

APP WORLD

USO DE CARDS E **IMAGENS**



7A

LISTA DE FIGURAS

Figura 1 – Layout da Aplicação	7
Figura 2 – Estrutura do projeto	8
Figura 3 - Pasta “ drawable ”	10
Figura 4 – Logotipo da aplicação	11
Figura 5 – Imagens da pasta “drawable”	12
Figura 6 – Visualização da imagem	13
Figura 7 – Resultado do modificador da imagem	14
Figura 8 – Localização do arquivo colors	14
Figura 9 – Mudança da cor de background	16
Figura 10 – Visualizando o Card	18
Figura 11 – Visualização da nova forma do Card	20
Figura 12 – Localização do build.gradle	21
Figura 13 - “Sync Now”	23
Figura 14 - Painel “Build”	23
Figura 15 – Conteúdo do Card	25
Figura 16 – Visualização do Card autoajustável	26
Figura 17 – Visualização do Card resultado	28
Figura 18 – Criar novo arquivo	31
Figura 19 – Inserindo o nome do arquivo	31
Figura 20 – Estrutura do projeto com o arquivo CalculoIMC.kt	32
Figura 21 – Visualização do calculo de IMC	34
Figura 22 – Visualização da formatação de decimal	35
Figura 23 – Aplicação concluída	37

LISTA DE TABELAS

Tabela 1 - Classificação do IMC.....	30
--------------------------------------	----

EMENDAS

LISTA DE CÓDIGOS-FONTE

Código-fonte 1 – Estrutura básica da aplicação	10
Código-fonte 2 – Composable Image	12
Código-fonte 3 – Alteração do tamanho da imagem	13
Código-fonte 4 – Criação da cor vermelho_fiap	15
Código-fonte 5 – Inclusão do título e alteração do background da aplicação	16
Código-fonte 6 – Criação de um Card	17
Código-fonte 7 – Estilização do Card	19
Código-fonte 8 – Alteração do plugin do Kotlin	21
Código-fonte 9 – Alteração da versão de extensões do Compose	22
Código-fonte 10 – Alteração da versão da biblioteca material3	22
Código-fonte 11 – Inclusão do conteúdo do Card	25
Código-fonte 12 – Altura do Card autoajustável	26
Código-fonte 13 – Card resultado	27
Código-fonte 14 – Criação das variáveis de estado	29
Código-fonte 15 – Definição do atributo value	29
Código-fonte 16 – Implementação do atributo onChange	30
Código-fonte 17 – Função CalcularIMC	32
Código-fonte 18 – Implementação do clique do botão calcular	33
Código-fonte 19 – Atribuição da variável de estado ao atributo text	33
Código-fonte 20 – Formatação de casas decimais	34
Código-fonte 21 – Função para classificação de IMC	36
Código-fonte 22 – Uso da função para classificação do IMC	36
Código-fonte 23 – Atualização do status do IMC na IU	37

SUMÁRIO

1 USO DE CARDS E IMAGENS	6
1.1 Apresentação do projeto	6
1.2 Inserindo imagens no aplicativo	10
1.3 Configurando o arquivo colors.xml	14
1.4 Trabalhando com Cards	16
2 MUDANDO A APARÊNCIA DE UM CARD	18
2.1 Ajustando o Gradle do projeto.....	20
2.2 Inserindo o formulário no Card	23
2.3 Inserindo o Card Resultado.....	27
2.4 Gerenciando o estado da aplicação	28
2.5 Funções para calcular o IMC.....	30
3 DESAFIO.....	38
CONCLUSÃO.....	39
REFERÊNCIAS.....	40

1 USO DE CARDS E IMAGENS

As aplicações Android, de modo geral, utilizam imagens para apresentar conteúdo ao usuário. Seja uma aplicação para uma agência de viagens ou de aluguel de veículos, lá estarão elas.

Também é bastante comum utilizarmos os cartões para criarmos uma interface visualmente mais agradável e organizada. O “**Card**” oferece uma maneira fácil de agrupar e representar informações relacionadas no formato de bloco.


Como já aprendemos como implementar vários componentes no Android utilizando Jetpack Compose, a partir deste capítulo vamos começar a trabalhar com pequenos projetos. Vai ficar muito mais fácil e interessante para vermos como tudo se encaixa em nossa aplicação. Então, prepare-se e vamos lá!

1.1 Apresentação do projeto

O projeto que vamos criar é uma simples aplicação para cálculo de Índice de Massa Corpórea, ou simplesmente IMC. Neste projeto vamos utilizar alguns dos componentes que já aprendemos nos capítulos anteriores, além dos componentes “**Image**” e “**Card**”, que são o foco deste capítulo.

Nossa Interface de Usuário (IU) ao final deverá se parecer com a **figura** “Layout da aplicação”:

Calculadora IMC



Calculadora IMC

Seus dados

Seu peso:

Sua altura:

CALCULAR

Resultado
Peso Ideal **21.3**

Figura 1 – Layout da Aplicação
Fonte: Elaborado pelo autor (2023)

A primeira coisa que devemos fazer é “ver” a estrutura do nosso projeto. Já sabemos como organizar o nosso layout, então, após uma análise do layout proposto identificamos os componentes estruturais, de acordo com a figura “Estrutura do projeto”:

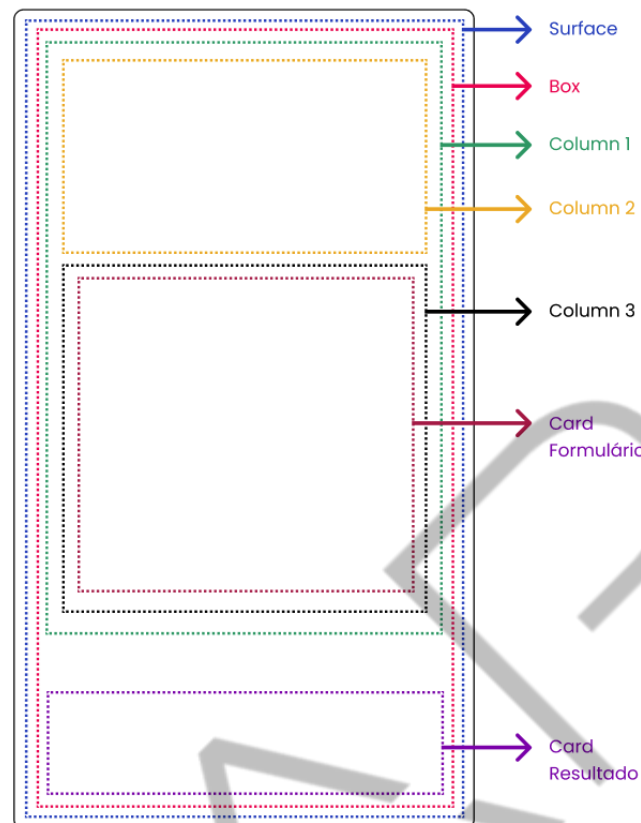


Figura 2 – Estrutura do projeto
Fonte: Elaborado pelo autor (2023)

Lembre-se! Planeje antes de começar. De acordo com o esboço que fizemos, nossa estrutura foi pensada da seguinte forma:

Surface: Envolverá toda a nossa *Activity*.

Box: A “**Box**” ocupará todo o tamanho da *Activity*. Escolhemos a “**Box**”, para facilitar o posicionamento do “**Card**” com o resultado na parte inferior.

Column1: Será um componente “**Column**” que organizará verticalmente a **Column2**, que será o cabeçalho e a **Column3** que será ocupado pelo formulário.

Column2: O cabeçalho da *Activity*.

Column3: Nosso Formulário.

Card Formulário: Aqui temos um componente “**Card**”, que será usado para estilizar nosso formulário.

Card Resultado: Neste “**Card**” exibiremos o resultado do cálculo de IMC.

Explicações dadas, crie um projeto Jetpack Compose no Android Studio com o nome “IMC App”.

Da mesma forma como já fizemos em exercícios anteriores, apague as funções “Greeting” e “GreetingPreview” da classe “MainActivity” e não se esqueça de excluir a chamada para a função “Greeting” no “setContent”. Acrescente o código abaixo para criarmos a estrutura básica da nossa IU. Seu código deverá se parecer com a listagem abaixo:

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

@Composable
fun IMCScreen() {
    Box(
        modifier = Modifier.fillMaxSize()
    ) {
        Column(
            modifier = Modifier
                .fillMaxWidth()
        ) {
            // ---- header -----
            Column(
                horizontalAlignment = Alignment.CenterHorizontally,
                modifier = Modifier
                    .fillMaxWidth()
                    .height(160.dp)
            ) {
            }

            // --- formulário
            Column(
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(horizontal = 32.dp)
            ) {
            }
        }
    }
}
```

```
    }  
  }  
  // -- Card Resultado  
}  
}
```

Código-fonte 1 – Estrutura básica da aplicação
Fonte: Elaborado pelo autor (2023)

O que o código acima implementa é simplesmente a estrutura inicial da nossa IU.

1.2 Inserindo imagens no aplicativo

A utilização de imagens nos aplicativos garante uma IU mais interessante e intuitiva ao usuário. Com as imagens podemos transmitir as informações de forma visual, melhorando a experiência do usuário.

As imagens que utilizamos nos aplicativos Android podem estar disponíveis no pacote da aplicação na forma de recursos, ou podemos até mesmo obtê-las através de um repositório na Internet.

Inicialmente, vamos trabalhar com as imagens como recursos da aplicação, então, devemos armazenar as imagens que utilizaremos na pasta “**drawable**” do nosso projeto. Esta pasta se encontra na pasta “**res**”. A **figura** “Pasta “**drawable**”, nos mostra onde encontrar essas pastas:

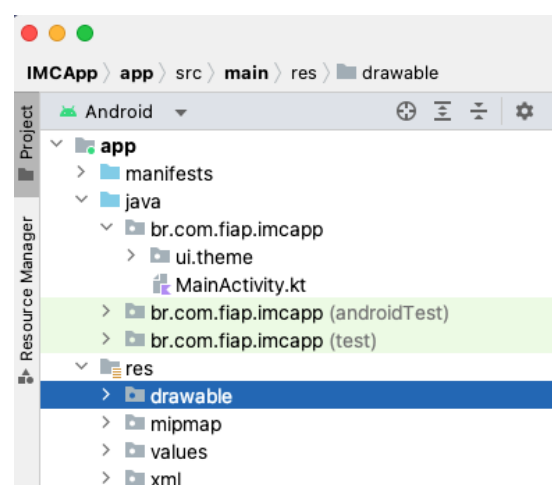


Figura 3 - Pasta “**drawable**”
Fonte: Elaborado pelo autor (2023)

O layout do nosso projeto sugere a imagem de acordo com a **figura** “Logotipo da aplicação”, que pode ser encontrada com facilidade no endereço <https://www.flaticon.com/>, mas sinta-se à vontade para escolher uma imagem de sua preferência.



Figura 4 – Logotipo da aplicação
Fonte: Elaborado pelo autor (2023)

Agora que você já tem a imagem escolhida disponível no seu computador, garanta que o nome da imagem atende os requisitos para uso. Garanta que o nome do arquivo:

- Esteja grafado somente com letras minúsculas
- Não possua caracteres especiais.
- Não possua espaço em branco.
- Não comece por números ou caracteres especiais.

Exemplos de nomes que você **não deve** usar:

- **Bmi.jpg,**
- **bmi-fiap.jpg,**
- **bmi fiap.jpg,**
- **1bmi.jpg.**

Exemplos de nomes que você **deve** usar:

- **bmi.jpg,**
- **bmi_fiap.jpg,**
- **bmi1.jpg**

O Android Studio irá sublinhar de vermelho o nome do arquivo se não atender os requisitos necessários.

Ao colar o arquivo na pasta “drawable”, ela deverá se parecer com a **figura** “Imagens da pasta “drawable”:

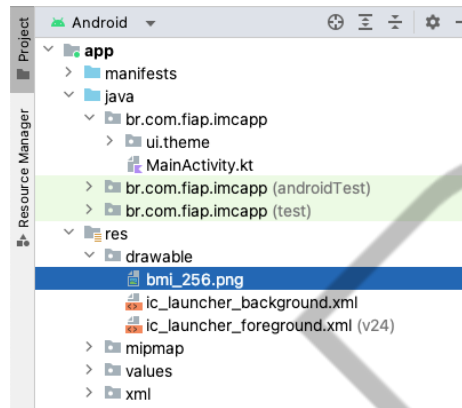


Figura 5 – Imagens da pasta “drawable”
Fonte: Elaborado pelo autor (2023)

A inclusão de imagens em um projeto Jetpack Compose no Android é feita com o composable “Image”, então, vamos adicionar as seguintes instruções conforme a listagem abaixo:

```
// ---- header -----  
Column(  
    horizontalAlignment = Alignment.CenterHorizontally,  
    modifier = Modifier  
        .fillMaxWidth()  
        .height(160.dp)  
) {  
    Image(  
        painter = painterResource(id = R.drawable.bmi_256) ,  
        contentDescription = "logo"  
    )  
}
```

Código-fonte 2 – Composable Image
Fonte: Elaborado pelo autor (2023)

Na função de composição “Image”, temos que fornecer, obrigatoriamente, dois parâmetros, que são:

painter: especifica qual imagem será usada pelo “Image”, através do “painterResource” que indica a imagem local que será usada.

contentDescription: utilizada para descrever a imagem. O uso deste parâmetro torna nossa aplicação mais acessível para pessoas que não podem ver a imagem.

Execute a aplicação no emulador, o resultado esperado deve se parecer com a **figura** “Visualização da imagem”:

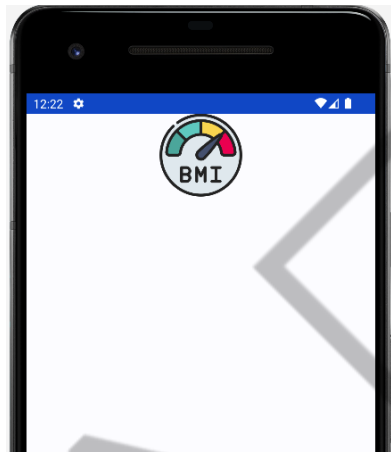


Figura 6 – Visualização da imagem
Fonte: Elaborado pelo autor (2023)

Vamos ajustar a posição e tamanho da imagem, para que não fique colada ao topo da Activity. Após os ajustes, seu código deverá se parecer com a listagem abaixo:

```
// ---- header -----  
Column(  
    horizontalAlignment = Alignment.CenterHorizontally,  
    modifier = Modifier  
        .fillMaxWidth()  
        .height(160.dp)  
) {  
    Image(  
        painter = painterResource(id = R.drawable.bmi_256),  
        contentDescription = "logo",  
        modifier = Modifier  
            .size(60.dp)  
            .padding(top = 16.dp)  
    )  
}
```

Código-fonte 3 – Alteração do tamanho da imagem
Fonte: Elaborado pelo autor (2023)

No código acima, acrescentamos um “Modifier” para alterar o tamanho da imagem através da função “**size(60.dp)**” e acrescentamos um espaço na parte superior da imagem utilizando a função “**padding(top = 16.dp)**”. Ao executar a aplicação, sua IU deve se parecer com a **figura** “Resultado do modificador da imagem”:

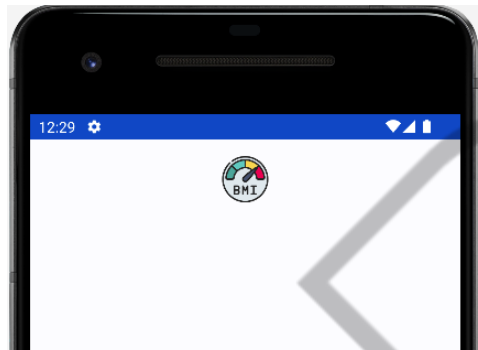


Figura 7 – Resultado do modificador da imagem
Fonte: Elaborado pelo autor (2023)

1.3 Configurando o arquivo colors.xml

Em nosso layout, faremos bastante uso da cor vermelha, representada pelo código hexadecimal ED145B. É tedioso ficar digitando esse código sempre que queremos utilizar essa cor, então, vamos cadastrar essa cor no arquivo “**colors**” do nosso projeto. A **figura** “Localização do arquivo colors” mostra a localização do arquivo de cores:

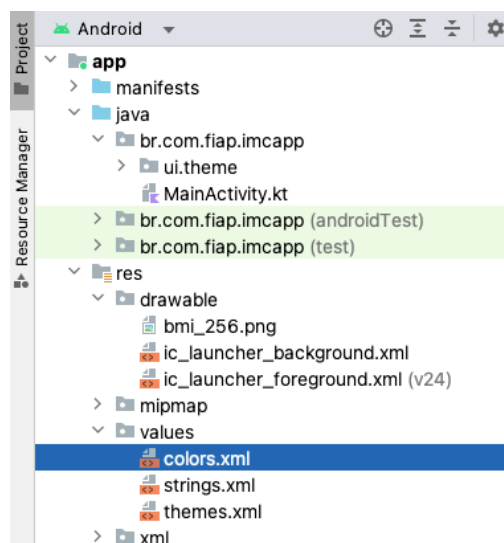


Figura 8 – Localização do arquivo colors
Fonte: Elaborado pelo autor (2023)

Este arquivo é um recurso utilizado para definir e gerenciar todas as cores utilizadas no projeto. Através dele podemos criar toda a identidade de cores do nosso aplicativo, o que torna muito fácil e rápido a troca das cores em caso de mudança.

No arquivo “**colors.xml**” definimos as cores utilizando o padrão XML, onde temos uma “**tag**” que identifica a cor e o valor desta “**tag**” que é o código hexadecimal da cor. Abra o arquivo “colors.xml” e acrescente a cor “**vermelho_fiap**”, de acordo com a listagem abaixo:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
    <color name="vermelho_fiap">#FFED145B</color>
</resources>
```

Código-fonte 4 – Criação da cor vermelho_fiap
Fonte: Elaborado pelo autor (2023)

A partir de agora, sempre que precisarmos da nossa cor vermelha utilizaremos o recurso de cor que acabamos de inserir em “**color.xml**”.

Vamos trocar a cor de background da “**Column**” que contém nossa imagem para “**vermelho_fiap**” e acrescentar o título do nosso aplicativo. Após o ajuste o seu código deverá se parecer com a listagem abaixo:

```
// ---- header -----
Column(
    horizontalAlignment = Alignment.CenterHorizontally,
    modifier = Modifier
        .fillMaxWidth()
        .height(160.dp)
        .background(colorResource(id = R.color.vermelho_fiap))
) {
    Image(
        painter = painterResource(id = R.drawable.bmi_256),
        contentDescription = "logo",
        modifier = Modifier
            .size(60.dp)
            .padding(top = 16.dp)
```

```
)  
Text(  
    text = "Calculadora IMC",  
    fontSize = 24.sp,  
    color = Color.White,  
    fontWeight = FontWeight.Bold,  
    modifier = Modifier.padding(top = 12.dp, bottom = 24.dp)  
)  
}
```

Código-fonte 5 – Inclusão do título e alteração do background da aplicação
Fonte: Elaborado pelo autor (2023)

Após a alteração, a sua IU deve se parecer com a **figura** “Background da “Column”:

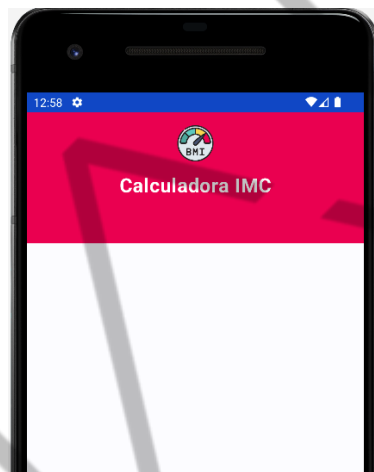


Figura 9 – Mudança da cor de background
Fonte: Elaborado pelo autor (2023)

1.4 Trabalhando com Cards

Para a implementação do formulário que será usado para inserir os dados de peso e altura do usuário utilizaremos o composável “**Card**”. O “**Card**” permite a criação de uma superfície elevada que conterà o conteúdo. Frequentemente utilizamos o “**Card**” para agrupar conteúdo que estão relacionados e fornecer uma aparência mais organizada.

Vamos criar um “**Card**” que organizará todos os componentes do formulário. Insira as instruções de acordo com a listagem abaixo:


```
// --- formulário
Column(
  modifier = Modifier
    .fillMaxWidth()
    .padding(horizontal = 32.dp)
) {
  Card(
    modifier = Modifier
      .offset(y = (-30).dp)
      .fillMaxWidth()
      .height(300.dp) ,
    colors = CardDefaults
      .cardColors(containerColor = Color(0xffff9f6f6)),
    elevation = CardDefaults.cardElevation(4.dp)
  ) {

  }
}
```

Código-fonte 6 – Criação de um Card
Fonte: Elaborado pelo autor (2023)

O “**Modifier**” do “**Card**” está utilizando as seguintes funções:

- **offset(y = (-30).dp)**: essa função está deslocando o card no eixo y. O valor negativo, deve ser colocado entre parênteses e movimenta o “**Card**” para cima. Isso faz com que o “**Card**” fique sobreposto à “**Column**” do cabeçalho.
- **fillMaxWidth()**: determina que o “**Card**” ocupe a largura total do componente pai, que é um “**Column**”.
- **height(300.dp)**: determina a altura do “**Card**” em **300.dp**. Isso será temporário, fizemos isso agora para que possamos visualizar o “**Card**” mesmo que ele ainda não tenha conteúdo algum.

Além do modificador, ainda estamos utilizando os seguintes parâmetros:

- **colors**: o parâmetro “**colors**” pode definir a cor do “**Card**” ou do seu conteúdo. Nesta configuração, a função “**cardColors**” está modificando o parâmetro “**containerColor**”, que muda a cor de *background* do “**Card**”;
- **elevation**: este parâmetro inseri uma sombra sob o “**Card**”, causando o efeito de elevação.

Ao executar o aplicativo, a IU deve se parecer com a **figura** “Visualizando o Card”:



Figura 10 – Visualizando o Card
Fonte: Elaborado pelo autor (2023)

2 MUDANDO A APARÊNCIA DE UM CARD

É possível ainda alterarmos a forma do “**Card**” utilizando o parâmetro “**shape**”, assim como é possível também colocarmos uma borda utilizando o parâmetro “**border**”. Vamos experimentar esses dois parâmetros. O seu código deverá se parecer com a listagem abaixo:

```
// --- formulário
Column(
    modifier = Modifier
        .fillMaxWidth()
        .padding(horizontal = 32.dp)
) {
    Card(
        modifier = Modifier
            .offset(y = (-30).dp)
            //.fillMaxWidth()
            //.height(300.dp),
            .size(250.dp),
        colors = CardDefaults
            .cardColors(containerColor = Color(0xffff9f6f6)),
```

```
elevation = CardDefaults.cardElevation(4.dp),  
shape = CircleShape,  
border = BorderStroke(width = 4.dp, color = Color.Black)  
) {  
  
}  
}
```

Código-fonte 7 – Estilização do Card

Fonte: Elaborado pelo autor (2023)

As modificações em nosso código foram as seguintes:

- Comentamos os modificadores “**fillMaxWidth()**” e “**height(300.dp)**”, para definirmos um tamanho fixo de **250.dp** através da função modificadora “**size(25.dp)**”. Isso foi necessário para o uso do “**shape**”;
- Adicionamos o parâmetro “**shape**” com o valor “**CircleShape**”, para que o “**Card**” ficasse circular. A circunferência perfeita necessita que a forma original seja um quadrado e esse foi o motivo de usarmos o modificador “**size**”.
- Colocamos uma borda através do parâmetro “**border**”. Esse parâmetro recebe como valor a função “**BorderStroke**” que por sua vez precisa da largura e cor da borda, utilizando os parâmetros “**width**” e “**color**”.

Ao executar a aplicação, nosso “Card” deverá se parecer com a figura “Visualização da nova forma do Card”:

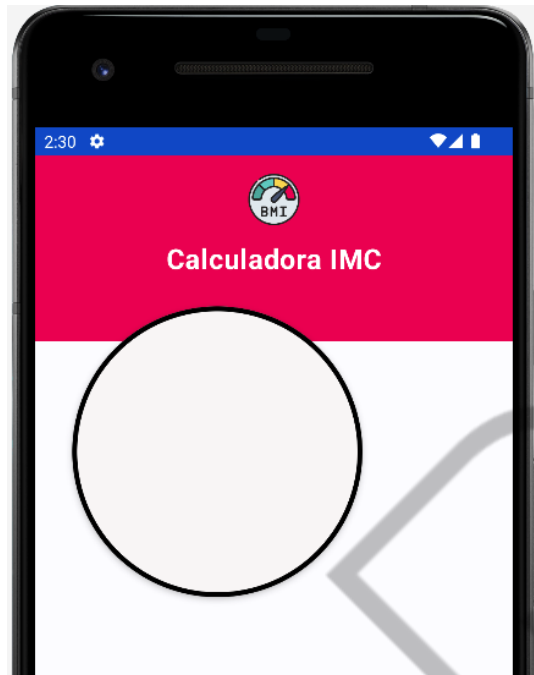


Figura 11 – Visualização da nova forma do Card
Fonte: Elaborado pelo autor (2023)

Essas modificações foram apenas para experimentarmos outros parâmetros visuais que podemos modificar em um “**Card**”. Não será esse o visual final que desejamos, então, remova essas últimas alterações de modo que nosso “**Card**” fique com a aparência da **figura 10**.

2.1 Ajustando o Gradle do projeto

O **Gradle** é uma ferramenta que automatiza todo o processo de construção, também conhecido como “**build**”, do nosso projeto. O **Gradle** é responsável pela criação de toda a estrutura de pastas necessárias ao projeto, além do download e gerenciamento das dependências do projeto, facilitando a inclusão das bibliotecas e módulos externos que utilizamos para criar nossas aplicações.

Neste projeto estamos utilizando o “**Material Design 3**”, que é uma biblioteca de componentes e diretrizes lançadas pelo Google para o Jetpack Compose que torna a IU mais moderna e intuitiva ao usuário.

Para que possamos utilizar o “**Material Design 3**” da forma mais adequada precisamos atualizar a versão do plugin de suporte ao Kotlin e do compilador de

extensões do Compose, então, localize os arquivos “**build.gradle (Project: IMC_App)**” e “**build.gradle(Module :app)**”, de acordo com a figura “Localização do build.gradle”:

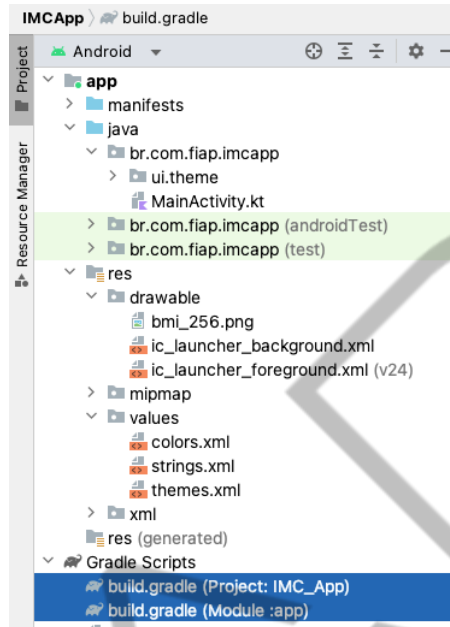


Figura 12 – Localização do build.gradle
Fonte: Elaborado pelo autor (2023)

Abra o arquivo “**build.gradle (Project: IMC_App)**”, e altere a versão do plugin do Kotlin para “1.8.0”. Seu código deverá se parecer com a listagem abaixo:

```
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 8 – Alteração do plugin do Kotlin
Fonte: Elaborado pelo autor (2023)

Note que a configuração anterior foi mantida e comentada para critério de documentação.

No arquivo “**build.gradle (Module :app)**”, localize o bloco “ComposeOptions” e o deixe como a listagem abaixo:

```
composeOptions {  
    // kotlinCompilerExtensionVersion '1.3.2'
```

```
}  
    kotlinCompilerExtensionVersion '1.4.0'  
}
```

Código-fonte 9 – Alteração da versão de extensões do Compose
Fonte: Elaborado pelo autor (2023)

Ainda neste arquivo, localize o bloco de código “dependencies” e altere a anotação de implementação do “material3”, seu código deverá se parecer com o da listagem abaixo:

```
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 10 – Alteração da versão da biblioteca material3
Fonte: Elaborado pelo autor (2023)

Após ajustes no Gradle, vamos sincronizar as alterações para que sejam aplicadas corretamente no processo de compilação do projeto. Na parte superior do editor de código do Android Studio, sempre que alguma coisa for alterada no Gradle, teremos uma barra de aviso solicitando que a sincronização seja feita, conforme a **figura** “Sync Now”. Clique em “Sync Now” e aguarde o término da aplicação das novas configurações.

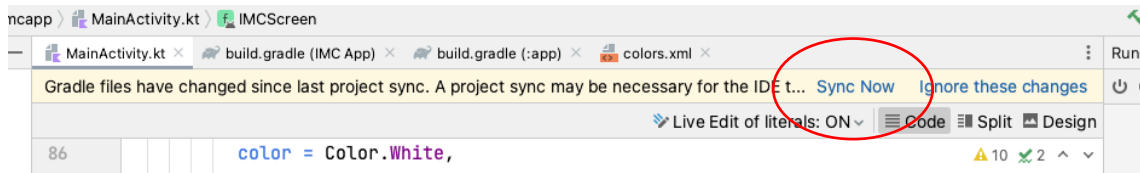


Figura 13 - “Sync Now”
Fonte: Elaborado pelo autor (2023)

Se tudo estiver OK, você deverá ver a mensagem “BUILD SUCCESSFUL” no painel “Build”, na parte inferior da sua tela, conforme a **figura** “Painel “Build””:

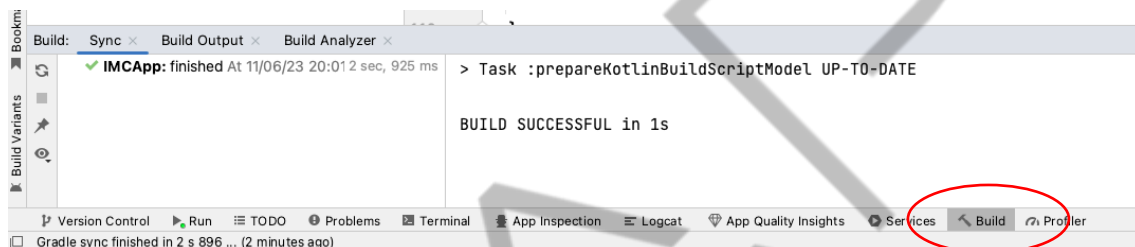


Figura 14 - Painel “Build”
Fonte: Elaborado pelo autor (2023)

2.2 Inserindo o formulário no Card

Para a construção do formulário da aplicação vamos utilizar os composables que já conhecemos, como “Text”, “OutlinedTextField” e “Button”. O seu código deverá se parecer com a listagem abaixo:

```
// --- formulário
Column(
    modifier = Modifier
        .fillMaxWidth()
        .padding(horizontal = 32.dp)
) {
    Card(
        modifier = Modifier
            .offset(y = (-30).dp)
            .fillMaxWidth()
            .height(300.dp),
        colors = CardDefaults
```

```

        .cardColors(containerColor = Color(0xffff9f6f6)),
        elevation = CardDefaults.cardElevation(4.dp)
    ) {
        Column(
            modifier = Modifier.padding(
                vertical = 16.dp,
                horizontal = 32.dp
            )
        ) {
            Text(
                text = "Seus dados",
                modifier = Modifier.fillMaxWidth(),
                fontSize = 24.sp,
                fontWeight = FontWeight.Bold,
                color = colorResource(id = R.color.vermelho_fiap),
                textAlign = TextAlign.Center
            )
            Spacer(modifier = Modifier.height(32.dp))
            Text(
                text = "Seu peso",
                modifier = Modifier.padding(bottom = 8.dp),
                fontSize = 12.sp,
                fontWeight = FontWeight.Normal,
                color = colorResource(id = R.color.vermelho_fiap)
            )
            OutlinedTextField(
                value = "",
                onChange = {},
                modifier = Modifier.fillMaxWidth(),
                placeholder = {
                    Text(text = "Seu peso em Kg.")
                },
                colors = OutlinedTextFieldDefaults.colors(
                    unfocusedBorderColor = colorResource(id = R.color.vermelho_fiap),
                    focusedBorderColor = colorResource(id = R.color.vermelho_fiap)
                ),
                shape = RoundedCornerShape(16.dp),
                keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Number)
            )
            Spacer(modifier = Modifier.height(16.dp))
            Text(
                text = "Sua altura",
                modifier = Modifier.padding(bottom = 8.dp),
                fontSize = 12.sp,
                fontWeight = FontWeight.Normal,
                color = colorResource(id = R.color.vermelho_fiap)
            )
            OutlinedTextField(
                value = "",
                onChange = {},
                modifier = Modifier.fillMaxWidth(),
                placeholder = {
                    Text(
                        text = "Sua altura em cm."
                    )
                },
                colors = OutlinedTextFieldDefaults.colors(
                    unfocusedBorderColor = colorResource(id = R.color.vermelho_fiap),
                    focusedBorderColor = colorResource(id = R.color.vermelho_fiap)
                ),
            ),
        )
    }
}

```



```

        shape = RoundedCornerShape(16.dp) ,
        keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Decimal)
    )
    Spacer(modifier = Modifier.height(16.dp))
    Button(
        onClick = {},
        modifier = Modifier
            .fillMaxWidth()
            .height(48.dp) ,
        shape = RoundedCornerShape(16.dp) ,
        colors = ButtonDefaults.buttonColors(containerColor =
colorResource(id = R.color.vermelho_fiap))
    ) {
        Text(
            text = "CALCULAR",
            fontWeight = FontWeight.Bold,
            color = Color.White,
            fontSize = 14.sp
        )
    }
}
}
}
}

```

Código-fonte 11 – Inclusão do conteúdo do Card
 Fonte: Elaborado pelo autor (2023)

Execute a aplicação no emulador. O resultado da IU deve se parecer com a figura “Conteúdo do Card”:



Figura 15 – Conteúdo do Card
 Fonte: Elaborado pelo autor (2023)

O formulário é maior do que o “**Card**”. Isso acontece porque definimos a altura do “**Card**” em **300.dp**. Vamos fazer com que o “**Card**” se ajuste ao conteúdo removendo a função “**height**” do modificador do “**Card**”. Seu código deverá se parecer com a listagem abaixo:

```
// --- formulário
Column(
  modifier = Modifier
    .fillMaxWidth()
    .padding(horizontal = 32.dp)
) {
  Card(
    modifier = Modifier
      .offset(y = (-30).dp)
      .fillMaxWidth(),
    // .height(300.dp), ← Remova esta linha
    colors = CardDefaults
      .cardColors(containerColor = Color(0xffff9f6f6f)),
    elevation = CardDefaults.cardElevation(4.dp)
  )
}
```

Código-fonte 12 – Altura do Card autoajustável
Fonte: Elaborado pelo autor (2023)

Execute novamente a aplicação. Agora o resultado deve se parecer com a **figura** “Visualização do Card autoajustável”.



Figura 16 – Visualização do Card autoajustável
Fonte: Elaborado pelo autor (2023)

Como não definimos uma altura fixa para o “**Card**”, ele se expandirá para se ajustar ao seu conteúdo.

2.3 Inserindo o Card Resultado

Em relação ao layout da aplicação, só falta incluir o “**Card**” que exibirá o resultado do IMC. O seu código deverá se parecer com a listagem abaixo:

```
// --- card resultado
Card(
    modifier = Modifier
        .fillMaxWidth()
        .height(200.dp)
        .padding(horizontal = 32.dp, vertical = 24.dp)
        .align(Alignment.BottomCenter),
    colors = CardDefaults.cardColors(containerColor = Color(0xff329F6B)),
    elevation = CardDefaults.cardElevation(4.dp),
    //border = BorderStroke(width = 1.dp, Color(0xffed145b))
) {
    Row(
        verticalAlignment = Alignment.CenterVertically,
        modifier = Modifier
            .padding(horizontal = 32.dp)
            .fillMaxSize()
    ) {
        Column() {
            Text(
                text = "Resultado",
                color = Color.White,
                fontSize = 14.sp
            )
            Text(
                text = "Peso Ideal.",
                fontWeight = FontWeight.Bold,
                color = Color.White,
                fontSize = 20.sp
            )
        }
        Text(
            text = "23.2",
            modifier = Modifier.fillMaxWidth(),
            fontWeight = FontWeight.Bold,
            color = Color.White,
            fontSize = 36.sp,
            textAlign = TextAlign.End
        )
    }
}
```

Código-fonte 13 – Card resultado
Fonte: Elaborado pelo autor (2023)

Execute a aplicação novamente. A IU da aplicação deverá se parecer com a **figura** “Visualização do Card resultado”:



Figura 17 – Visualização do Card resultado
Fonte: Elaborado pelo autor (2023)

2.4 Gerenciando o estado da aplicação

Nosso layout já está pronto, mas não podemos interagir com a aplicação. Ao clicar em qualquer caixa de texto, o teclado virtual será exibido, mas ao digitar nada acontece. Para resolver este problema vamos criar as variáveis responsáveis por manter o estado dos componentes da tela. Precisamos de quatro variáveis. São elas:

- **Peso:** guardará o peso informado pelo usuário em quilos.
- **Altura:** guardará a altura do usuário em centímetros.
- **Imc:** armazenará o valor do IMC calculado para o usuário.
- **Statusimc:** armazenará o status do IMC calculado para o usuário.

No início da função “**IMCScreen**”, acrescente as instruções de acordo com a listagem abaixo:

```
@Composable
fun IMCScreen() {

    var peso = remember {
        mutableStateOf("")
    }

    var altura = remember {
        mutableStateOf("")
    }

    var imc = remember {
        mutableStateOf(0.0)
    }

    var statusImc = remember {
        mutableStateOf("")
    }

    Box(
        modifier = Modifier.fillMaxSize()
    )
    // restante do código foi omitido.
```

Código-fonte 14 – Criação das variáveis de estado
Fonte: Elaborado pelo autor (2023)

No código acima, estamos criando variáveis de estado, que se “lembrarão” do valor que elas armazenam durante o processo de recomposição de um *composable*. Isso é necessário para que o usuário veja o estado atual da IU.

Essas variáveis deverão ser atribuídas ao parâmetro “**value**” dos seus composables. Também é necessário implementarmos a função “**onValueChange**” dos componentes “**OutlinedTextField**” para que as variáveis sejam atualizadas durante a digitação do usuário. Seu código deverá se parecer com as listagens abaixo:

```
OutlinedTextField(
    value = peso.value,
    onValueChange = { peso.value = it },
    modifier = Modifier.fillMaxWidth(),
    placeholder = {
        Text(text = "Seu peso em Kg.")
    },
)
```

Código-fonte 15 – Definição do atributo value
Fonte: Elaborado pelo autor (2023)

```
OutlinedTextField(
    value = altura.value,
    onValueChange = { altura.value = it },
    modifier = Modifier.fillMaxWidth(),
    placeholder = {
        Text(
            text = "Sua altura em cm."
        )
    },
)
```

Código-fonte 16 – Implementação do atributo onValueChange
 Fonte: Elaborado pelo autor (2023)

Com os ajustes implementados nos componentes “**OutlinedTextField**”, já será possível digitar os valores. Rode a aplicação e verifique isso.

2.5 Funções para calcular o IMC

O aplicativo deverá calcular o IMC e a sua classificação baseado na tabela abaixo:

IMC	CLASSIFICAÇÃO
Abaixo de 18,5	Abaixo do peso.
> 18, 5 e < 25	Peso ideal.
>= 25 e < 30	Levemente acima do peso.
>= 30 e < 35	Obesidade grau I.
>= 35 e < 40	Obesidade grau II.
> 40	Obesidade grau III.

Tabela 1 - Classificação do IMC
 Fonte: Elaborado pelo autor (2023)

Vamos criar um novo arquivo na pasta “**br.com.fiap.imcapp**”, com o nome “**CalculoIMC**”, em que teremos duas funções: “**calcularIMC()**” e “**determinarClassificacaoIMC**”. Para isso siga os passos abaixo:

Clique com o botão direito do mouse na pasta “**br.com.fiap.imcapp**”, selecione a opção “**New**” e “**Kotlin Class/File**”, de acordo com a **figura** “Criar novo arquivo”:

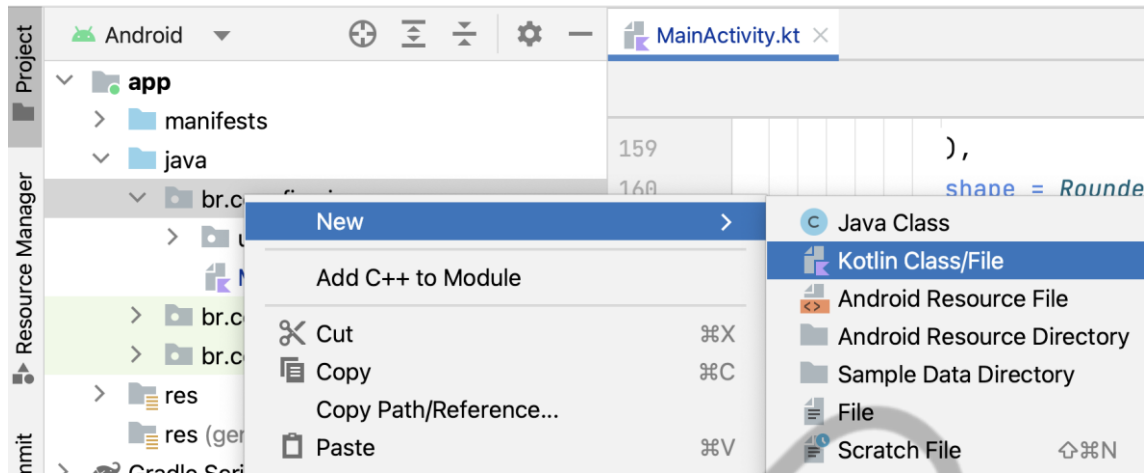


Figura 18 – Criar novo arquivo
Fonte: Elaborado pelo autor (2023)

Na caixa de diálogo “**New Kotlin Class/File**” marque a opção “**File**” e digite o nome no arquivo “**CalculoIMC**” e pressione “**Enter**”, conforme a **figura** “Inserindo o nome do arquivo”:

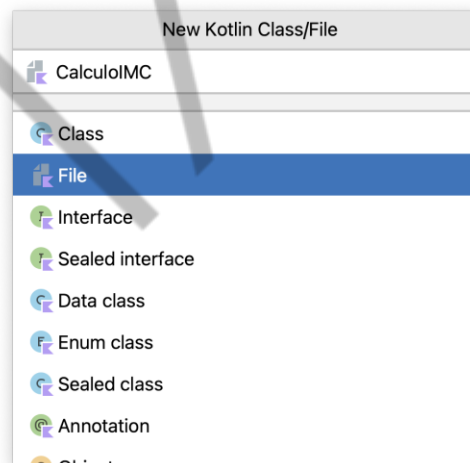


Figura 19 – Inserindo o nome do arquivo
Fonte: Elaborado pelo autor (2023)

Seu painel de projeto deverá se parecer com a **figura** “Estrutura do projeto com o arquivo CalculoIMC.kt”:

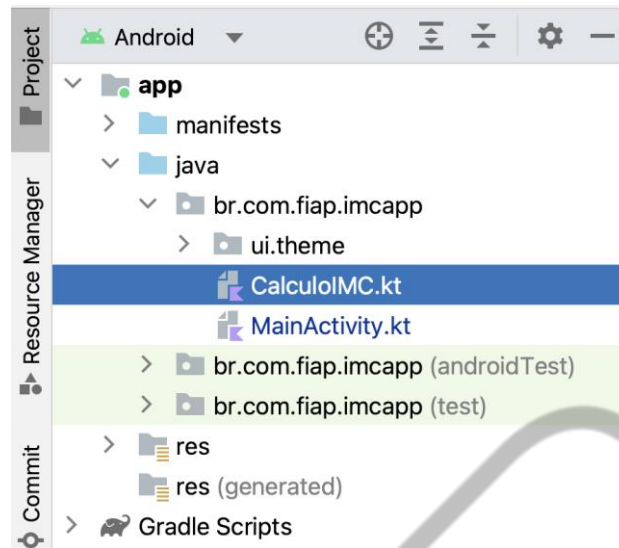


Figura 20 – Estrutura do projeto com o arquivo CalculoIMC.kt
Fonte: Elaborado pelo autor (2023)

Escreva as seguintes instruções no arquivo “CalculoIMC”:

```
package br.com.fiap.imcapp

import kotlin.math.pow

fun calcularIMC(altura: Double, peso: Double) : Double {
    return peso / (altura / 100).pow(2.0)
}
```

Código-fonte 17 – Função CalcularIMC
Fonte: Elaborado pelo autor (2023)

No trecho de código acima criamos uma função utilizando a palavra reservada “**fun**”, com o nome “**calcularIMC**”. Essa função recebe dois argumentos chamados “**peso**” “**altura**” do tipo “**Double**”. Essa função retornará o resultado da divisão do “**peso**” pelo quadrado da “**altura**”.

Vamos chamar essa função quando o usuário clicar no botão. Adicione as seguintes instruções na função “**IMCScreen**”, de acordo com a listagem de código abaixo:


```
Button(  
    onClick = {  
        imc.value = calcularIMC(  
            altura = altura.value.toDouble(),  
            peso = peso.value.toDouble()  
        )  
    },  
),
```

Código-fonte 18 – Implementação do clique do botão calcular
Fonte: Elaborado pelo autor (2023)

Quando ocorrer o clique no botão haverá a chamada para a função “**calcularIMC()**” passando-se os dois argumentos “**peso**” e “**altura**”. Essas duas variáveis são do tipo “**String**”, então, utilizamos a função “**toDouble()**” para fazer o casting de “**String**” para “**Double**”.

Já temos o valor do IMC armazenado na variável de estado “**imc**”. Vamos atribuir essa variável ao parâmetro “**value**” do composável “**Text**” para que o usuário veja o resultado. Localize no código o “**Text**” responsável por exibir o valor do IMC e efetue a modificação de acordo com a listagem abaixo:

```
Text(  
    text = imc.value.toString(),  
    modifier = Modifier.fillMaxWidth(),  
    fontWeight = FontWeight.Bold,  
    color = Color.White,  
    fontSize = 36.sp,  
    textAlign = TextAlign.End  
)
```

Código-fonte 19 – Atribuição da variável de estado ao atributo text
Fonte: Elaborado pelo autor (2023)

Execute a aplicação, preencha os campos “**peso**” e “**altura**”, clique no botão “**Calcular**”. O resultado deverá ser parecido com a **figura** “Visualização do calculo de IMC”:



Figura 21 – Visualização do calculo de IMC
Fonte: Elaborado pelo autor (2023)

Como podemos observar o resultado do IMC foi calculado corretamente, mas estamos exibindo muitas casas decimais, que prejudica a questão visual do nosso aplicativo. Vamos ajustar a exibição para apenas 1 casa decimal. Efetue o ajuste de acordo com a listagem abaixo:

```
Text(  
    text = String.format("%.1f", imc.value),  
    modifier = Modifier.fillMaxWidth(),  
    fontWeight = FontWeight.Bold,  
    color = Color.White,  
    fontSize = 36.sp,  
    textAlign = TextAlign.End  
)
```

Código-fonte 20 – Formatação de casas decimais
Fonte: Elaborado pelo autor (2023)

Utilizamos a função “**format**” da classe “**String**” para determinar como o valor deveria ser mostrado. O método “**format**” recebe dois parâmetros: o primeiro define o formato da exibição, que no caso foi “%.1f”. O símbolo de porcentagem (%) indica qualquer valor numérico antes do ponto decimal. O “1f”, indica que o resultado será apresentado com uma casa decimal. Se você quiser mostrar duas casas decimais, utilize o formato “%.2f”, e assim por diante.

Execute a aplicação e preencha novamente os campos. Ao clicar no botão o resultado deverá se apresentar como na **figura** “Visualização da formatação de decimal”:

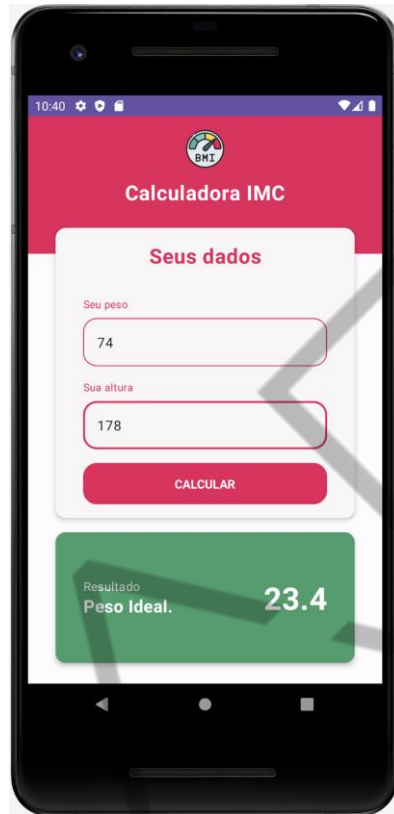


Figura 22 – Visualização da formatação de decimal
Fonte: Elaborado pelo autor (2023)

Vamos implementar a função que determina a classificação o IMC do usuário. A classificação deve atender à tabela 1. Abra o arquivo “CalculoIMC” e acrescente a função “**determinarClassificacaoIMC()**”, de acordo com a listagem abaixo:

```

package br.com.fiap.imcapp

import kotlin.math.pow

fun calcularIMC(altura: Double, peso: Double) : Double {
    return peso / (altura / 100).pow(2.0)
}

fun determinarClassificacaoIMC(imc: Double): String {
    return if (imc < 18.5) {
        "Abaixo do peso"
    } else if (imc >= 18.5 && imc < 25.0) {
        "Peso Ideal"
    } else if (imc >= 25.0 && imc < 30.0) {
        "Levemente acima do peso"
    } else if (imc >= 30.0 && imc < 35.0) {
        "Obesidade Grau I"
    } else if (imc >= 35.0 && imc < 40.0) {
        "Obesidade Grau II"
    } else {"Obesidade Grau III"}
}

```

Código-fonte 21 – Função para classificação de IMC
 Fonte: Elaborado pelo autor (2023)

A função “determinarClassificacaoIMC()” simplesmente recebe o valor do IMC do usuário e retorna uma expressão “String” de acordo com as condicionais.

O retorno dessa função deve ser atribuído à variável de estado “**statusImc**”, então, inclua a seguinte instrução ao parâmetro “**onClick**” do botão:

```

Button(
    onClick = {
        imc.value = calcularIMC(
            altura = altura.value.toDouble(),
            peso = peso.value.toDouble()
        )
        statusImc.value = determinarClassificacaoIMC(imc.value)
    },
    modifier = Modifier
        .fillMaxWidth()
        .height(48.dp),
    shape = RoundedCornerShape(16.dp),
    colors = ButtonDefaults.buttonColors(containerColor = colorResource(id =
        R.color.vermelho_fiap))
)

```

Código-fonte 22 – Uso da função para classificação do IMC
 Fonte: Elaborado pelo autor (2023)

Em seguida, a variável “**statusImc**” deve ser atribuída ao parâmetro “**value**” do composável “**Text**” responsável por exibir a classificação de IMC do usuário, então,

localize este “Text” e inclua as modificações necessárias de acordo com a listagem abaixo:

```
Text(  
    text = statusImc.value,  
    fontWeight = FontWeight.Bold,  
    color = Color.White,  
    fontSize = 20.sp  
)
```

Código-fonte 23 – Atualização do status do IMC na IU

Fonte: Elaborado pelo autor (2023)

Execute a aplicação novamente e preencha os campos com os dados do usuário. O resultado esperado deve se parecer com a **figura** “Aplicação concluída”:



Figura 23 – Aplicação concluída
Fonte: Elaborado pelo autor (2023)

3 DESAFIO

A melhor forma de obter uma nova habilidade é praticando, então, reúna os conhecimentos que você obteve até aqui e adicione ao layout um novo botão para limpar a IU, este botão deve limpar todas as informações da última interação do usuário para iniciar uma nova interação. O layout fica por sua conta, seja criativo!

Outra melhoria que vai deixar a sua aplicação muito mais interessante é alterar a cor do **“Card”** de acordo com a classificação de IMC do usuário. As cores que você pode implementar são as seguintes:

- **Card Verde:** para a classificação “Peso Ideal”.
- **Card Vermelho:** para as classificações “Abaixo do Peso”, “Obesidade Grau I”, “Obesidade Grau II” e “Obesidade Grau III”.
- **Card Amarelo ou Laranja:** para a classificação “Levemente Acima do Peso”.

Você irá aprender muito implementando essas melhorias. Não perca esta oportunidade de praticar!

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. **Layouts no Compose.** 2023. Disponível em: <<https://developer.android.com/jetpack/compose/layout?hl=pt-br>>. Acesso em: 06 jul. 2023.

DEVELOPERS. **Como trabalhar com imagens.** 2023. Disponível em: <<https://developer.android.com/jetpack/compose/graphics/images?hl=pt-b>>. Acesso em: 06 jul. 2023.

EMANDA