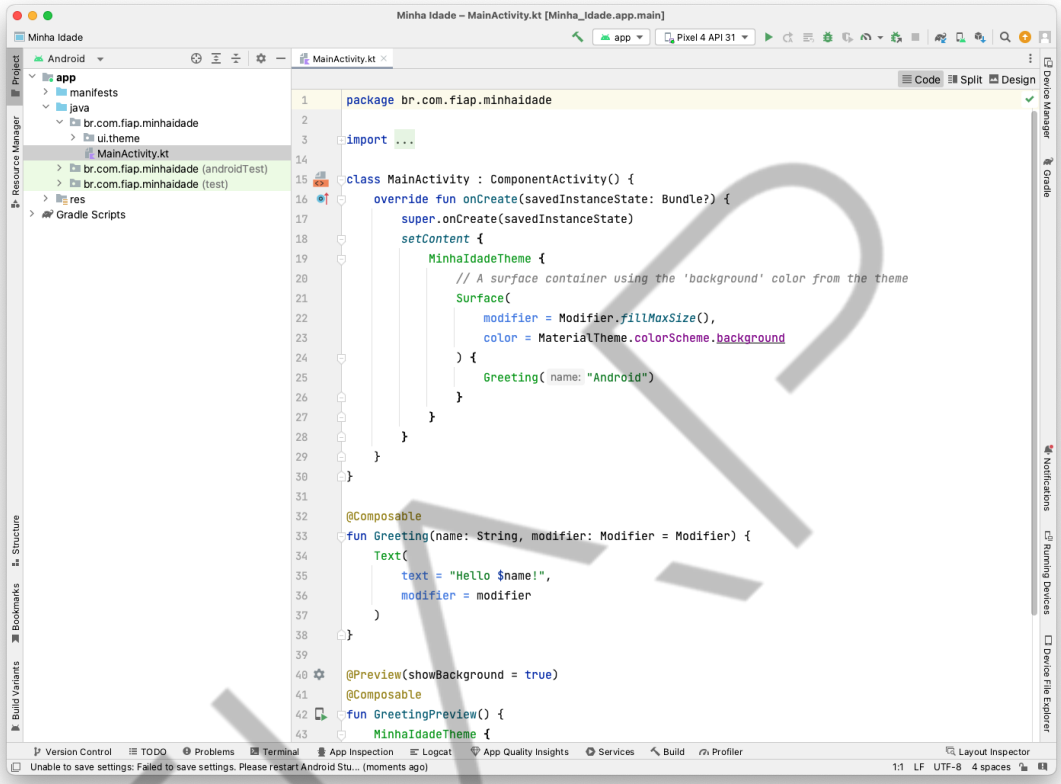

Figura 5 – Configuração do projeto
Fonte: Elaborado pelo autor (2023)

2.1 Estrutura de um projeto Jetpack Compose

Assim que o Android Studio finalizar a configuração do nosso projeto, teremos um arquivo chamado “*MainActivity.kt*”. Esse arquivo será o ponto de partida para nossa aplicação. A “*MainActivity*” é uma subclasse de “*ComponentActivity*”. Isso é necessário para que a *MainActivity* utilize os métodos e propriedades do Jetpack Compose, como o “*setContent*”, que permitirá a criação de um Composable que será nossa IU.

O método “setContent” é chamado na função “*onCreate*” da Activity, que será utilizado para iniciar nossa aplicação. Vamos analisar esse método:

```
override fun onCreate(savedInstanceState: Bundle?) {
  super.onCreate(savedInstanceState)
  setContent {
    MinhaIdadeTheme {
      // A surface container using the 'background' color from the theme
      Surface(
       modifier = Modifier.fillMaxSize(),
       color = MaterialTheme.colorScheme.background
      ) {
         Greeting("Android")
      }
    }
  }
}
```

Código-fonte 3 – Método “OnCreate” padrão
Fonte: Elaborado pelo autor (2023)

O método “*setContent*” é uma função que recebe como parâmetro um *composable*. Pelo nome da função já podemos deduzir o seu papel, ela é responsável por definir o conteúdo da nossa IU, que neste caso é um *composable*.

O primeiro *composable* que é passado ao “*setContent*” é um *composable* responsável por definir o tema da nossa aplicação, ou seja, cores, fontes, dimensões etc. Por ora, é o que precisamos saber sobre os temas, teremos um capítulo especial sobre este assunto mais adiante. O *composable* “MinhaIdadeTheme” por sua vez, recebe como parâmetro outros *composables*, que por padrão no Android Studio começa por um “*Surface*”.

O *Surface* é um container, ele é usado para envolver outros *composables*, por enquanto, pense no *Surface* como se fosse uma DIV do HTML. O *Surface* será o *composable* principal, que conterá todos os outros *composables* que definirão a nossa IU. No nosso exemplo o *composable Surface* está recebendo dois parâmetros, um que determina que ele deverá ocupar toda a tela do dispositivo:

```
modifier = Modifier.fillMaxSize()
```

E outro que define que ele terá uma cor de fundo padrão:

```
color = MaterialTheme.colorScheme.background
```

Além do tamanho e cor é possível modificar outros parâmetros do *Surface.*

O *composable Surface*, por sua vez, recebe um *composable* chamado “*Greeting*”. Esse *composable* recebe um parâmetro do tipo *String*. É importante

ressaltar que “*Greeting*” é uma função, mas não uma função qualquer, ela é uma função *composable*, mas como sabemos que ela é um *composable*? Ela está anotada com *@Composable:*

```
@Composable
fun Greeting(name: String) {
     // Código omitido
}
```

Atenção: *Por padrão, os nomes das funções de composição são escritos com a inicial maiúscula. Isso é importante para diferenciarmos as funções de composição das funções regulares.*

A função “*Greeting*” está declarando um *composable* “*Text*”, que simplesmente exibirá na IU a palavra “Android”

Execute a aplicação em um emulador. O resultado deverá ser parecido com o da figura “Tela emulador”:

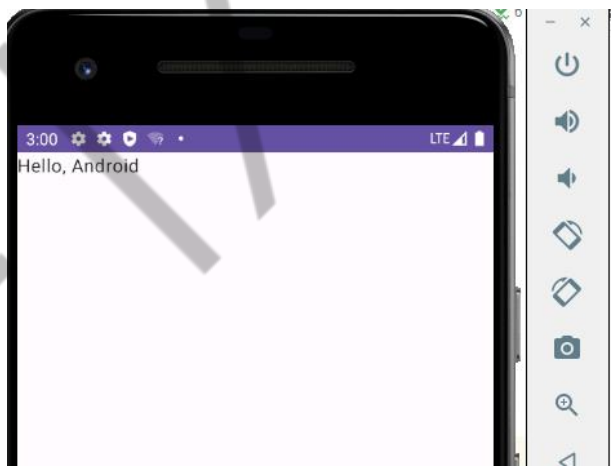

Figura 6 – Tela emulador
Fonte: Elaborado pelo autor (2023)

2.2 Criação da primeira IU

Agora que já sabemos como um projeto Jetpack Compose está estruturado vamos criar o nosso primeiro aplicativo, que consistirá em uma tela onde o usuário irá informar a sua idade utilizando 2 botões, um para incrementar a idade e outro para decrementar. Ao final nossa IU deverá se parecer com a imagem da figura “IU”:

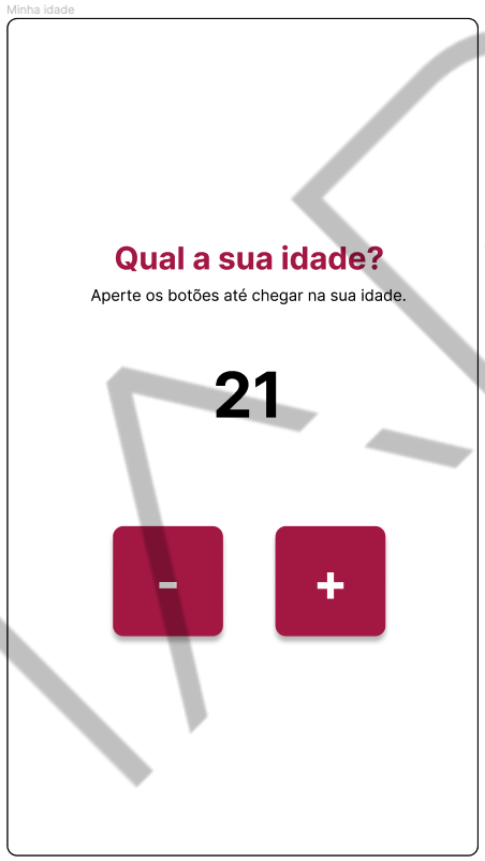

Figura 7 – IU
Fonte: Elaborado pelo autor (2023)

A imagem da figura “IU” nos mostra como nossa IU deverá se parecer. Na figura “Estrutura da IU” vamos olhar com um pouco mais de detalhe e identificar todos os elementos que a compõe:

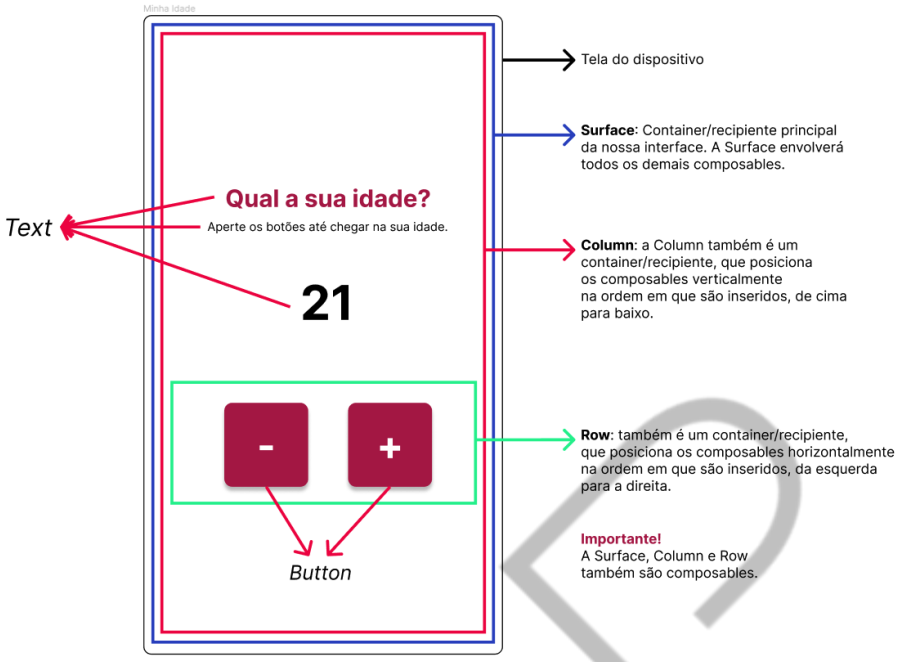

Figura 8 – Estrutura da IU
Fonte: Elaborado pelo autor (2023)

Para implementar a IU sugerida na figura “IU”, abra o arquivo “*MainActivity.kt*” do projeto que acabamos de criar. Vamos apagar algumas linhas que não precisamos agora. Ao final, o seu código deverá se parecer com a listagem abaixo:

```
package br.com.fiap.minhaidade

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.ui.Modifier
import br.com.fiap.contador.ui.theme.ContadorTheme

class MainActivity : ComponentActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContent {
      MinhaIdadeTheme {
        Surface(
          modifier = Modifier.fillMaxSize(),
          color = MaterialTheme.colorScheme.background
        ) {

        }
      }
    }
  }
}
```

Código-fonte 4 – Arquivo MainActivity.kt inicial do projeto
Fonte: Elaborado pelo autor (2023)

Agora vamos escrever uma nova função de composição que incluirá todos os *composables* descritos na figura. Posicione-se no final do arquivo "*MainActivity*", logo após a chave de fechamento da classe e acrescente o seguinte código:

```
@Composable
fun CounterScreen() {
  Column() {
    Text(text = "Qual a sua idade?")
    Text(text = "Aperte os botões para informar a sua idade.")
    Text(text = "21")
    Row() {
      Button(onClick = {}) {
        Text(text = "-")
      }
      Button(onClick = {}) {
        Text(text = "+")
      }
    }
  }
}
```

Código-fonte 5 – Função CounterScreen com componentes iniciais
Fonte: Elaborado pelo autor (XXX)

A função "*CounterScreen*" foi anotada com "*@Composable*", isso foi necessário para identificar essa função como sendo uma função responsável por definir a aparência e o comportamento de um componente visual. Funções composable são os blocos de construção da IU.

Na função "*CounterScreen*" inserimos uma *Column* que organizará nosso layout verticalmente. Dentro da *Column* inserimos três *Texts* e seus respectivos textos, além de uma *Row*, que está posicionando 2 *Buttons* horizontalmente.

Ao final, seu código deverá se parecer com o da listagem abaixo:

```
package br.com.fiap.minhaidade

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material3.Button
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 androidx.compose.ui.tooling.preview.Preview
import br.com.fiap.contador.ui.theme.ContadorTheme

class MainActivity : ComponentActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContent {
      MinhaIdadeTheme {
        Surface(
          modifier = Modifier.fillMaxSize(),
          color = MaterialTheme.colorScheme.background
        ) {

        }
      }
    }
  }
}

@Composable
fun CounterScreen() {
  Column() {
    Text(text = "Qual a sua idade?")
    Text(text = "Aperte os botões para informar a sua idade.")
    Text(text = "21")
    Row() {
      Button(onClick = {}) {
        Text(text = "-")
      }
      Button(onClick = {}) {
        Text(text = "+")
      }
    }
  }
}
```

Código-fonte 6 – Arquivo MainActivity.kt com inclusão da função CounterScreen
Fonte: Elaborado pelo autor (2023)

Lembre-se que nosso aplicativo começa a ser executado pelo método "*onCreate*" da classe "*MainActivity*". Esse método faz uma chamada para o método "*setContent*" que define qual será o conteúdo da nossa IU, que neste caso está sendo definido pela função de composição "*CounterScreen*", então, no corpo do método "setContent" faça uma chamada para a função "*CounterScreen*". O método "*onCreate*" deverá se parecer com o da listagem abaixo:

```
class MainActivity : ComponentActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContent {
      MinhaIdadeTheme {
        Surface(
          modifier = Modifier.fillMaxSize(),
          color = MaterialTheme.colorScheme.background
        ) {
          CounterScreen()
        }
      }
    }
  }
}
```

Código-fonte 7 – Arquivo MainActivity.kt com chamada para a função CounterScreen.
Fonte: Elaborado pelo autor (XXX)

E chegou o grande momento, vamos ver o resultado do que acabamos de criar, então, vamos executar o nosso aplicativo em um emulador que você criou anteriormente.

Na barra de ferramentas do Android Studio, selecione o emulador que você deseja utilizar e clique no botão “*Run app*”, conforme a figura “Lista de emuladores” ou simplesmente mantenha pressionada a tecla “Shift” enquanto pressiona uma vez a tecla “F10”.

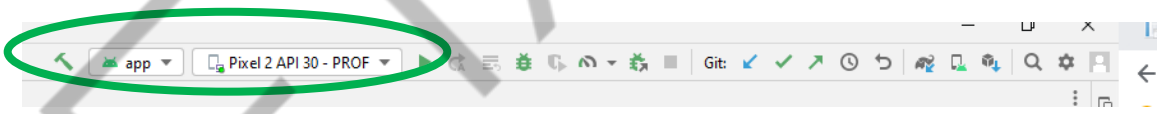

Figura 9 – Lista de emuladores
Fonte: Elaborado pelo autor (2023)

O resultado da aplicação rodando no dispositivo deverá ser parecido com o da figura “Resultado aplicação”:

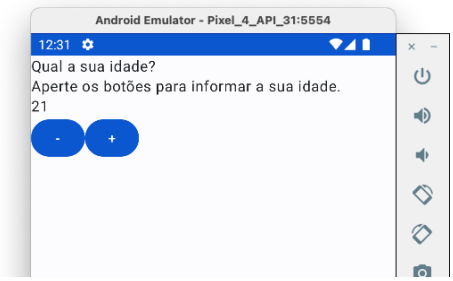

Figura 10 – Resultado aplicação
Fonte: Elaborado pelo autor (2023)

Você já deve ter notado que o resultado obtido não se parece em nada com o que foi sugerido na figura anterior, mas não se preocupe, isso era esperado. Nós inserimos os componentes, mas ainda não os formatamos, ou seja, não declaramos a sua aparência, nem tão pouco o seu comportamento. Vamos fazer isso agora!

Vamos começar com o texto “Qual a sua idade”, inclua os seguintes parâmetros no *composable Text* responsável por renderizar este texto:

```
Text(
  text = "Qual a sua idade?",
  fontSize = 24.sp,
  color = Color(0xFFAD1F4E),
  fontWeight = FontWeight.Bold
)
```

Código-fonte 8 – Formatação básica do composable Text
Fonte: Elaborado pelo autor (2023)

Text é o composable responsável por inserir texto em nossa IU. *Text* é uma função onde podemos fornecer diversos argumentos. Para nosso propósito utilizamos os seguintes argumentos:

- **text** – responsável por fornecer o texto que será exibido.
- **fontSize** – responsável por definir o tamanho da fonte.
- **color** – utilizado para definir a cor do texto.
- **fontWeight** – utilizado para definir o estilo, tais como negrito e itálico.

Além desses argumentos, há muitos outros que veremos mais adiante no curso.

Execute novamente a aplicação e o resultado obtido deverá ser o exibido na figura “Composable Text”:

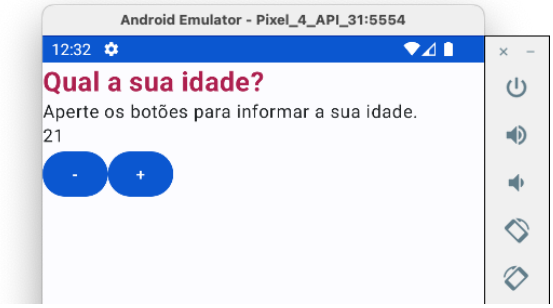

Figura 11 – *Composable Text*
Fonte: Elaborado pelo autor (2023)

Vamos formatar os outros composables, altere o código da função de composição “*CounterScreen*”. Ao final, a função deverá se parecer com a listagem abaixo:

```
@Composable
fun CounterScreen() {
  Column() {
    Text(
      text = "Qual a sua idade?",
      fontSize = 24.sp,
      color = Color(0xFFAD1F4E),
      fontWeight = FontWeight.Bold
    )
    Text(
      text = "Aperte os botões para informar a sua idade.",
      fontSize = 12.sp
    )
    Text(
      text = "21",
      fontSize = 48.sp,
      fontWeight = FontWeight.Bold
    )
    Row() {
      Button(
        onClick = {},
        modifier = Modifier.size(84.dp),
        shape = RoundedCornerShape(8.dp),
        colors = ButtonDefaults.buttonColors(Color(0xFFAD1F4E))
      ) {
        Text(text = "-", fontSize = 40.sp)
      }
      Button(
        onClick = {},
        modifier = Modifier.size(84.dp),
        shape = RoundedCornerShape(8.dp),
        colors = ButtonDefaults.buttonColors(Color(0xFFAD1F4E))
      ) {
        Text(text = "+", fontSize = 40.sp)
      }
    }
  }
}
```

Código-fonte 9 – Função CounterScreen com estilização dos componentes básicos.
Fonte: Elaborado pelo autor (2023)

Em relação aos botões fizemos a inclusão dos seguintes parâmetros:

```
modifier = Modifier.size(84.dp)
```

O parâmetro "*modifier*" é usado para aplicar modificações aos composables, tais como espaçamento, cor, tamanho, etc. Neste caso, o parâmetro *modifier* está alterando o tamanho do *Button*. Como informamos apenas uma dimensão o tamanho está sendo aplicado tanto ao comprimento quanto à altura, tornando o botão quadrado.

```
Shape = RoundedCornerShape(8.dp)
```

O parâmetro “*shape*” modifica a forma do *Button*. Utilizamos a forma “*RoundedCornerShape*”, ou seja, forma com cantos arredondados, e definimos o raio de curvatura para 8 dps. Há outras formas que podem ser usadas, como “*RectangleShape*”, “*CircleShape*”, dentre outras.

```
Colors = ButtonDefaults.buttonColors(Color(0xFFAD1F4E))
```

No parâmetro “*colors*” modificamos a cor de preenchimento do botão. Neste caso estamos utilizando o código hexadecimal da cor.

Após os ajustes, execute novamente a aplicação no emulador. O resultado esperado deve ser parecido com o da figura “Botões”:

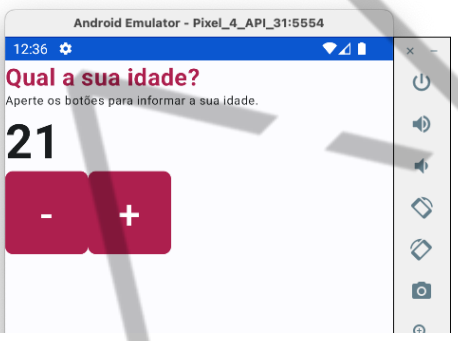

Figura 12 – Botões
Fonte: Elaborado pelo autor (2023)

O resultado da figura “Botões” já está mais próximo do que queremos fazer, mas o alinhamento ainda não está correto. Neste caso, queremos que todo o conteúdo fique no centro, tanto vertical quanto horizontalmente. Lembre-se de que todos os *Texts* e *Buttons* estão dentro de uma *Column*, então queremos alinhar o conteúdo desta *Column*. Acrescente à *Column* os seguintes parâmetros:

```
Column(
  horizontalAlignment = Alignment.CenterHorizontally,
  verticalArrangement = Arrangement.Center
) {
    // Código omitido
}
```

Código-fonte 10 – Uso de alinhamento e arranjo do componente Column.
Fonte: Elaborado pelo autor (2023)

O parâmetro “*horizontalAlignment*” define o alinhamento horizontal no interior da *Column*. O valor que passamos neste caso, foi “*Alignment.CenterHorizontally*”, que alinha o conteúdo de forma centralizada na horizontal.

O parâmetro “*verticalArrangement*” define a disposição vertical no interior da *Column*. O valor “*Arrangement.Center*”, posiciona o conteúdo no centro vertical da *Column*.

Essas duas configurações colocaram o conteúdo exatamente no centro da IU.

Execute a aplicação no emulador, o resultado obtido deve ser parecido com a figura “Arrangement Center”:

Figura 13 - *Arrangement.Center*
Fonte: Elaborado pelo autor (2023)

Ainda há mais um ajuste para fazermos. De acordo com o layout sugerido, existe um espaço maior antes e depois do texto “21”. Para obtermos esse espaçamento vamos utilizar um *composable* chamado “*Spacer*”, então, vamos acrescentar o “*Spacer*” antes e depois do texto “21”. O seu código deverá ficar da seguinte forma:

```
... Código omitido
Spacer(modifier = Modifier.height(32.dp))
Text(
  text = "21",
  fontSize = 48.sp,
  fontWeight = FontWeight.Bold
)
Spacer(modifier = Modifier.height(32.dp))
... Código omitido
```

Código-fonte 11 – Uso de espaçamento vertical com Spacer.
Fonte: Elaborado pelo autor (2023)

Execute novamente o aplicativo no emulador. O aplicativo deve se parecer com a figura "Spacer":

Figura 14 – *Spacer*
Fonte: Elaborado pelo autor (2023)

Mais um ajuste para finalizar. Entre os botões para incrementar ou decrementar a idade também há um espaço. Faremos da mesma forma que no espaçamento entre os textos na vertical. Naquele momento utilizamos o *Spacer* com o modificador

"*height*", agora vamos utilizar o modificador "*width*" para que o espaçamento seja na horizontal. Faça o ajuste como na listagem abaixo:

```
Row {
  Button(
    onClick = {},
    modifier = Modifier.size(84.dp),
    shape = RoundedCornerShape(8.dp),
    colors = ButtonDefaults.buttonColors(Color(0xFFAD1F4E))
  ) {
    Text(text = "-", fontSize = 40.sp)
  }
  Spacer(modifier = Modifier.width(32.dp))
  Button(
    onClick = {},
    modifier = Modifier.size(84.dp),
    shape = RoundedCornerShape(8.dp),
    colors = ButtonDefaults.buttonColors(Color(0xFFAD1F4E))
  ) {
    Text(text = "+", fontSize = 40.sp)
  }
}
```

Código-fonte 12 – Uso do espaçamento horizontal com Spacer
Fonte: Elaborado pelo autor (2023)

Após o ajuste, finalmente já temos nossa IU conforme o layout sugerido.

Execute novamente o aplicativo no emulador, seu aplicativo deve-se parecer com o da figura "Space width":

Figura 15 – *Spacer width*
Fonte: Elaborado pelo autor (2023)

2.3 Definindo o comportamento da nossa aplicação

O objetivo da nossa aplicação é permitir que o usuário indique a sua idade pressionando os botões “+” e “-” que irão incrementar ou decrementar o número exibido na caixa de texto, que inicialmente possui o número “21”. Para isso será necessário implementarmos os métodos “onClick” dos botões.

Na primeira linha da função de composição “*CounterScreen*” vamos criar a variável “idade”, conforme a listagem abaixo:

```
@Composable
fun CounterScreen() {

  var idade = 0

  Column(
    horizontalAlignment = Alignment.CenterHorizontally,
    verticalArrangement = Arrangement.Center
  )
.... trecho de código omitido
```

Código-fonte 13 – Criação e inicialização da variável idade.
Fonte: Elaborado pelo autor (XXX)

Agora, vamos alterar o valor do parâmetro “text” do *composable* “Text” que exibe a idade do usuário de “21” para “$idade”. O seu código deverá ficar como na listagem abaixo:

```
@Composable
fun CounterScreen() {

  var idade = 0

  Column(
    horizontalAlignment = Alignment.CenterHorizontally,
    verticalArrangement = Arrangement.Center
  ) {
    Text(
      text = "Qual a sua idade?",
      fontSize = 24.sp,
      color = Color(0xFFAD1F4E),
      fontWeight = FontWeight.Bold
    )
    Text(
      text = "Aperte os botões para informar a sua idade.",
      fontSize = 12.sp
    )
    Spacer(modifier = Modifier.height(32.dp))
    Text(
      text = "$idade",
      fontSize = 48.sp,
      fontWeight = FontWeight.Bold
    )
    Spacer(modifier = Modifier.height(32.dp))
    Row {
      Button(
        onClick = {},
        modifier = Modifier.size(84.dp),
        shape = RoundedCornerShape(8.dp),
        colors = ButtonDefaults.buttonColors(Color(0xFFAD1F4E))
      ) {
        Text(text = "-", fontSize = 40.sp)
      }
      Spacer(modifier = Modifier.width(32.dp))
      Button(
        onClick = {},
        modifier = Modifier.size(84.dp),
        shape = RoundedCornerShape(8.dp),
        colors = ButtonDefaults.buttonColors(Color(0xFFAD1F4E))
      ) {
        Text(text = "+", fontSize = 40.sp)
      }
    }
  }
}
```

Código-fonte 14 – Atribuição de valor ao parâmetro value do Text.
Fonte: Elaborado pelo autor (2023)

Execute o aplicativo, veremos que agora o texto já inicia com o valor “0”. Sua aplicação deverá se parecer com a figura “Composable Text $idade”:

Figura 16 - *Composable* “Text “$idade”
Fonte: Elaborado pelo autor (2023)

Quando clicamos nos botões, nada acontece. Claro, ainda não programamos o comportamento. O *composable Button* possui um método obrigatório chamado “onClick”, esse método é o responsável por capturar o clique no botão e executar uma tarefa qualquer. Em nosso caso, queremos que a cada clique nos botões “+” ou “-” seja incrementado ou decrementado 1 a idade. Então vamos alterar o método “onClick” dos botões para que executem essa tarefa. Seu código, com a alteração deve se parecer com a listagem abaixo:

```
Row {
  Button(
    onClick = { idade-- },
    modifier = Modifier.size(84.dp),
    shape = RoundedCornerShape(8.dp),
    colors = ButtonDefaults.buttonColors(Color(0xFFAD1F4E))
  ) {
    Text(text = "-", fontSize = 40.sp)
  }
  Spacer(modifier = Modifier.width(32.dp))
  Button(
    onClick = { idade++ },
    modifier = Modifier.size(84.dp),
    shape = RoundedCornerShape(8.dp),
    colors = ButtonDefaults.buttonColors(Color(0xFFAD1F4E))
  ) {
    Text(text = "+", fontSize = 40.sp)
  }
}
```

Código-fonte 15 – Implementação do click dos botões.
Fonte: Elaborado pelo autor (2023)

Ao executar o aplicativo verificamos que nada ocorre, o valor da idade continua sendo “0”, mas o que está acontecendo?

2.4 Gerenciando o State da nossa aplicação

Quando pressionamos os botões, queremos que o valor da idade seja atualizado. O Jetpack Compose gerencia o que deve ser mostrado na IU através do “*State*”. O “state” é um dos principais fundamentos do Jetpack Compose, já que é ele quem vai orientar os *composables* a atualizarem a sua aparência com base nos dados que eles exibem.

Em nosso aplicativo o *composable Text* deve atualizar o seu valor, ou seja, a sua aparência, para refletir o seu estado atual, que é um valor diferente do que estava sendo exibido antes do clique.

Quem mantém o estado do *Text* é a variável “idade”, pois é ela que guarda o valor que deve ser apresentado ao usuário. Sempre que “idade” mudar, o *Text* deve reagir a essa mudança e exibir o valor correto. É como se disséssemos ao *Text* algo como:

"*Text, fique observando a variável idade, se o valor de idade mudar, reflita essa mudança!".*

Assim sendo, a variável "idade" é uma variável de estado. Precisamos fazer um pequeno ajuste na declaração a variável "idade" para que o Jetpack Compose possa gerenciá-lo. Altere a declaração da variável "idade", para que se pareça com o da listagem abaixo:

```
var idade = remember {
  mutableStateOf(0)
}
```

Código-fonte 16 – Criação da variável de estado com a função mutableStateOf()
Fonte: Elaborado pelo autor (2023)

A função "remember" é utilizada para criarmos as variáveis de estado juntamente com a função "*mutableSateOf*", que torna a variável mutável e determina o valor inicial. No caso da variável "idade" estamos dizendo que ela poderá ter seu valor alterado e é do tipo "Int", já que a inicializamos com "0".

Para acessarmos o valor em uma variável de estado utilizamos a função "*value*" da variável, então ajuste os métodos "*onClick*" dos botões de acordo com a listagem abaixo:

```
Row {
  Button(
    onClick = { idade.value-- },
    modifier = Modifier.size(84.dp),
    shape = RoundedCornerShape(8.dp),
    colors = ButtonDefaults.buttonColors(Color(0xFFAD1F4E))
  ) {
    Text(text = "-", fontSize = 40.sp)
  }
  Spacer(modifier = Modifier.width(32.dp))
  Button(
    onClick = { idade.value++ },
    modifier = Modifier.size(84.dp),
    shape = RoundedCornerShape(8.dp),
    colors = ButtonDefaults.buttonColors(Color(0xFFAD1F4E))
  ) {
    Text(text = "+", fontSize = 40.sp)
  }
}
```

Código-fonte 17 – Uso do parâmetro value.
Fonte: Elaborado pelo autor (XXX)

Execute novamente a aplicação e clique nos botões. Será que agora nosso aplicativo exibirá o valor correto? Sim, agora nosso aplicativo se comporta da forma correta, incrementando ou decrementando o valor da idade.

3 DESAFIO

Para praticar o que você aprendeu neste capítulo, te desafio a implementar algumas funcionalidades extras em nosso aplicativo.

1. Adicione um texto abaixo dos botões que exiba uma mensagem informando se o usuário é maior ou menor de idade. Se a idade apresentada for maior ou igual a 18 exiba a mensagem “Você é MAIOR de idade!”, caso contrário, exiba a mensagem “Você é MENOR de Idade!”.
2. Outro ajuste interessante, seria impedir que o valor da idade seja menor do que zero ou maior do que 130. Corrija os métodos “*onClick*” de modo a atender esse requisito.

Sugestão de layout:

Figura 17 – Layout de IU sugerido para o desafio
Fonte: Elaborado pelo autor (2023)

CONCLUSÃO

Neste capítulo, aprendemos de forma geral como é uma aplicação Android utilizando o Jetpack Compose. Aprendemos como podemos desenvolver uma aplicação Android de forma rápida, eficiente e com menos código.

Alguns detalhes e explicações foram omitidas de maneira proposital. Nos próximos capítulos entraremos mais a fundo nos conceitos de estado, *composables*, componentização, dentre outros assuntos interessantíssimos. Então, continue seus estudos e logo você será um Android Developer. Bons Estudos!

REFERÊNCIAS

ANDROID. **Primeiros passos com o Jetpack Compose**. 2023. Disponível em: <https://developer.android.com/jetpack/compose/documentation?hl=pt-br>. Acesso em: 28 jun. 2023.

ANDROID. **Trabalhando com Compose**. 2023. Disponível em: <https://developer.android.com/jetpack/compose/mental-model?hl=pt-br>. Acesso em: 28 jun. 2023.

ANDROID. **Estado e Jetpack Compose**. 2023. Disponível em: <https://developer.android.com/jetpack/compose/state?hl=pt-br>. Acesso em: 28 jun. 2023.