APP WORLD

COMPONENTES BÁSICOS COM JETPACK COMPOSES

6A

LISTA DE FIGURAS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

1 COMPONENTES BÁSICOS COM JETPACK

Já vimos como organizar nossos componentes através do uso dos composables de layout, como *Column*, *Row* e *Box*. Mas, esses composables são utilizados apenas para organizar a distribuição dos componentes em nossa IU.

Nós precisamos de muito mais. Precisamos de que o usuário interaja com nossa aplicação através de botões, listas, rádios, caixas de checagem, listas suspensas etc.

O Jetpack Compose possui uma biblioteca muito grande, com praticamente todos os componentes necessários para construir nossa aplicação.

A partir de agora vamos conhecer esses componentes e entender a mecânica de funcionamento de cada um neles. Eu te convido para essa nova viagem. Vamos lá?

1.1 Exibindo texto para o usuário

O componente mais básico do Jetpack Compose é o "**Text**", com ele podemos exibir informações para o usuário.

Vamos criar um projeto no Android Studio chamado "***Basic Components***".

Assim que o Android Studio concluir a criação do projeto apague todas as funções do arquivo "**MainActivity.kt**". Seu projeto deverá se parecer com a **figura "**Arquivo MainActivity.kt":

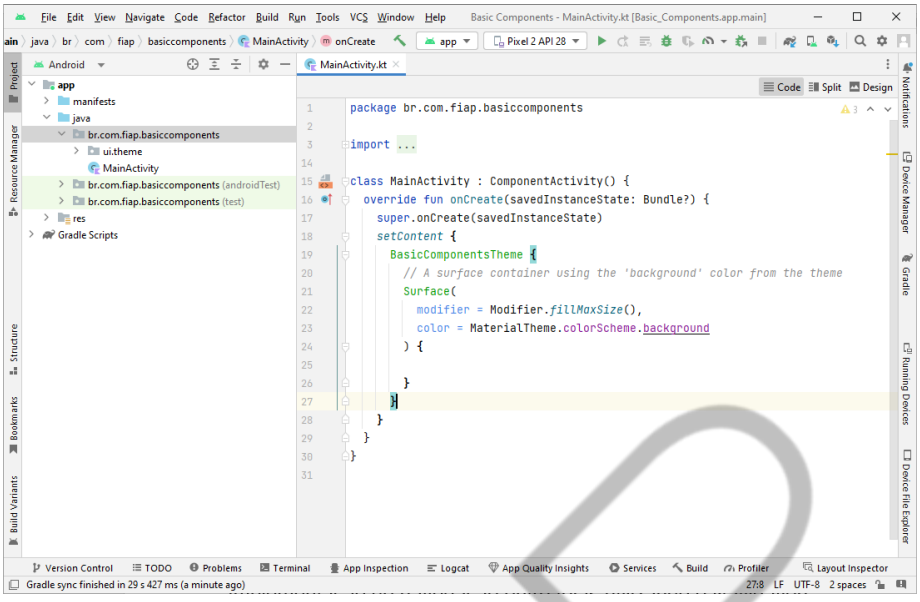

Figura 1 – Arquivo MainActivity.kt
Fonte: Elaborado pelo autor (2023)

Vamos criar uma função de composição chamada "***BasicComponentsScreen***", e inserir dois composables do tipo "Text". Seu código deverá se parecer com a listagem abaixo:

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

@Composable
fun BasicComponentsScreen() {
  Column(modifier = Modifier.fillMaxWidth()) {
    Text(text = "FIAP")
    Text(text = "Desenvolvendo aplicações Android")
  }
}
```

Código-fonte 1 – Função BasicComponentsScreen
Fonte: Elaborado pelo autor (2023)

Vamos executar a nossa aplicação em um emulador. O resultado deverá se parecer com a **figura "**Preview da função BasicComponentsScreen**"**:

Figura 2 – Preview da função BasicComponentsScreen
Fonte: Elaborado pelo autor (2023)

Os componentes "Text" ficaram empilhados na vertical pois estão inseridos em uma *Column*. Vamos alterar a formatação dos componentes para que nossa IU se pareça com a **figura "**Composables Text após formatação de texto***"*:**

Figura 3 – Composables Text após formatação de texto.
Fonte: Elaborado pelo autor (2023)

1.2 Formatação básica do texto

O *composable* "Text*"* implementa vários parâmetros que são responsáveis pela formatação da aparência do texto inserido no "Text*"*. Além disso, temos o "***Modifier***", que nos permite modificar a aparência do componente. Vamos aplicar as modificações para que o título "**FIAP**" fique com a aparência de acordo com a **figura "**Composables Text após formatação de texto**"**.

O código da função "***BasicComponentsScreen***" deverá se parecer com a listagem abaixo:

```
@Composable
fun BasicComponentsScreen() {
  Column(modifier = Modifier.fillMaxWidth()) {
    Text(
      text = "FIAP",
      fontSize = 32.sp,
      fontWeight = FontWeight.Bold,
      color = Color(0xFFED145B)
    )
    Text(text = "Desenvolvendo aplicações Android")
  }
}
```

Código-fonte 2 – Formatação do título.
Fonte: Elaborado pelo autor (2023)

O que mudamos no texto:

fontSize: este parâmetro foi definido para "**32.sp**". O **32** representa o tamanho que queremos e o "**sp**" significa "***scaled pixels***", que é uma unidade utilizada para definir o tamanho do texto com base nas configurações de tamanho de fonte do dispositivo. Lembre-se de que o usuário pode alterar o tamanho das fontes utilizadas pelo dispositivo. Utilizar "**sp**" vai garantir que nossa aplicação obedecerá a esta configuração.

fontWeight: neste parâmetro configuramos a intensidade ou peso da fonte. Utilizamos "**FontWeight.Bold**", que colocará nosso texto em estilo negrito. Além do "**bold**", temos outras opções, tais como:

1. **FontWeight.Thin**: Define a fonte com peso fino.
2. **FontWeight.ExtraLight**: Define a fonte com peso extra leve.
3. **FontWeight.Light**: Define a fonte com peso leve.

4. **FontWeight.Normal**: Define a fonte com peso normal.
5. **FontWeight.Medium**: Define a fonte com peso médio.
6. **FontWeight.SemiBold**: Define a fonte com peso semi-negrito.
7. **FontWeight.Bold**: Define a fonte com peso negrito.
8. **FontWeight.ExtraBold**: Define a fonte com peso extra negrito.
9. **FontWeight.Black**: Define a fonte com peso preto.

color: com este parâmetro configuramos a cor do nosso texto. Em nosso exemplo utilizamos o código hexadecimal da cor vermelha que escolhemos. O código hexadecimal da cor que escolhemos é **ED145B**, mas o que significa o "**0xFF**"?

0x: sempre que vamos fornecer um valor hexadecimal para uma cor no Jetpack Compose devemos começar com esse prefixo.

FF: chamamos isso de "canal alpha" e representa a transparência de uma cor, onde FF significa totalmente opaco e 00 totalmente transparente.

Importante: Além do uso de hexadecimal para definição da cor, também podemos utilizar o **RGB**, que é a mistura de vermelho, verde e azul. Para o nosso exemplo, poderíamos ter utilizado **Color(237, 20, 91)**.

Vamos modificar agora o *background* da nossa IU. A *Column* está ocupando toda a nossa tela e ela é o nosso contêiner principal. Altere o seu código de modo que fique de acordo com a listagem abaixo:

```
@Composable
fun BasicComponentsScreen() {
  Column(modifier = Modifier
    .fillMaxWidth()
    .background(Color.Black)) {
    Text(
      text = "FIAP",
      fontSize = 32.sp,
      fontWeight = FontWeight.Bold,
      color = Color(237, 20, 91)
    )
    Text(text = "Desenvolvendo aplicações Android")
  }
}
```

Código-fonte 3 – Formatação de background
Fonte: Elaborado pelo autor (2023)

Observe que para alterar a cor de fundo da *Column*, utilizamos o "***Modifier.background(Color.Black)***", então, além de usar RGB e hexadecimal para as cores, também podemos utilizar cores pré-configuradas. Além do "*Black*" temos outras cores, como *Red*, *White*, *Green* etc.

Ao executar o seu aplicativo, o resultado esperado deverá ser conforme a **figura "**Preview do título formatado**"**:

Figura 4 – Preview do título formatado
Fonte: Elaborado pelo autor (2023)

O subtítulo sumiu! Na verdade, a cor do texto é a mesma do *background*, então vamos alterar a configuração para que fique de acordo com o layout sugerido. Altere seu código de modo que ele se pareça com a listagem abaixo:

```
@Composable
fun BasicComponentsScreen() {
  Column(modifier = Modifier
    .fillMaxWidth()
    .background(Color.Black)) {
    Text(
      text = "FIAP",
      fontSize = 32.sp,
      fontWeight = FontWeight.Bold,
      color = Color(237, 20, 91)
    )
    Text(
      text = "Desenvolvendo aplicações Android",
      fontSize = 16.sp,
      fontWeight = FontWeight.SemiBold,
      color = Color.White
    )
  }
}
```

Código-fonte 4 – Formatação do texto de subtítulo
Fonte: Elaborado pelo autor (2023)

Execute a aplicação novamente, seu código deve se parecer com a **figura** **“**Preview do subtítulo formatado**”**:

Figura 5 – Preview do subtítulo formatado
Fonte: Elaborado pelo autor (2023)

1.3 Adicionando fontes ao projeto

O Android Studio já possui algumas famílias de fonte genéricas que podemos utilizar por padrão. As principais são as seguintes:

1. **Monospace.**
2. **Serif.**
3. **SansSerif.**
4. **Cursive.**
5. **Default.**

Vamos trocar a fonte do título da nossa IU. Seu código deverá se parecer como a listagem abaixo:

```
@Composable
fun BasicComponentsScreen() {
  Column(modifier = Modifier
    .fillMaxWidth()
    .background(Color.Black)) {
    Text(
      text = "FIAP",
      fontSize = 32.sp,
      fontWeight = FontWeight.Bold,
      color = Color(237, 20, 91),
      fontFamily = FontFamily.Serif
    )
    Text(
      text = "Desenvolvendo aplicações Android",
      fontSize = 16.sp,
      fontWeight = FontWeight.Bold,
      color = Color.White
    )
  }
}
```

Código-fonte 5 – Formatação do título principal
Fonte: Elaborado pelo autor (2023)

Vamos executar o aplicativo no emulador e observar o que mudou. O resultado esperado deve ser como o apresentado na figura “Preview da nova fonte do título”:

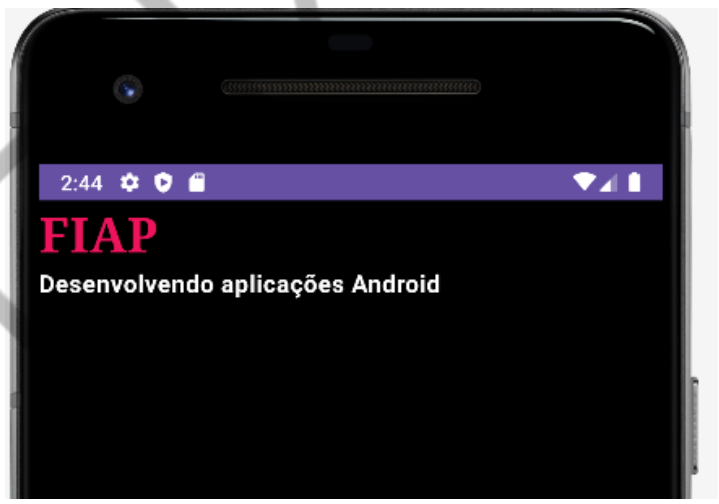

Figura 6 – Preview da nova fonte do título
Fonte: Elaborado pelo autor (2023)

A fonte do título da aplicação mudou, não é mesmo? Agora percebemos alguns detalhes nos cantos das letras, esses detalhes chamamos de “**Serifa**”.

Podemos utilizar qualquer outra fonte que desejarmos. Para isso é necessário adicionarmos essas fontes ao nosso projeto. Vamos acessar o site do Google Fonts e baixar uma fonte chamada “**Righteous**”. Siga os passos abaixo:

1 – Acesse, a partir do seu navegador o endereço https://fonts.google.com/;

2 – No campo de busca de fontes digite o nome da fonte que queremos baixar, que é “**Righteous**”. O resultado da página deverá ser como o apresentado na **figura “**Filtro de fontes no Google Fonts”:

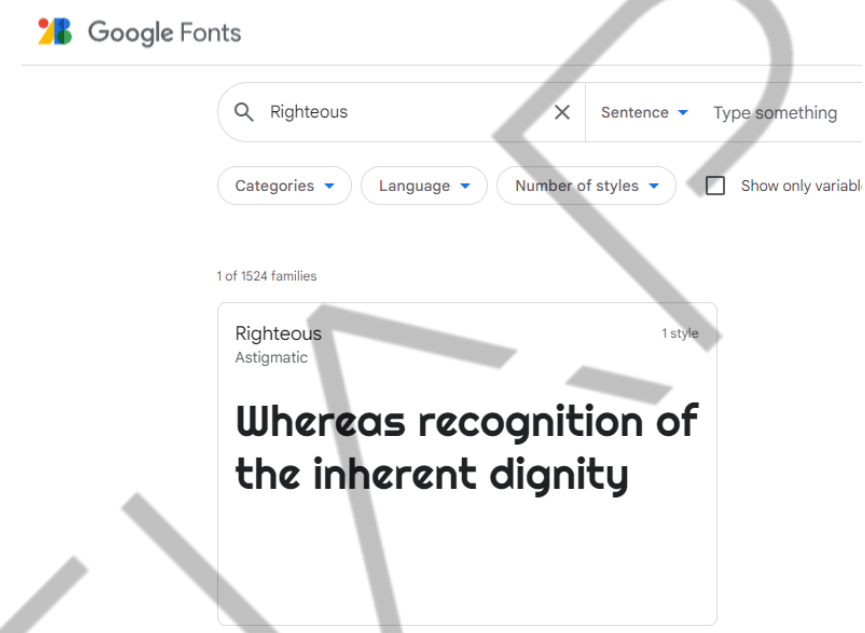

Figura 7 – Filtro de fontes no Google Fonts
Fonte: Elaborado pelo autor (2023)

3 – Clique na fonte resultante da pesquisa. Você será direcionado para outra página com as opções da fonte selecionada.

4 – Clique no botão “Download Family”, do lado superior direito da página, conforme a figura “Botão de download de fontes”:

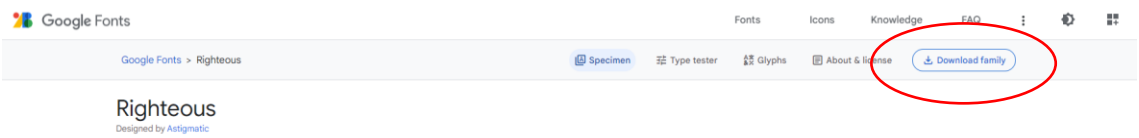

Figura 8 – Botão de download de fontes
Fonte: Elaborado pelo autor (2023)

5 – Salve o arquivo “.zip” em uma pasta qualquer do seu computador.

6 – Descompacte o arquivo “.zip” em uma pasta de sua escolha.

Agora que já temos a fonte que desejamos utilizar, vamos configurar o Android Studio para utilizar essa nova fonte.

Do lado esquerdo do Android Studio temos o painel de projeto, conhecido como “***Project***”. Temos um exemplo deste painel na **figura “**Painel com estrutura do projeto”:

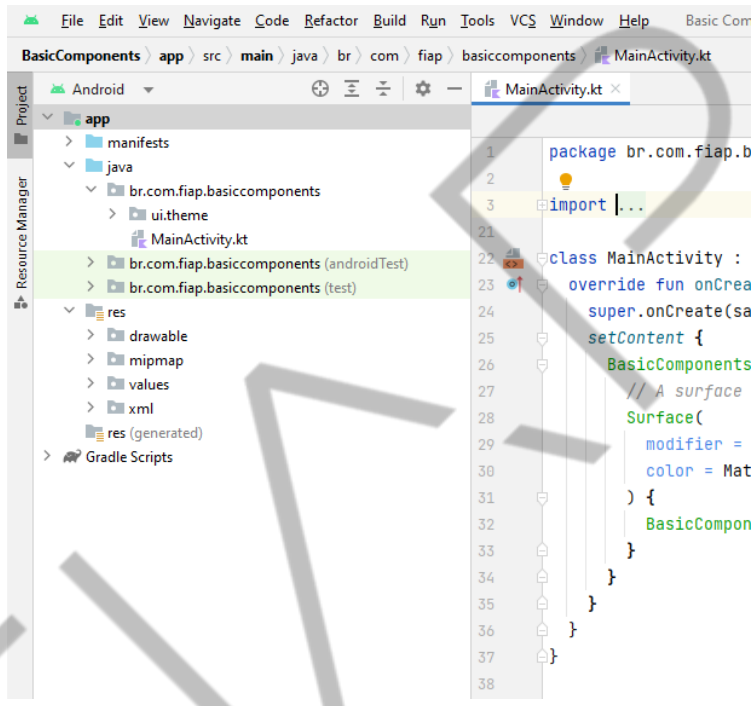

Figura 9 – Painel com estrutura do projeto
Fonte: Elaborado pelo autor (2023)

Neste painel temos a estrutura de pastas do nosso projeto. Nesta estrutura temos uma pasta chamada “**res**”, que significa “**resources**”, ou seja, recursos. Essa pasta é utilizada para colocarmos todos os recursos que serão utilizados em nosso projeto, como as imagens que ficam na pasta “**drawable**” e “**mipmap**”. Há outros recursos que podemos armazenar nesta pasta, mas veremos isso em outro momento. O que nos interessa agora são os recursos de fonte. Para criar uma pasta de recursos de fonte, siga os passos abaixo:

1 – Clique como o botão direito do mouse na pasta “res”, selecione a opção “***Android Resource Directory***”, conforme mostrado na **figura “**Menu Android Resource Directory**”**:

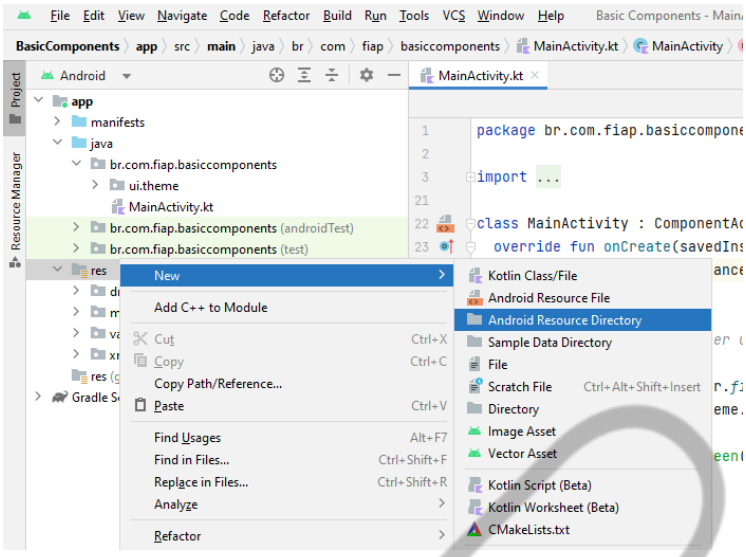

Figura 10 – Menu Android Resource Directory
Fonte: Elaborado pelo autor (2023)

2 – Na janela “**New Resource Directory**” que será aberta, em “**Resource Type**”, abra a lista e selecione a opção “**Font**”. Sua tela deverá ficar como na **figura** “Janela New Resource Directory”:

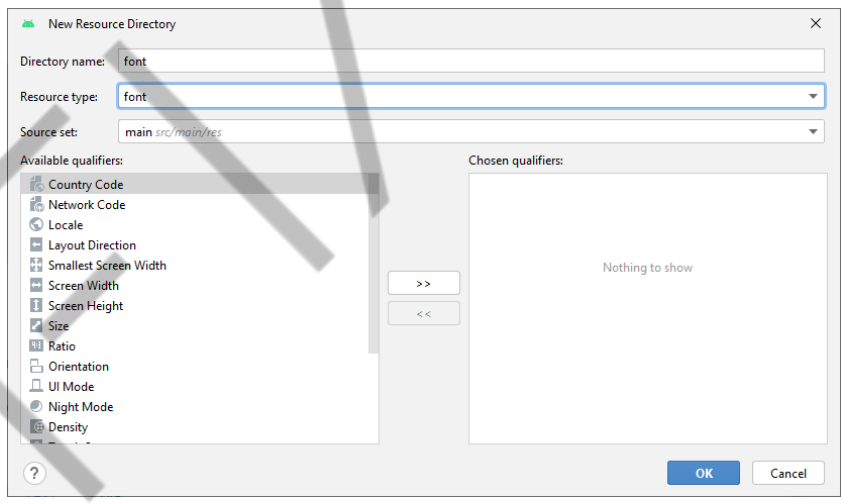

Figura 11 - janela New Resource Directory
Fonte: Elaborado pelo autor (2023)

3 – Pressione o botão “**OK**”. Agora, na pasta “res” temos uma nova pasta chamada “**font**” de acordo com a **figura “**Nova pasta de recursos para fonte”:

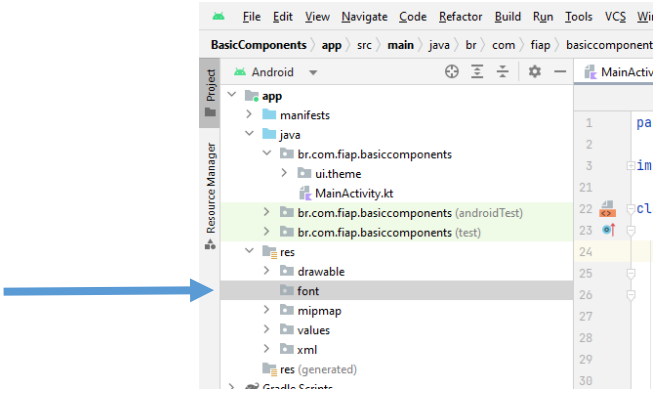

Figura 12 – Nova pasta de recursos para fonte
Fonte: Elaborado pelo autor (2023)

4 – Localize a pasta de fontes que você usou para descompactar a fonte que baixamos do *Google Fonts*. Vamos copiar o arquivo de fonte chamado "**Righteous-Regular.ttf**" para a pasta "**font**" que acabamos de criar no Android Studio. O resultado deverá ser como o mostrado na **figura "**Nova fonte na pasta de recursos do Android Studio":

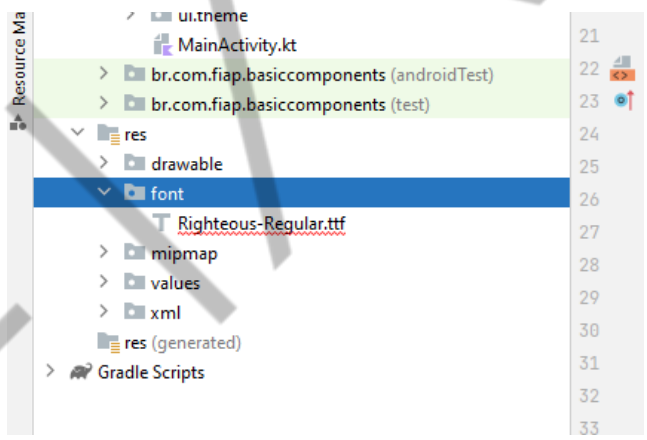

Figura 13 – Nova fonte na pasta de recursos do Android Studio
Fonte: Elaborado pelo autor (2023)

5 – Vamos renomear o arquivo da fonte, porque o nome atual não atende as regras de nome de arquivos de recursos no Android Studio. Vamos trocar o seu nome para "**righteous_regular.ttf**". Clique como o botão direito do mouse no nome arquivo atual, selecione "**Refactor**" e em seguida "**Rename**", como mostrado na **figura "**Renomear o arquivo da fonte":

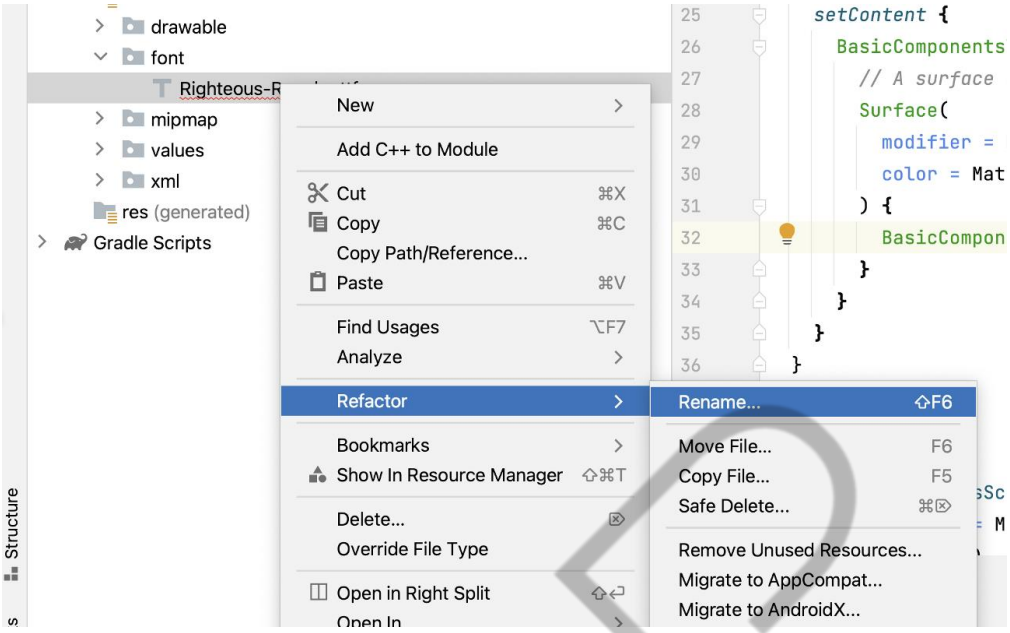

Figura 14 – Renomear o arquivo da fonte
Fonte: Elaborado pelo autor (2023)

6 – Na janela “**Rename**”, troque o nome do arquivo e pressione o botão “**Refactor**”, de acordo com a **figura “**Tela Rename”:

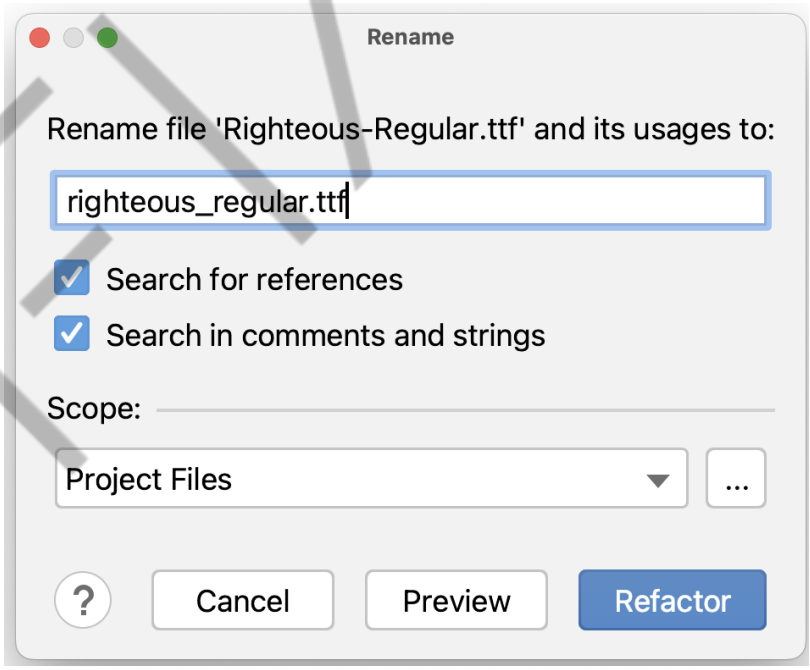

Figura 15 – Tela Rename
Fonte: Elaborado pelo autor (2023)

Prontinho, nossa nova fonte já está disponível para ser utilizada em nosso aplicativo. Mas, para que possamos utilizá-la, teremos que adicionar algumas configurações no arquivo "***Type.kt***". Este arquivo está localizado no pacote "***ui.theme***" como mostrado na **figura "**Arquivo Type.kt".:

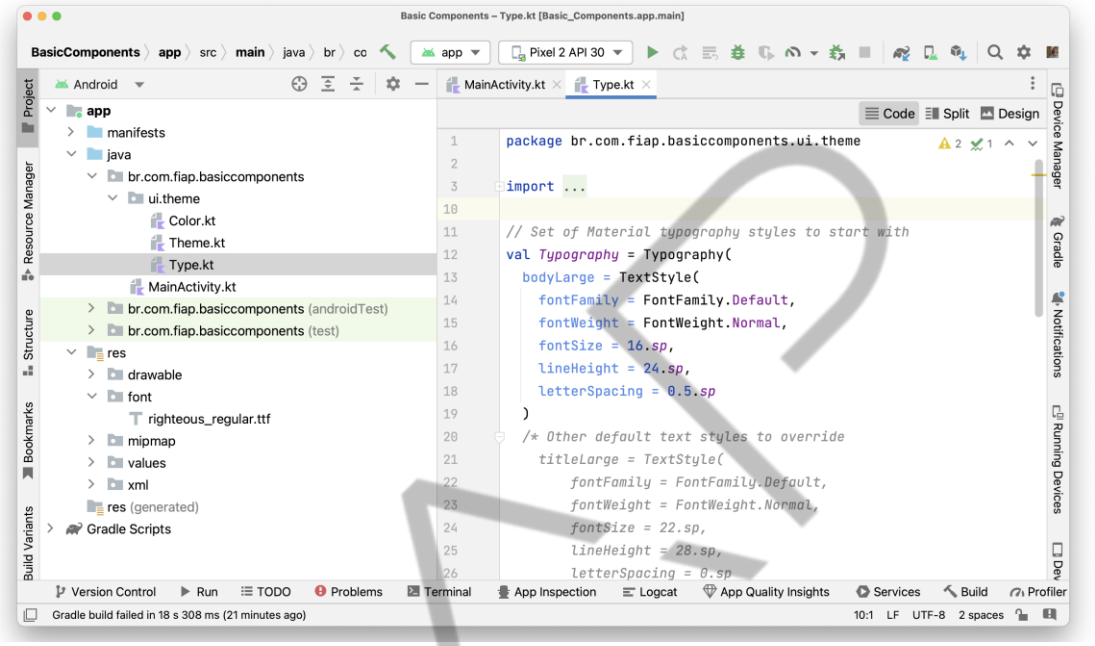

Figura 16 – Arquivo Type.kt
Fonte: Elaborado pelo autor (2023)

O arquivo "**Type.kt**" é um arquivo de código fonte responsável pela padronização de fontes utilizadas no aplicativo. Nele podemos centralizar toda a identidade tipográfica da aplicação, facilitando rápida troca de fontes e personalização.

Abra o arquivo "Type.kt" e adicione as seguintes linhas de código. Ao final, seu código deverá se parecer com a listagem abaixo:

```
package br.com.fiap.basiccomponents.ui.theme

import androidx.compose.material3.Typography
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.Font
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.sp
import br.com.fiap.basiccomponents.R

val Righteous = FontFamily(
  Font(R.font.righteous_regular)
)

// Set of Material typography styles to start with
val Typography = Typography(
  bodyLarge = TextStyle(
    fontFamily = FontFamily.Default,
    fontWeight = FontWeight.Normal,
    fontSize = 16.sp,
    lineHeight = 24.sp,
    letterSpacing = 0.5.sp
  )
  /* Other default text styles to override
    titleLarge = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 22.sp,
        lineHeight = 28.sp,
        letterSpacing = 0.sp
    ),
    labelSmall = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Medium,
        fontSize = 11.sp,
        lineHeight = 16.sp,
        letterSpacing = 0.5.sp
    )
    */
)
```

Código-fonte 6 – Registro da fonte Righteous
Fonte: Elaborado pelo autor (2023)

O que fizemos foi criar uma variável global chamada “Righteous” que guarda a nova fonte. A partir de agora já podemos utilizar essa fonte em nosso aplicativo.

Vamos retornar ao arquivo “***MainActivity.kt***” para trocar a fonte do segundo *Text* da nossa aplicação. Ao concluir, o código fonte da função “***BasicComponentsScreen***” deverá se parecer com a listagem abaixo:

```
@Composable
fun BasicComponentsScreen() {
  Column(modifier = Modifier
    .fillMaxWidth()
    .background(Color.Black)) {
    Text(
      text = "FIAP",
      fontSize = 32.sp,
      fontWeight = FontWeight.Bold,
      color = Color(237, 20, 91),
      fontFamily = FontFamily.Serif
    )
    Text(
      text = "Desenvolvendo aplicações Android",
      fontSize = 16.sp,
      fontWeight = FontWeight.Bold,
      color = Color.White,
      fontFamily = Righteous
    )
  }
}
```

Código-fonte 7 – Troca da fonte do subtítulo
Fonte: Elaborado pelo autor (2023)

Ao rodarmos nosso aplicativo no emulador, o texto deve apresentar a nova fonte, conforme mostrado na figura “Função “Subtítulo com a fonte Righteous”:

Figura 17 – Subtítulo com a fonte Righteous
Fonte: Elaborado pelo autor (2023)

2 ALINHANDO O NOSSO TEXTO

Outro ajuste importante é o alinhamento do texto no interior do componente *Text*. Para percebermos se o alinhamento está ocorrendo, vamos acrescentar uma cor de fundo ao *Text* "**FIAP**". Neste momento é importante entendermos uma coisa. Então, altere o código fonte da função "**BasicComponentsScreen**" para que fique igual a listagem abaixo:

```
@Composable
fun BasicComponentsScreen() {
  Column(modifier = Modifier
    .fillMaxWidth()
    .background(Color.Black)) {
    Text(
      text = "FIAP",
      fontSize = 32.sp,
      fontWeight = FontWeight.Bold,
      color = Color(237, 20, 91),
      fontFamily = FontFamily.Serif,
      modifier = Modifier.background(Color.Yellow)
    )
    Text(
      text = "Desenvolvendo aplicações Android",
      fontSize = 16.sp,
      fontWeight = FontWeight.Bold,
      color = Color.White,
      fontFamily = Righteous
    )
  }
}
```

Código-fonte 8 – Formatação de background do título
Fonte: Elaborado pelo autor (2023)

Utilizamos o "**Modifier.background**", para trocar a cor de fundo do *Text*, que era transparente para amarelo. O resultado deve se parecer com a **figura "**Código-fonte da função "Preview do background do Text título":

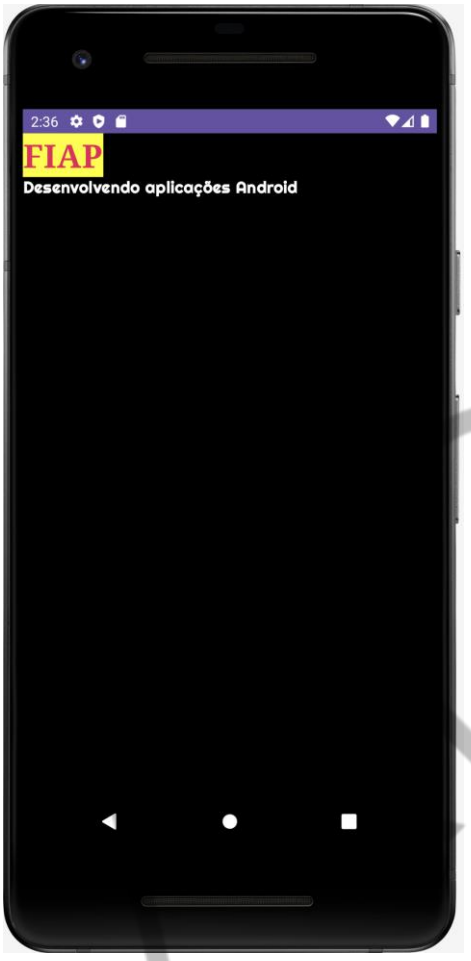

Figura 18 – Preview do background do Text título
Fonte: Elaborado pelo autor (2023)

O que aconteceu? Pare e pense um pouquinho antes de continuar lendo.

Isso mesmo! O *Text* tem exatamente o tamanho do texto que ele contém. Então se alinharmos o texto não vamos perceber nenhuma alteração. Vamos mudar este comportamento com o "**Modifier.fillMaxWidth**", que fará com que o nosso *Text* tenha a largura total da tela. Seu código deve se parecer com a listagem abaixo:

```
@Composable
fun BasicComponentsScreen() {
  Column(modifier = Modifier
    .fillMaxWidth()
    .background(Color.Black)) {
    Text(
      text = "FIAP",
      fontSize = 32.sp,
      fontWeight = FontWeight.Bold,
      color = Color(237, 20, 91),
      fontFamily = FontFamily.Serif,
      modifier = Modifier
        .background(Color.Yellow)
        .fillMaxWidth()
    )
    Text(
      text = "Desenvolvendo aplicações Android",
      fontSize = 16.sp,
      fontWeight = FontWeight.Bold,
      color = Color.White,
      fontFamily = Righteous
    )
  }
}
```

Código-fonte 9 – Ajuste da largura do Text do título
Fonte: Elaborado pelo autor (2023)

Execute a aplicação no emulador. O resultado esperado deverá ser como o mostrado na **figura "**Preview da nova largura do Text título":

Figura 19 – Preview da nova largura do Text título
Fonte: Elaborado pelo autor (2023)

Agora podemos alinhar o nosso *Text* e perceber o resultado. Vamos alinhar o texto para o lado final/direito do componente *Text*. Ajuste o seu código para que ele se pareça com a listagem abaixo:

```
@Composable
fun BasicComponentsScreen() {
  Column(modifier = Modifier
    .fillMaxWidth()
    .background(Color.Black)) {
    Text(
      text = "FIAP",
      fontSize = 32.sp,
      fontWeight = FontWeight.Bold,
      color = Color(237, 20, 91),
      fontFamily = FontFamily.Serif,
      modifier = Modifier
        .background(Color.Yellow)
        .fillMaxWidth(),
      textAlign = TextAlign.End
    )
    Text(
      text = "Desenvolvendo aplicações Android",
      fontSize = 16.sp,
      fontWeight = FontWeight.Bold,
      color = Color.White,
      fontFamily = Righteous
    )
  }
}
```

Código-fonte 10 – Alteração do alinhamento do Text de título
Fonte: Elaborado pelo autor (2023)

Execute a aplicação novamente. O resultado deverá ser como o apresentado na **figura "**Preview do alinhamento à direita do Text título":

Figura 20 – Preview do alinhamento à direita do Text título
Fonte: Elaborado pelo autor (2023)

O “**textAlign**” é um parâmetro do *composable Text* que alinha o texto no seu interior. Além de “End” temos também o ***Start***, ***Justify*** e ***Center***. Teste cada um deles e observe o comportamento do texto no interior do *Text*.

Para alinhar o *composable Text* em relação ao seu componente pai, precisamos utilizar o “**Modifier**”. Vamos centralizar o *Text* do subtítulo no centro da *Column*. Seu código deverá se parecer com a listagem abaixo:

```
@Composable
fun BasicComponentsScreen() {
  Column(modifier = Modifier
    .fillMaxWidth()
    .background(Color.Black)) {
    Text(
      text = "FIAP",
      fontSize = 32.sp,
      fontWeight = FontWeight.Bold,
      color = Color(237, 20, 91),
      fontFamily = FontFamily.Serif,
      modifier = Modifier
        .background(Color.Yellow)
        .fillMaxWidth(),
      textAlign = TextAlign.End
    )
    Text(
      text = "Desenvolvendo aplicações Android",
      fontSize = 16.sp,
      fontWeight = FontWeight.Bold,
      color = Color.White,
      fontFamily = Righteous,
      modifier = Modifier.align(Alignment.CenterHorizontally)
    )
  }
}
```

Código-fonte 11 – Reposicionamento do Text de subtítulo
Fonte: Elaborado pelo autor (2023)

Ao executar o aplicativo, o resultado esperado deverá ser como o mostrado na **figura** "Preview do novo posicionamento do Text de subtítulo":

Figura 21 – Preview do novo posicionamento do Text de subtítulo
Fonte: Elaborado pelo autor (2023)

Atenção, no exemplo acima, não alinhamos o texto no centro, alinhamos todo o composable *Text* no centro do *composable* pai, que é a *Column*. Treine bastante para entender a diferença entre um e outro 😁.

2.1 Entrada de dados do usuário

Quando falamos de aplicativos, uma das primeiras coisas que nos vem a mente são os formulários que preenchemos para um cadastro, uma pesquisa etc. Então, tornar o nosso aplicativo capaz de receber dados do usuário é uma das funcionalidades mais importantes.

Há diversos composables que podemos utilizar para que o usuário forneça dados ao nosso aplicativo. Os mais utilizados são os campos de texto editáveis, as caixas de checagem, botões rádio e listas suspensas.

Chegou o momento de tornar o nosso aplicativo mais interativo através destes componentes. Vamos?

2.2 Caixa de texto editável

Um dos componentes mais importantes para entrada de dados do usuário são as caixas de texto editáveis, ou seja, que permitem ao usuário alterar o seu conteúdo. No Jetpack Compose temos várias opções para esta finalidade, que vão desde estilos mais simples até os mais sofisticados.

Antes de começarmos a incluir os novos componentes, faremos alguns ajustes nas versões do **Kotlin** e da biblioteca “**material3**” do Jetpack Compose.

Abra o arquivo “**build.gradle**” em nível de projeto. Este arquivo se encontra na pasta “***Gradle Scripts***”, de acordo com a **figura “**Localização do arquivo build.gradle do projeto”:

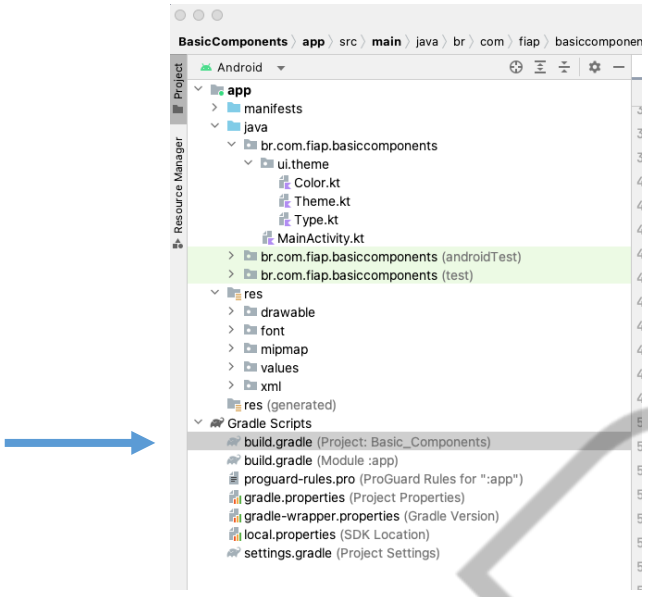

Figura 22 – Localização do arquivo build.gradle do projeto
Fonte: Elaborado pelo autor (2023)

Por padrão, o Android Studio está configurado para trabalhar com o *plugin* do **Kotlin 1.7.20**, vamos alterar para a versão **1.8.0**. Edite o arquivo build.gradle do projeto, de modo que fique como a listagem abaixo:

```
// Top-level build file where you can add configuration options common to
all sub-projects/modules.
plugins {
    id 'com.android.application' version '8.0.0' apply false
    id 'com.android.library' version '8.0.0' apply false
    //id 'org.jetbrains.kotlin.android' version '1.7.20' apply false
    id 'org.jetbrains.kotlin.android' version '1.8.0' apply false
}
```

Código-fonte 12 – Atualização de versão do plugin do Kotlin
Fonte: Elaborado pelo autor (2023)

Abra o arquivo “**build.gradle**” do módulo, de acordo com a **figura “**Localização do arquivo build.gradle do módulo”:

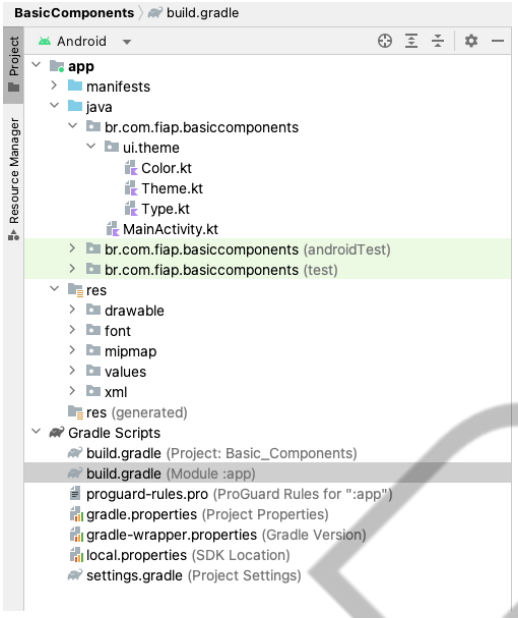

Figura 23 – Localização do arquivo build.gradle do módulo
Fonte: Elaborado pelo autor (2023)

O arquivo de configuração “**build.gradle**” da pasta módulo, é essencial para o processo de “build” de uma aplicação Android. Neste arquivo estão todas as informações necessárias para compilação, empacotamento e geração do pacote de instalação final do aplicativo. Mais adiante estudaremos estes arquivos mais profundamente.

Por ora, vamos alterar a versão das extensões do compilador Kotlin do Jetpack Compose. Localize o bloco “**composeOptions**” e altere a versão do atributo “**kotlinCompilerExtensionsVersion**” para **1.4.0**. No bloco “***dependencies***”, acrescente a versão do pacote “**material3**”. Seu código deverá se parecer com a listagem abaixo:

```
composeOptions {
        //kotlinCompilerExtensionVersion '1.3.2'
        kotlinCompilerExtensionVersion '1.4.0'
    }
    packagingOptions {
        resources {
            excludes += '/META-INF/{AL2.0,LGPL2.1}'
        }
    }
}

dependencies {

    implementation 'androidx.core:core-ktx:1.8.0'
    implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.3.1'
    implementation 'androidx.activity:activity-compose:1.5.1'
    implementation platform('androidx.compose:compose-bom:2022.10.00')
    implementation 'androidx.compose.ui:ui'
    implementation 'androidx.compose.ui:ui-graphics'
    implementation 'androidx.compose.ui:ui-tooling-preview'
//    implementation 'androidx.compose.material3:material3'
    implementation 'androidx.compose.material3:material3:1.1.0'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'
    androidTestImplementation platform('androidx.compose:compose-
bom:2022.10.00')
    androidTestImplementation 'androidx.compose.ui:ui-test-junit4'
    debugImplementation 'androidx.compose.ui:ui-tooling'
    debugImplementation 'androidx.compose.ui:ui-test-manifest'
}
```

Código-fonte 13 – Configuração do arquivo build.gradle do módulo
Fonte: Elaborado pelo autor (2023)

Após os ajustes clique em “***Sync Now***”, de acordo com a **figura “**Botão de sincronização do Gradle”:

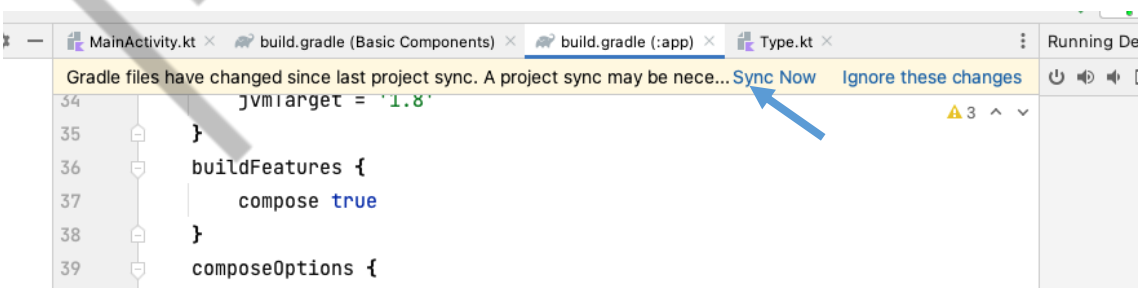

Figura 24 – Botão de sincronização do Gradle
Fonte: Elaborado pelo autor (2023)

Aguarde o Android Studio concluir o download de todas as novas bibliotecas.

Após estes ajustes, já estamos prontos para começar a implementar as novas funcionalidades da nossa aplicação. Vamos lá!

O composable que permite ao usuário digitar dados é o "***TextField***". Vamos inserir ao nosso projeto este *composable*. A função "**BasicComponentesScreen**" deverá ficar como na listagem abaixo:

```
@Composable
fun BasicComponentsScreen() {
  Column(modifier = Modifier
    .fillMaxWidth()
    .background(Color.Black)) {
    Text(
      text = "FIAP",
      fontSize = 32.sp,
      fontWeight = FontWeight.Bold,
      color = Color(237, 20, 91),
      fontFamily = FontFamily.Serif,
      modifier = Modifier
        .background(Color.Yellow)
        .fillMaxWidth(),
      textAlign = TextAlign.End
    )
    Text(
      text = "Desenvolvendo aplicações Android",
      fontSize = 16.sp,
      fontWeight = FontWeight.Bold,
      color = Color.White,
      fontFamily = Righteous,
      modifier = Modifier.align(Alignment.CenterHorizontally)
    )
    TextField(value = "", onValueChange = {})
  }
}
```

Código-fonte 14 – Acréscimo do componente TextField
Fonte: Elaborado pelo autor (2023)

O "TextField" é um novo componente em nossa IU. Execute o aplicativo no emulador. O resultado deve ser como o apresentado na **figura "**Preview do componente TextField":

Figura 25 – Preview do componente TextField
Fonte: Elaborado pelo autor (2023)

Foi inserido um componente “TextField” que ocupa uma parte horizontal da tela. Podemos utilizar o “Modifier” para que ele ocupe toda a largura da tela. Seu código deverá se parecer com o da figura “Digitar dados TextField”:

```
@Composable
fun BasicComponentsScreen() {
  Column(modifier = Modifier
    .fillMaxWidth()
    .background(Color.Black)) {
    Text(
      text = "FIAP",
      fontSize = 32.sp,
      fontWeight = FontWeight.Bold,
      color = Color(237, 20, 91),
      fontFamily = FontFamily.Serif,
      modifier = Modifier
        .background(Color.Yellow)
        .fillMaxWidth(),
      textAlign = TextAlign.End
    )
    Text(
      text = "Desenvolvendo aplicações Android",
      fontSize = 16.sp,
      fontWeight = FontWeight.Bold,
      color = Color.White,
      fontFamily = Righteous,
      modifier = Modifier.align(Alignment.CenterHorizontally)
    )
    TextField(
      value = "",
      onValueChange = {},
      modifier = Modifier.fillMaxWidth()
    )
  }
}
```

Código-fonte 15 – Ajuste da largura do componente TextField
Fonte: Elaborado pelo autor (2023)

Se executarmos novamente nosso aplicativo, ele deverá se parecer com a **figura "**Preview da nova largura do TextField":

Figura 26 – Preview da nova largura do TextField
Fonte: Elaborado pelo autor (2023)

O "**TextField**" possui dois parâmetros obrigatórios, o "***value***" e o "***onValueChange***". O "**value**" é o valor que será colocado dentro do nosso "**TextField**". O "**onValueChange**" é um parâmetro que recebe como valor uma função, por isso que dissemos que o valor deste parâmetro é "{}". Dentro desse par de chaves podemos colocar uma instrução qualquer e ela será executada quando o "**value**" do "**TextField**" for alterado.

Vamos trocar o valor do parâmetro "**value**" para a palavra "Android". O código da declaração do "**TextField**" deverá ficar de acordo com a listagem abaixo:

```
TextField(
  value = "Android",
  onValueChange = {},
  modifier = Modifier.fillMaxWidth()
)
```

Código-fonte 16 – Atribuição de valor ao parâmetro value do TextField
Fonte: Elaborado pelo autor (2023)

Ao executar a aplicação o resultado deverá ser de acordo com a **figura "**Preview do parâmetro value do TextField":

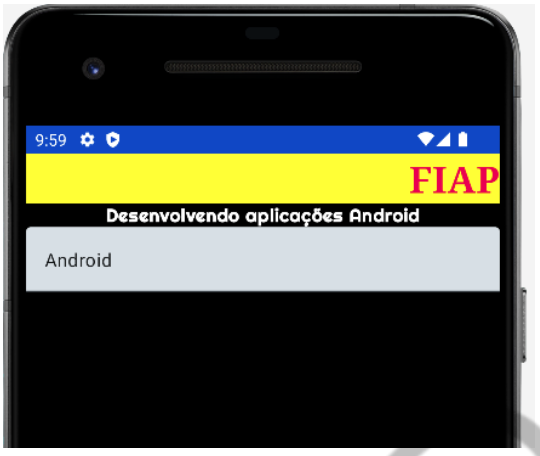

Figura 27 – Preview do parâmetro value do TextField
Fonte: Elaborado pelo autor (2023)

O valor atribuído ao parâmetro “**value**” está sendo exibido em nosso “**TextField**”.

2.3 Gerenciando o estado do TextField

O objetivo do “**TextField**” é permitir que o usuário possa digitar algum valor, mas ao clicar no “**TextField**”, apesar de acionarmos o teclado do dispositivo, conforme a **figura “**Teclado virtual do Android”, nada acontece quando digitamos. Por que isso acontece? Para respondermos esta pergunta vamos ter que nos lembrar do conceito de “**state**”.

Figura 28 – Teclado virtual do Android
Fonte: Elaborado pelo autor (2023)

O estado em um aplicativo é qualquer valor que pode mudar ao longo do tempo. Quando estamos olhando para uma Interface de Usuário no Android estamos observando o seu estado atual. Se um valor mudar, precisamos que a Interface também seja atualizada, ou seja, o estado mudou!

Quando digitamos algo no "***TextField***" estamos alterando o seu "value" e essa mudança de estado causa a recomposição do "***TextField***". O problema é que durante essa recomposição os parâmetros são carregados novamente e o "value" retorna ao seu valor inicial. Assim, é necessário armazenarmos esse valor que está sendo digitado para que possamos nos lembrar dele na próxima recomposição. É aí que entra o "state".

Para que o Android se "lembre" dos valores entre recomposições, utilizamos a função "**mutableStateOf()**".

No início da função de composição "**BasicComponentsScreen**()", vamos criar a variável de estado que armazenará o "**value**" do "**TextField**". No início da função, vamos acrescentar a seguinte instrução:

```
var textFieldValue = remember {
  mutableStateOf("")
}
```

Código-fonte 17 – Declaração da variável de estado textFieldValue
Fonte: Elaborado pelo autor (2023)

Nossa variável "textFieldValue" é uma variável de estado, e o seu valor será lembrado entre as recomposições.

Agora, vamos ajustar os parâmetros "value" e "onValueChange" do "TextField" para que possamos utilizá-lo. Ao final, o "TextField" deverá estar como o da listagem abaixo:

```
TextField(
  value = textFieldValue.value,
  onValueChange = { novoValor ->
    textFieldValue.value = novoValor
  },
  modifier = Modifier.fillMaxWidth()
)
```

Código-fonte 18 – Atribuindo a variável de estado ao TextField
Fonte: Elaborado pelo autor (2023)

Ao ocorrer a composição inicial do “**TextField**” ele receberá o valor vazio, que é o valor de inicialização da variável de estado “**textFieldValue**”. A cada caractere digitado o método “**onValueChange**” do “**TextField**” nos retorna o valor atual que será atribuído à variável “**textFieldValue**”, que está armazenado na variável “**novoValor**” e que provocará a recomposição do “**TextField**”, pois seu estado mudou. Mas agora o valor será lembrado, e o value do “**TextField**” terá o comportamento que desejamos.

O código final da função de recomposição “BasicComponentsScreen” deverá se parecer com a listagem abaixo:

```
@Composable
fun BasicComponentsScreen() {

  var textFieldValue = remember {
    mutableStateOf("")
  }

  Column(
    modifier = Modifier
      .fillMaxWidth()
      .background(Color.Black)
  ) {
    Text(
      text = "FIAP",
      fontSize = 32.sp,
      fontWeight = FontWeight.Bold,
      color = Color(237, 20, 91),
      fontFamily = FontFamily.Serif,
      modifier = Modifier
        .background(Color.Yellow)
        .fillMaxWidth(),
      textAlign = TextAlign.End
    )
    Text(
      text = "Desenvolvendo aplicações Android",
      fontSize = 16.sp,
      fontWeight = FontWeight.Bold,
      color = Color.White,
      fontFamily = Righteous,
      modifier = Modifier.align(Alignment.CenterHorizontally)
    )
    TextField(
      value = textFieldValue.value,
      onValueChange = { novoValor ->
        textFieldValue.value = novoValor
      },
      modifier = Modifier.fillMaxWidth()
    )
  }
}
```

Código-fonte 19 – Função BasicComponentsScreen
Fonte: Elaborado pelo autor (2023)

Execute a aplicação no emulador e digite alguma coisa!

2.4 Tipos de entrada

Quando focamos em um “**TextField**” o Android abre o teclado virtual para que possamos inserir o nosso texto. Há diversos teclados para a digitação dos mais variados tipos de informação. É indicado que o teclado fornecido pelo Android esteja de acordo com o tipo de informação que vamos digitar. É muito chato ter que ficar trocando de teclado no momento da digitação.

Vamos acrescentar um novo “**TextField**” que será utilizado para digitarmos uma quantidade qualquer, ou seja, valor numérico, então, seria ideal se o teclado apresentado fosse o numérico. Adicione, após o primeiro “**TextField**”, o código necessário para adicionarmos este novo “**TextField**”. O seu código deverá se parecer com o da listagem abaixo:

```
var quantidade = remember {
  mutableStateOf("")
}
TextField(
  value = "${quantidade.value}",
  onValueChange = { novoValor ->
    quantidade.value = novoValor
  },
  keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Number)
)
```

Código-fonte 20 – Utilização do KeyboardOptions para entrada de números
Fonte: Elaborado pelo autor (2023)

A função “**KeyboardOptions**”, permite que alteremos o teclado exibido para o usuário, e o comportamento enquanto digitamos. Neste exemplo, o parâmetro “**keyboardType**”, da função “**KeyboardOptions**” diz ao Android que o teclado que será exibido deverá ser o numérico.

Execute o seu aplicativo no emulador. Ao focar no segundo “**TextField**” o teclado numérico deverá ser aberto. O resultado esperado deve se parecer com a **figura “**Preview do KeyboardOptions para entrada de números”**:**

Figura 29 – Preview do KeyboardOptions para entrada de números
Fonte: Elaborado pelo autor (2023)

Há outras opções além de “**Number**”. É possível fornecer os seguintes “**KeyboardType**”:

Number: apresentará o teclado numérico.

Text: apresentará o tecado alfanumérico.

Decimal: apresentará o teclado numérico com teclas para ponto decimal.

Email: apresentará o teclado com o caractere “@”.

NumberPassword: apresenta o teclado numérico e não vemos os números digitados.

Password: apresenta o teclado alfanumérico e não vemos o que estamos digitando.

Phone: apresenta o teclado para discagem.

Uri: fornece o teclado ideal para digitarmos um endereço de Internet, por exemplo.

Experimente cada um deles e observe o resultado!

Também é possível definirmos como o texto será inserido. Suponhamos que o primeiro “TextField” seja utilizado para inserirmos o nome completo de uma pessoa, neste caso seria interessante que ao digitarmos o espaço o teclado fique maiúsculo

para digitarmos o sobrenome. Ajuste o seu primeiro “TextField” de modo que o código se pareça com a listagem abaixo:

```
TextField(
  value = textFieldValue.value,
  onValueChange = { novoValor ->
    textFieldValue.value = novoValor
  },
  modifier = Modifier.fillMaxWidth(),
  keyboardOptions = KeyboardOptions(
    capitalization = KeyboardCapitalization.Words
  )
)
```

Código-fonte 21 – KeyboardOptions para palavras com iniciais maiúsculas
Fonte: Elaborado pelo autor (2023)

Execute o seu aplicativo, ao focar no primeiro “TextField” o teclado maiúsculo deverá ser disponibilizado. Digite seu nome e sobrenome para experimentar o recurso. Seu aplicativo deverá se parecer com a figura “Preview do teclado maiúsculo para cada palavra”:

Figura 30 – Preview do teclado maiúsculo para cada palavra
Fonte: Elaborado pelo autor (2023)

Além de “Words”, há as opções “**Characters**”, “**None**” e “**Sentences**”. Experimente cada um deles e observe as mudanças que causam no teclado e no comportamento da digitação.

2.5 Dicas de entrada

É bastante comum quando vamos preencher um formulário, que esse nos apresente algum texto de ajuda conhecido como “placeholder”. Vamos adicionar um “***placeholder***” ao nosso segundo “***TextField***”. Seu código deverá ficar de acordo com a listagem abaixo:

```
TextField(
  value = "${quantidade.value}",
  onValueChange = { novoValor ->
    quantidade.value = novoValor
  },
  keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Number),
  placeholder = {
    Text(text = "Qual a quantidade?")
  }
)
```

Código-fonte 22 – Utilização do placeholder
Fonte: Elaborado pelo autor (2023)

Note que o “***placeholder***” é um *composable*, então, este parâmetro pode receber como valor o “***Text***” que será usado para exibir o texto.

Execute o seu aplicativo no emulador. O resultado deve ser parecido com o exibido na **figura “**Preview do parâmetro placeholder do TextField”:

Figura 31 – Preview do parâmetro placeholder do TextField
Fonte: Elaborado pelo autor (2023)

O texto do “placeholder” é substituído pelo conteúdo digitado pelo usuário.

Outro recurso de ajuda bastante interessante é o “**label**”. Este parâmetro inicialmente parece um “**placeholder**”, mas ao clicarmos no “TextField” o texto será usado como uma etiqueta do nosso “**TextField**”.

Vamos adicionar o parâmetro “**label**” ao nosso primeiro “**TextField**”. O código do “**TextField**” deverá estar como o da listagem abaixo:

```
TextField(
  value = textFieldValue.value,
  onValueChange = { novoValor
    textFieldValue.value = novoValor
  },
  modifier = Modifier.fillMaxWidth(),
  keyboardOptions = KeyboardOptions(
    capitalization = KeyboardCapitalization.Words
  ),
  label = {
    Text(text = "Nome e sobrenome")
  }
)
```

Código-fonte 23 – Utilização do parâmetro label do TextField
Fonte: Elaborado pelo autor (2023)

Ao executar o seu aplicativo em um emulador, ele deverá se parecer com a **figura “**Preview do parâmetro label do TextField”:

Figura 32 – Preview do parâmetro label do TextField
Fonte: Elaborado pelo autor (2023)

1.1.1.1 Inserindo ícone ao TextField

Outra forma de chamar a atenção do usuário sobre o dado que deve ser inserido é com a utilização de ícones. Podemos colocar ícones no início ou no fim do

"**TextField**". Vamos adicionar um ícone no lado inicial do nosso primeiro "**TextField**", seu código deverá ser parecer com o da listagem abaixo:

```
TextField(
  value = textFieldValue.value,
  onValueChange = { novoValor
    textFieldValue.value = novoValor
  },
  modifier = Modifier.fillMaxWidth(),
  keyboardOptions = KeyboardOptions(
    capitalization = KeyboardCapitalization.Words
  ),
  label = {
    Text(text = "Nome e sobrenome")
  },
  leadingIcon = {
    Icon(
      imageVector = Icons.Default.Person,
      contentDescription = "",
      tint = Color(237, 20, 91)
    )
  }
)
```

Código-fonte 24 – Uso do parâmetro leadingIcon do TextField
Fonte: Elaborado pelo autor (2023)

Com o parâmetro "**leadingIcon**" colocamos o ícone no início do "**TextField**". Você também pode utilizar o "**trailingIcon**", que posicionará o ícone no fim do "**TextField**". Experimente!

Ao executar a sua aplicação, a sua IU deverá se parecer com a **figura "**Preview do parâmetro leadingIcon do TextField":

Figura 33 – Preview do parâmetro leadingIcon do TextField
Fonte: Elaborado pelo autor (2023)

2.6 Alterando a cor do texto de um TextField

É possível mudar a cor de um texto no “**TextField**”, mas não simplesmente isso. Quando o usuário está focado no “**TextField**” a cor pode ser diferente de quando o usuário não está focado no “TextField”, podemos alterar a cor do “**placeholder**”, do “**label**”, etc.

Vamos alterar algumas configurações de cor no segundo “TextField”, seu código deverá se parecer com a listagem abaixo:

```
TextField(
  value = "${quantidade.value}",
  onValueChange = { novoValor ->
    quantidade.value = novoValor
  },
  keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Number),
  placeholder = {
    Text(text = "Qual a quantidade?")
  },
  colors = TextFieldDefaults.colors(
    focusedTextColor = Color.White,
    unfocusedTextColor = Color.Green,
    unfocusedPlaceholderColor = Color.Magenta
  )
)
```

Código-fonte 25 – Utilização do parâmetro colors do TextField
Fonte: Elaborado pelo autor (2023)

O parâmetro “**focusedTextColor**” define a cor do texto para branco enquanto o editamos. O “**unfocusedTextColor**” altera a cor do texto para verde quando saímos do “TextField” e o “**unfocusedPlaceholderColor**” altera a cor do “placeholder” para magenta.

Execute o aplicativo novamente. Sua IU deverá se parecer com a **figura** “Preview do parâmetro colors do TextField”:

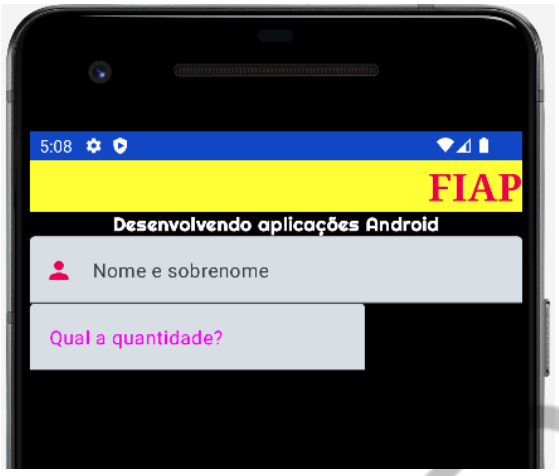

Figura 34 – Preview do parâmetro colors do TextField
Fonte: Elaborado pelo autor (2023)

Há muitas possibilidades para ajuste de cores no texto do “**TextField**”. Para obter uma lista de todas as opções possíveis, aponte o mouse para a palavra “colors” da função “**TextFieldDefaults**”. Você deverá ver uma caixa de diálogo de acordo com a figura “Lista de parâmetros da função colors”.

```
public final fun colors(
    focusedTextColor: Color,
    unfocusedTextColor: Color,
    disabledTextColor: Color,
    errorTextColor: Color,
    focusedContainerColor: Color,
    unfocusedContainerColor: Color,
    disabledContainerColor: Color,
    errorContainerColor: Color,
    cursorColor: Color,
    errorCursorColor: Color,
    selectionColors: TextSelectionColors,
    focusedIndicatorColor: Color,
    unfocusedIndicatorColor: Color,
    disabledIndicatorColor: Color,
    errorIndicatorColor: Color,
    focusedLeadingIconColor: Color,
    unfocusedLeadingIconColor: Color,
    disabledLeadingIconColor: Color,
    errorLeadingIconColor: Color,
    focusedTrailingIconColor: Color,
    unfocusedTrailingIconColor: Color,
    disabledTrailingIconColor: Color,
    errorTrailingIconColor: Color,
    focusedLabelColor: Color,
    unfocusedLabelColor: Color,
    disabledLabelColor: Color,
    errorLabelColor: Color,
    focusedPlaceholderColor: Color,
    unfocusedPlaceholderColor: Color,
    disabledPlaceholderColor: Color,
    errorPlaceholderColor: Color,
    focusedSupportingTextColor: Color,
    unfocusedSupportingTextColor: Color,
    disabledSupportingTextColor: Color,
    errorSupportingTextColor: Color,
    focusedPrefixColor: Color,
    unfocusedPrefixColor: Color,
    disabledPrefixColor: Color,
    errorPrefixColor: Color,
    focusedSuffixColor: Color,
    unfocusedSuffixColor: Color,
    disabledSuffixColor: Color,
    errorSuffixColor: Color
```

Figura 35 – Lista de parâmetros da função colors
Fonte: Elaborado pelo autor (2023)

Nós experimentamos apenas 3 opções. Explore todas essas possiblidades, seja curioso e observe o resultado das alterações.

2.7 OutlinedTextField

Até agora estávamos utilizando o “**TextField**”, mas há uma variação dele que é o “**OutlinedTextField**”, que implementa uma aparência diferente, mas todos os parâmetros que vimos até agora estão disponíveis neste componente também.

Vamos criar um “**OutlinedTextField**” após o segundo “**TextField**”. Seu código deverá se parecer como o da listagem abaixo:

```
@Composable
fun BasicComponentsScreen() {

  var textFieldValue = remember {
    mutableStateOf("")
  }

  Column(
    modifier = Modifier
      .fillMaxWidth()
      .background(Color.Black)
  ) {
    Text(
      text = "FIAP",
      fontSize = 32.sp,
      fontWeight = FontWeight.Bold,
      color = Color(237, 20, 91),
      fontFamily = FontFamily.Serif,
      modifier = Modifier
        .background(Color.Yellow)
        .fillMaxWidth(),
      textAlign = TextAlign.End
    )
    Text(
      text = "Desenvolvendo aplicações Android",
      fontSize = 16.sp,
      fontWeight = FontWeight.Bold,
      color = Color.White,
      fontFamily = Righteous,
      modifier = Modifier.align(Alignment.CenterHorizontally)
    )
    TextField(
      value = textFieldValue.value,
      onValueChange = { novoValor ->
        textFieldValue.value = novoValor
      },
      modifier = Modifier.fillMaxWidth(),
      keyboardOptions = KeyboardOptions(
        capitalization = KeyboardCapitalization.Words
      ),
      label = {
        Text(text = "Nome e sobrenome")
      },
      leadingIcon = {
        Icon(
          imageVector = Icons.Default.Person,
          contentDescription = "",
          tint = Color(237, 20, 91)
        )
```

```
        }
      )
      var quantidade = remember {
        mutableStateOf("")
      }
      TextField(
        value = "${quantidade.value}",
        onValueChange = { novoValor ->
          quantidade.value = novoValor
        },
        keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Number),
        placeholder = {
          Text(text = "Qual a quantidade?")
        },
        colors = TextFieldDefaults.colors(
          focusedTextColor = Color.White,
          unfocusedTextColor = Color.Green,
          unfocusedPlaceholderColor = Color.Magenta
        )
      )
      Spacer(modifier = Modifier.height(32.dp))
      var cidade = remember {
        mutableStateOf("")
      }
      OutlinedTextField(
        value = cidade.value,
        onValueChange = {
          cidade.value = it
        },
        modifier = Modifier
          .padding(16.dp)
          .fillMaxWidth(),
        textStyle = TextStyle(color = Color.White)
      )
    }
}
```

Código-fonte 26 – Utilização do OutlinedTextField
Fonte: Elaborado pelo autor (2023)

Neste exemplo, utilizamos um componente “**Spacer**” com o modificador “**Modifier.height(32.dp)**”, que colocará um espaço de “32.dp”, antes do “**OutlinedTextField**”.

No “**OutlinedTextField**”, configuramos o seu modificador para que ele tenha a largura total da tela com “**fillMaxWidth**”, um espaçamento nos quatro lados de **16.dp** com o “**padding(16.dp)**”. Além disso, utilizamos o “**textStyle**” para modificar a cor do texto.

Outra coisa que mudamos foi o parâmetro “**onValueChange**”, que usamos a variável “**it**”, é o valor que nos é passado quando digitamos algo. Nos exemplos anteriores, utilizamos uma função “**lambda**” para isso, onde recebíamos uma variável que chamamos de “**novoValor**”. O uso do “**it**” é muito utilizado em Kotlin, pois reduz digitação.

Com “**textStyle**” é possível modificar todas os parâmetros listados na **figura**

```
TextStyle(color = Color.White)
```

```
public constructor TextStyle(
    color: Color,
    fontSize: TextUnit,
    fontWeight: FontWeight?,
    fontStyle: FontStyle?,
    fontSynthesis: FontSynthesis?,
    fontFamily: FontFamily?,
    fontFeatureSettings: String?,
    letterSpacing: TextUnit,
    baselineShift: BaselineShift?,
    textGeometricTransform: TextGeometricTransform?,
    localeList: LocaleList?,
    background: Color,
    textDecoration: TextDecoration?,
    shadow: Shadow?,
    textAlign: TextAlign?,
    textDirection: TextDirection?,
    lineHeight: TextUnit,
    textIndent: TextIndent?,
    platformStyle: PlatformTextStyle?,
    lineHeightStyle: LineHeightStyle?,
    lineBreak: LineBreak?,
    hyphens: Hyphens?
)
```

Figura 36 – Lista de parâmetros da classe TextStyle
Fonte: Elaborado pelo autor (2023)

Execute o aplicativo no emulador. O resultado esperado deverá ser como o exibido na **figura “**Parâmetro “**textStyle**”:

Figura 37 – Preview do OutlinedTextField
Fonte: Elaborado pelo autor (2023)

O “**OutlinedTextField**” é renderizado numa forma retangular com uma borda e sem preenchimento, diferentemente do “**TextField**”.

Vamos modificar a forma do “**OutlinedTextField**” para que ele se pareça com a **figura “**Parâmetro “**OutlinedTextField**”:

Figura 38 – Estilo do OutlinedTextField proposto
Fonte: Elaborado pelo autor (2023)

Para que nosso “**OutlinedTextField**”, tenha a aparência da figura “Parâmetro “**OutlinedTextField**”, seu código deverá ser como a listagem abaixo:

```
OutlinedTextField(
  value = cidade.value,
  onValueChange = {
    cidade.value = it
  },
  modifier = Modifier
    .padding(16.dp)
    .fillMaxWidth(),
  textStyle = TextStyle(color = Color.White),
  shape = RoundedCornerShape(16.dp),
  colors = OutlinedTextFieldDefaults.colors(
    unfocusedBorderColor = Color.Yellow,
    focusedBorderColor = Color.Cyan
  )
)
```

Código-fonte 27 – Utilização dos parâmetros shape e colors do OutlinedTextField
Fonte: Elaborado pelo autor (2023)

No parâmetro “shape” utilizamos o valor “**RoundedCornerShape(16.dp)**”, que aplica uma forma com cantos arredondados em **16.dp** de raio.

Para mudar a cor das bordas utilizamos o parâmetro “**colors**”, que permite a mudança das cores de vários elementos do “**OutlinedTextField**”, utilizamos o parâmetro “**unfoucusedBorderColor**” com o valor amarelo, para a borda sem o foco do usuário. Ao focar será aplicado o parâmetro “**focusedBorderColor**” para ciano.

Execute o aplicativo e veja os resultados. Sua IU deve se parecer com as **figuras “**OutlinedTextField sem foco” e **“**OutlinedTextField com foco”:

Figura 39 - OutlinedTextField sem foco
Fonte: Elaborado pelo autor (2023)

Figura 40 - OutlinedTextField com foco
Fonte: Elaborado pelo autor (2023)

Experimente os parâmetros “**placeholder**”, “**label**”, “**leadIcon**” e “**trailingIcon**” e compare com o que já vimos no “**TextField**”.

Prontinho, agora você já tem algumas opções de entrada de texto para utilizar em seus aplicativos.

3 CAIXAS DE SELEÇÃO

A caixa de seleção ou “**checkbox**” é outro componente bastante utilizado na construção de uma aplicação Android. Com ela é possível que um usuário possa selecionar uma ou mais opções em uma lista.

Vamos adicionar um “**checkbox**” no final da nossa aplicação. Após nosso “**OutlinedTextField**” adicione o código abaixo:

```
Spacer(modifier = Modifier.height(32.dp))
Checkbox(
  checked = false,
  onCheckedChange = {},
  colors = CheckboxDefaults.colors(
    checkedColor = Color.White,
    uncheckedColor = Color(0xffed145b)
  )
)
```

Código-fonte 28 – Inclusão do componente Checkbox
Fonte: Elaborado pelo autor (2023)

Utilizamos um “**Spacer**” para separarmos nossa “**Checkbox**” do “**OutlinedTextField**”. Obrigatoriamente, o “**Checkbox**” deve ter os parâmetros “**checked**”, cujo valor é um booleano e representa se a caixa estará marcada (**true**) ou desmarcada (**false**). Em nosso exemplo na primeira composição a caixa estará desmarcada, pois passamos o valor “false” para “checked”.

Rode o aplicativo no emulador. O resultado deverá se parecer com a **figura “**Preview do componente Checkbox”:

Figura 41 – Preview do componente Checkbox
Fonte: Elaborado pelo autor (2023)

A “**Checkbox**” foi renderizada corretamente. Observe que a cor da caixa foi definida através do parâmetro “colors” utilizando como valor a função “**CheckboxDefaults.colors()**”. Aponte o mouse sobre “colors” e veja quais são os outros parâmetros de cor que você pode modificar 😉.

Mas se clicarmos nele nada irá ocorrer. Outra coisa importante, não temos um texto rotulando nossa caixa. Então acrescente o código da listagem abaixo para melhorarmos nossa caixa de seleção:

```
Spacer(modifier = Modifier.height(32.dp))
Row(modifier = Modifier.fillMaxWidth()) {
  Checkbox(
    checked = false,
    onCheckedChange = {},
    colors = CheckboxDefaults.colors(
      checkedColor = Color.White,
      uncheckedColor = Color(0xffed145b)
    )
  )
  Text(
    text = "Opção 1",
    color = Color.White
  )
}
```

Código-fonte 29 – Inclusão de texto para a Checkbox
Fonte: Elaborado pelo autor (2023)

Foi necessário utilizarmos uma “**Row**” para posicionarmos lado a lado a “**Checkbox**” e o texto utilizando um componente “**Text**”.

Execute o aplicativo. O resultado esperado deverá se parecer com a **figura “**Preview da Checkbox com texto desalinhado”**:**

Figura 42 – Preview da Checkbox com texto desalinhado
Fonte: Elaborado pelo autor (2023)

O texto “**Opção 1**” ficou desalinhado em relação à “**Checkbox**”. Para resolver isso vamos mexer na “**Row**”. Precisamos acertar o **alinhamento vertical**. O seu código deverá se parecer com a listagem abaixo:

```
Row(
  verticalAlignment = Alignment.CenterVertically,
  modifier = Modifier
    .fillMaxWidth()
) {
  Checkbox(
    checked = false,
    onCheckedChange = {},
    colors = CheckboxDefaults.colors(
      checkedColor = Color.White,
      uncheckedColor = Color(0xffed145b)
    )
  )
  Text(
    text = "Opção 1",
    color = Color.White
  )
}
```

Código-fonte 30 – Alinhamento da Row da Checkbox
Fonte: Elaborado pelo autor (2023)

O resultado deverá ser parecido com a **figura** **“**Preview da Checkbox com texto alinhado verticalmente”:

Figura 43 – Preview da Checkbox com texto alinhado verticalmente
Fonte: Elaborado pelo autor (2023)

3.1 Gerenciando o estado da caixa de seleção

Vamos inserir mais duas caixas de texto. Para isso somente copie e cole a caixa de texto que já fizemos e atualize o seu texto. Vamos fazer uma lista de linguagens de programação. Seu código deverá se parecer com a listagem abaixo:

```
Spacer(modifier = Modifier.height(32.dp))
Row(
  verticalAlignment = Alignment.CenterVertically,
  modifier = Modifier
    .fillMaxWidth()
) {
  Checkbox(
    checked = false,
    onCheckedChange = {},
    colors = CheckboxDefaults.colors(
      checkedColor = Color.White,
      uncheckedColor = Color(0xffed145b)
    )
  )
  Text(
    text = "Kotlin",
    color = Color.White
  )
}
Row(
  verticalAlignment = Alignment.CenterVertically,
```

```
    modifier = Modifier
      .fillMaxWidth()
) {
    Checkbox(
      checked = false,
      onCheckedChange = {},
      colors = CheckboxDefaults.colors(
        checkedColor = Color.White,
        uncheckedColor = Color(0xffed145b)
      )
    )
    Text(
      text = "Java",
      color = Color.White
    )
}
Row(
    verticalAlignment = Alignment.CenterVertically,
    modifier = Modifier
      .fillMaxWidth()
) {
    Checkbox(
      checked = false,
      onCheckedChange = {},
      colors = CheckboxDefaults.colors(
        checkedColor = Color.White,
        uncheckedColor = Color(0xffed145b)
      )
    )
    Text(
      text = "C#",
      color = Color.White
    )
}
```

Código-fonte 31 – Adição de vários Checkbox
Fonte: Elaborado pelo autor (2023)

Execute o aplicativo no emulador. O resultado esperado deverá ser como o da **figura “**Preview da lista de Checkbox”:

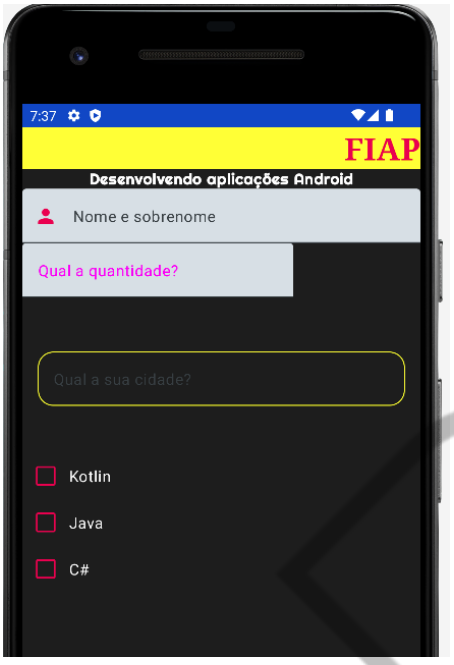

Figura 44 – Preview da lista de Checkbox
Fonte: Elaborado pelo autor (2023)

Se clicarmos em qualquer uma das opções nada acontece. Como faremos o "**Checkbox**" exibir a seleção? Isso mesmo através do "**state**". Dessa vez nossas variáveis de estado receberão como valor inicial um booleano. Seu código deverá ficar como o da listagem abaixo:

```
Spacer(modifier = Modifier.height(32.dp))

var kotlin = remember {
  mutableStateOf(true)
}

var java = remember {
  mutableStateOf(false)
}

var cSharp = remember {
  mutableStateOf(false)
}

Row(
  verticalAlignment = Alignment.CenterVertically,
  modifier = Modifier
    .fillMaxWidth()
) {
  Checkbox(
    checked = kotlin.value,
    onCheckedChange = { kotlin.value = it },
    colors = CheckboxDefaults.colors(
      checkedColor = Color.White,
      uncheckedColor = Color(0xffed145b)
    )
```

```
    )
    Text(
      text = "Kotlin",
      color = Color.White
    )
  }
  Row(
    verticalAlignment = Alignment.CenterVertically,
    modifier = Modifier
      .fillMaxWidth()
  ) {
    Checkbox(
      checked = java.value,
      onCheckedChange = { java.value = it },
      colors = CheckboxDefaults.colors(
        checkedColor = Color.White,
        uncheckedColor = Color(0xffed145b)
      )
    )
    Text(
      text = "Java",
      color = Color.White
    )
  }
  Row(
    verticalAlignment = Alignment.CenterVertically,
    modifier = Modifier
      .fillMaxWidth()
  ) {
    Checkbox(
      checked = cSharp.value,
      onCheckedChange = { cSharp.value = it },
      colors = CheckboxDefaults.colors(
        checkedColor = Color.White,
        uncheckedColor = Color(0xffed145b)
      )
    )
    Text(
      text = "C#",
      color = Color.White
    )
  }
```

Código-fonte 32 – Variáveis de estado para controlar estado das Checkbox
Fonte: Elaborado pelo autor (2023)

Criamos três variáveis que representarão o estado de cada “**Checkbox**”. Para a variável “kotlin” atribuímos o valor “true” e “false” para “java” e “cSharp”. Em seguida atribuímos a cada parâmetro “**checked**” de cada “**Checkbox**” a variável correspondente.

Para o parâmetro “**onCheckedChange**” atribuímos o valor de “**it**”, que neste caso é um valor booleano. Quando clicamos no “Checkbox” se este estiver desmarcado o seu valor mudará para “true” e será passado para a função

“onCheckedChange” através da variável “it”. Quando o valor for “true”, “it” receberá o valor “false” e assim a mágica acontece.

Rode sua aplicação no emulador. Sua IU deverá se parecer como o da figura “Preview do comportamento de seleção das Checkbox”:

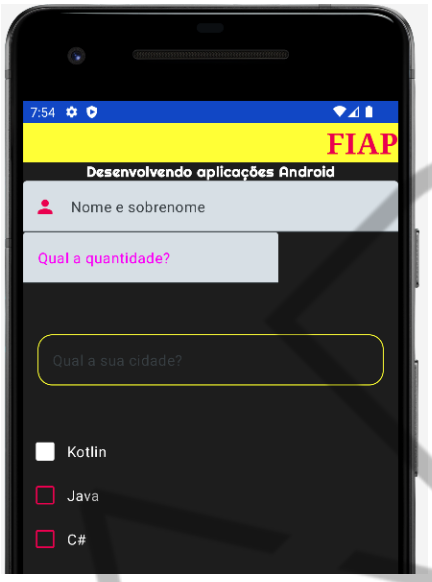

Figura 45 – Preview do comportamento de seleção das Checkbox
Fonte: Elaborado pelo autor (2023)

Perceba que a primeira caixa de seleção é renderizada selecionada. Clique nas outras caixas e verifique se estão tendo o comportamento de seleção desejada.

Lembre-se, com as caixas de seleção damos ao usuário a opção de escolher mais do que uma opção.

3.2 Opções únicas com RadioButton

O “**RadioButton**” é mais um componente fundamental para nossas aplicações. Enquanto o “**Checkbox**” permite a seleção múltipla em uma lista, o “**RadioButton**” é usado quando queremos que apenas uma opção seja selecionada.

Vamos adicionar três “**RadioButton**” logo após nossos “**Checkbox**”. O seu código deverá se parecer com a listagem abaixo:

```
Spacer(modifier = Modifier.height(16.dp))
Row(modifier = Modifier.fillMaxWidth()) {
  Row(verticalAlignment = Alignment.CenterVertically) {
    RadioButton(
      selected = false,
      onClick = { /*TODO*/ },
      colors = RadioButtonDefaults.colors(
        selectedColor = Color.White,
        unselectedColor = Color(0xffed145b)
      )
    )
    Text(text = "MacOS", color = Color.White)
  }
  Row(verticalAlignment = Alignment.CenterVertically) {
    RadioButton(
      selected = false,
      onClick = { /*TODO*/ },
      colors = RadioButtonDefaults.colors(
        selectedColor = Color.White,
        unselectedColor = Color(0xffed145b)
      )
    )
    Text(text = "GNU/Linux", color = Color.White)
  }
  Row(verticalAlignment = Alignment.CenterVertically) {
    RadioButton(
      selected = false,
      onClick = { /*TODO*/ },
      colors = RadioButtonDefaults.colors(
        selectedColor = Color.White,
        unselectedColor = Color(0xffed145b)
      )
    )
    Text(text = "Windows 11", color = Color.White)
  }
}
```

Código-fonte 33 – Inclusão do componente RadioButton
Fonte: Elaborado pelo autor (2023)

Neste exemplo, criamos uma "**Row**", pois queremos que todos os "**RadioButton**" fiquem alinhados lado a lado. E criamos uma "**Row**" para colocarmos o "**RadioButton**" e um texto. Para os "**RadioButton**", também utilizamos o parâmetro "**color**" cujo valor é a função "**RadioButtonDefaults.color()**". Nada muito diferente do que já fizemos com o "**Checkbox**".

Vale lembrar que ao criarmos um "**RadioButton**", obrigatoriamente devemos fornecer o valor para o parâmetro "**selected**", que é um booleano que define se o "**RadioButton**" estará marcado ou não. Além disso, temos que fornecer o parâmetro

“**onClick**”, que executará as instruções para quando o “**RadioButton**” for clicado, marcando ou desmarcando a opção.

Ao executar a aplicação no emulador, sua aplicação deverá se parecer com a **figura “**Preview dos componentes RadioButton”:

Figura 46 – Preview dos componentes RadioButton
Fonte: Elaborado pelo autor (2023)

Novamente, se clicarmos em qualquer um dos “**RadioButton**” não veremos nada acontecer. Mas, você já sabe que precisaremos gerenciar o estado deste componente também.

3.3 Gerenciando o estado do RadioButton

O estado do “RadioButton” determina se ele está marcado ou desmarcado, basicamente. A diferença é que agora podemos ter apenas uma opção selecionada, ou seja, se selecionarmos a opção “**MacOS**” todas as demais devem ser desmarcadas.

Para que as nossas opções de “**RadioButton**” funcionem adequadamente, vamos fazer as alterações de acordo com a listagem abaixo:

```
Spacer(modifier = Modifier.height(16.dp))

var selecionado = remember {
  mutableStateOf(0)
}

Row(modifier = Modifier.fillMaxWidth()) {
  Row(verticalAlignment = Alignment.CenterVertically) {
    RadioButton(
      selected = selecionado.value == 0,
      onClick = { selecionado.value = 0 },
      colors = RadioButtonDefaults.colors(
        selectedColor = Color.White,
        unselectedColor = Color(0xffed145b)
      )
    )
    Text(text = "MacOS", color = Color.White)
  }
  Row(verticalAlignment = Alignment.CenterVertically) {
    RadioButton(
      selected = selecionado.value == 1,
      onClick = { selecionado.value = 1 },
      colors = RadioButtonDefaults.colors(
        selectedColor = Color.White,
        unselectedColor = Color(0xffed145b)
      )
    )
    Text(text = "GNU/Linux", color = Color.White)
  }
  Row(verticalAlignment = Alignment.CenterVertically) {
    RadioButton(
      selected = selecionado.value == 2,
      onClick = { selecionado.value = 2 },
      colors = RadioButtonDefaults.colors(
        selectedColor = Color.White,
        unselectedColor = Color(0xffed145b)
      )
    )
    Text(text = "Windows 11", color = Color.White)
  }
}
```

Código-fonte 34 – Lógica para gerenciamento do estado dos RadioButton
Fonte: Elaborado pelo autor (2023)

Neste exemplo criamos uma variável de estado chamada “**selecionado**”, que recebe inicialmente o valor **zero** e que será atualizado quando um “**RadioButton**” é clicado.

O valor do parâmetro “selected” será o resultado da operação booleana que compara o valor da variável “selecionado” a um número atribuído ao “RadioButton”. O que o parâmetro “onClick” faz é atualizar o valor da variável de estado.

Execute a aplicação no emulador. O resultado esperado deverá ser parecido com o da **figura “**Preview do comportamento do RadioButton”:

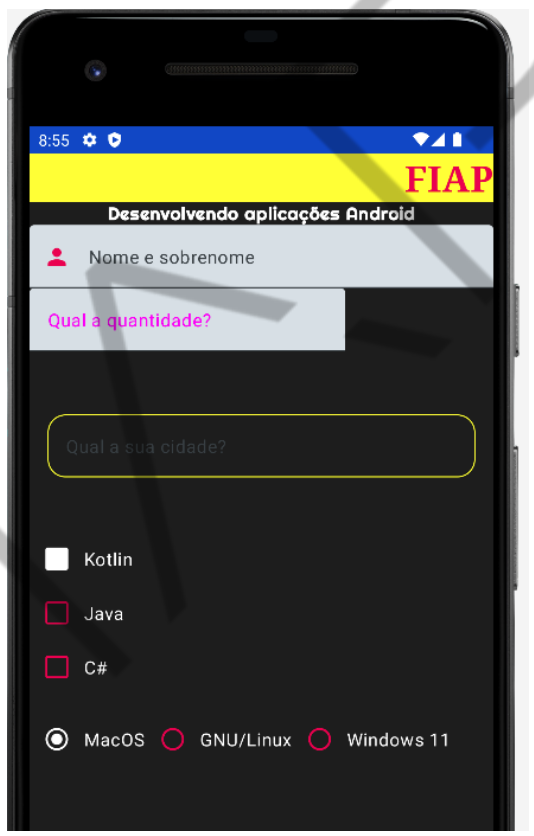

Figura 47 – Preview do comportamento do RadioButton
Fonte: Elaborado pelo autor (2023)

Agora clique nas outras opções e verifique se o comportamento é o esperado.

3.4 Botões

Não teríamos um aplicativo completo se não fosse pelos botões. Saber implementar um botão, é sem dúvida uma tarefa que todo desenvolvedor Android precisa saber. O composable utilizado para criar botões é o “**Button**”.

Vamos acrescentar um botão após a “**Row**” que mantém nossas “**Checkbox**”. Acrescente o seguinte trecho de código:

```
Button(onClick = { /*TODO*/ }) {
  Text(text = "Clique aqui!")
}
```

Código-fonte 35 – Inclusão do componente Button
Fonte: Elaborado pelo autor (2023)

Os botões possuem o parâmetro “**onClick**”, que é responsável por disparar alguma ação quando o botão for clicado pelo usuário. O texto do nosso botão será adicionado utilizando outro componente, o “**Text**”. Ao executar o aplicativo, sua IU deverá se parecer com a figura “Preview do componente Button”:

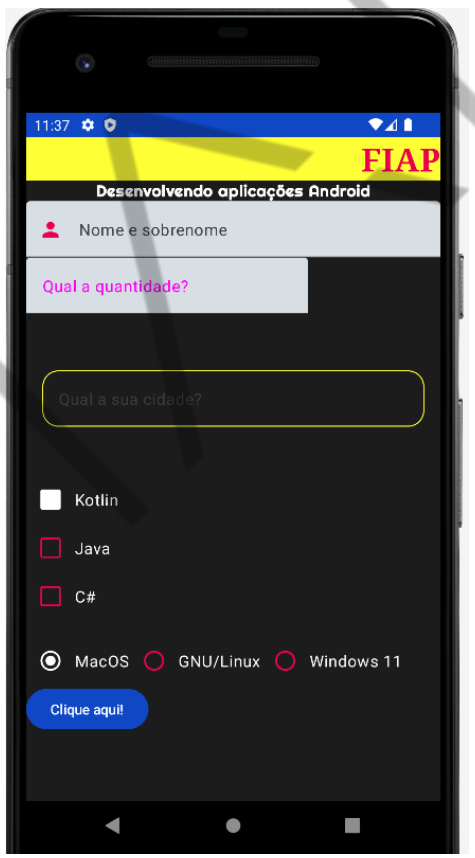

Figura 48 – Preview do componente Button
Fonte: Elaborado pelo autor (2023)

Como todo componente no Jetpack Compose é possível mudar a sua aparência, como cor, tamanho, borda, forma etc. Vamos começar mudando a cor de fundo, o tamanho e a borda. Seu código deverá se parecer com a listagem abaixo:

```
Button(
  onClick = { /*TODO*/ },
  modifier = Modifier.size(width = 200.dp, height = 60.dp),
  colors = ButtonDefaults.buttonColors(containerColor =
Color.Magenta),
  border = BorderStroke(width = 3.dp, color = Color.White)
) {
  Text(text = "Clique aqui!")
}
```

Código-fonte 36 – Estilização do componente Button
Fonte: Elaborado pelo autor (2023)

Através do parâmetro "**Modifier**", alteramos o tamanho para **200.dp** de largura por **60.dp** de altura. Utilizamos o parâmetro "**colors**" através da função "**ButtonDefaults.buttonColors**" para modificar a cor de fundo utilizando o "**containerColor**". O parâmetro "**border"** foi usado para mudarmos a espessura da linha, através do parâmetro "**width**" e a cor da borda através do parâmetro "**color**".

Execute o aplicativo no emulador. O resultado deve se parecer com a **figura "**Preview do componente Button estilizado":

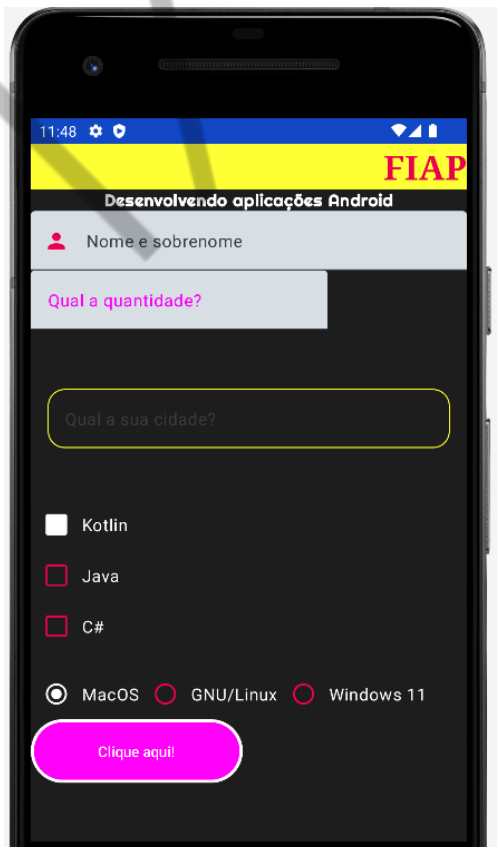

Figura 49 – Preview do componente Button estilizado
Fonte: Elaborado pelo autor (2023)

Além do “**Button**”, que é o botão mais tradicional, também podemos utilizar o “**OutlinedButton**”, que renderizará um botão sem preenchimento. Adicione, após o botão que acabamos de criar o seguinte trecho de código:

```
Button(
  onClick = { /*TODO*/ },
  modifier = Modifier.size(width = 200.dp, height = 60.dp),
  colors = ButtonDefaults.buttonColors(containerColor =
Color.Magenta),
  border = BorderStroke(width = 3.dp, color = Color.White)
) {
  Text(text = "Clique aqui!")
}
OutlinedButton(onClick = { /*TODO*/ }) {
  Text(text = "Outro botão")
}
```

Código-fonte 37 – Inclusão do componente OutlinedButton
Fonte: Elaborado pelo autor (2023)

Execute o aplicativo. Você deverá ver a sua nova IU conforme a **figura** “Preview do OutlinedButton”:

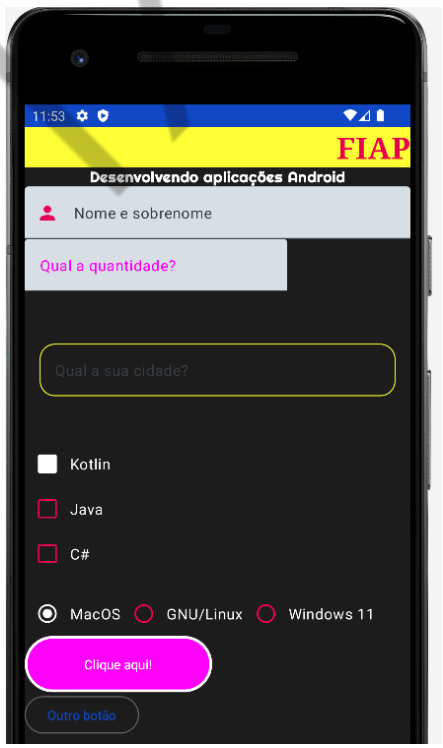

Figura 50 – Preview do OutlinedButton
Fonte: Elaborado pelo autor (2023)

Agora temos um botão sem preenchimento. As modificações do "OutlinedButton", são as mesmas do "Button". Experimente!

3.5 Implementando o clique do botão

Um botão não é um botão sem que execute alguma ação, não é mesmo? A ação do botão será executada através do parâmetro "onClick". A ação a ser executada pode ser escrita dentro das chaves do parâmetro ou podemos escrever uma função separada.

Para testarmos a ação do botão, vamos inserir a frase "**Unidade Paulista**" no "**OutlinedTextField**" assim que o nosso "**Button**" for pressionado.

Para fazer isso, lembre-se que temos uma variável de estado chamada "**cidade**" que é responsável por gerenciar o valor exibido em nossa "**OutlinedTextField**".

Faça a seguinte alteração no parâmetro "onClick" do nosso "Button":

```
Button(
  onClick = {
    cidade.value = "Unidade Paulista"
  },
  modifier = Modifier.size(width = 200.dp, height = 60.dp),
  colors = ButtonDefaults.buttonColors(containerColor =
Color.Magenta),
  border = BorderStroke(width = 3.dp, color = Color.White)
) {
  Text(text = "Clique aqui!")
}
OutlinedButton(onClick = { /*TODO*/ }, colors =
ButtonDefaults.buttonColors()) {
  Text(text = "Outro botão")
}
```

Código-fonte 38 – Implementação do clique do componente Button
Fonte: Elaborado pelo autor (2023)

Execute a aplicação no emulador e clique no botão, você deverá ver o texto "Unidade Paulista" sendo exibido no "**OutlinedTextField**".

Agora, vamos implementar o clique no "**OutlinedButton**" para limpar o "**OutlinedTextField**". Seu código deverá se parecer com a listagem abaixo:

```
Button(
  onClick = {
    cidade.value = "Unidade Paulista"
  },
  modifier = Modifier.size(width = 200.dp, height = 60.dp),
  colors = ButtonDefaults.buttonColors(containerColor =
Color.Magenta),
  border = BorderStroke(width = 3.dp, color = Color.White)
) {
  Text(text = "Clique aqui!")
}
OutlinedButton(onClick = {
  cidade.value = ""
}) {
  Text(text = "Outro botão")
}
```

Código-fonte 39 – Implementação do clique do componente OutlinedButton
Fonte: Elaborado pelo autor (2023)

Execute a aplicação e clique no “**OutlinedButton**”. O conteúdo do “**OutlinedTextField**” deverá ser apagado.

CONCLUSÃO

Neste capítulo aprendemos muito sobre os principais componentes do Jetpack Compose. Claro que ainda há muito mais a aprender, mas o que vimos até aqui nos apresentou conceitos que aplicaremos a praticamente qualquer outro novo componente que nos for apresentado.

O legal do Jetpack Compose é essa padronização nos parâmetros e no que eles irão aplicar aos componentes. Isso torna a programação bastante intuitiva.

A partir deste momento já temos recursos para criarmos aplicações com os principais componentes de uma aplicação Android utilizando Jetpack Compose.

Nos vemos no próximo capítulo!

REFERÊNCIAS

DEVELOPERS. **Texto no compose**. 2023. Disponível em: <https://developer.android.com/jetpack/compose/text?hl=pt-br>. Acesso em: 30 jun. 2023.

DEVELOPERS. **Top-levels functions**. 2023. Disponível em: <https://developer.android.com/reference/kotlin/androidx/compose/material/package-summary#top-level-functions> Acesso em: 30 jun. 2023.