**Aula XIV e XV - DevAppII**

**Componentes de Layout:**

**Guias (TabController e DefaulTabController)**

**Drawer**

**SnackBar**

**UI com base na orientação (OrientationBuilder)**

**Usando fontes personalizadas (Google Fonts)**

**Usando tema de cores personalizadas (Widget Theme)**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Fontes:***

[*https://flutter.dev/*](https://flutter.dev/) *- documentação*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Guias - TabController & DefaultTabController**

**TabController:** a classe TabController é um propriedade index que opera como um índice da guia selecionada com uma animação que é executada no momento do clique. O início da guia selecionada pode ser alterado com **animateTo** .

**Como ocorre?**

Um widget stateful que constrói um TabBar ou TabBarView pode criar um TabController e compartilhá-lo automaticamente. Quando TabBar e TabBarView não têm um ancestral com estado conveniente, um TabController pode ser compartilhado fornecendo um widget herdado DefaultTabController .



Já o **DefaultTabController é um widget** herdado que é usado para compartilhar um TabController com um TabBar ou TabBarView.

**Exemplo - fragmento de code:**



* **No home,** setamos o DefaultTabController
* **Length:** determina a quantidade de guias que a classe irá renderizar;
* **TabBar:** por meio do componente “tabs” insere os elementos desejados na TabBar. Neste caso, foi escolhida a galeria de ícones do Material Design.



Neste exemplo, criou-se um Widget para inserir imagens no conteúdo das “guias” da classe.

**Widget Drawer**

Em aplicativos que usam Material Design, existem duas opções principais de navegação: guias e drawer. Quando não há espaço suficiente para apoiar as guias, o drawer é uma alternativa prática.

No Flutter, usamos o Drawer Widget para construir a combinação com Scaffold para desenvolver um layout com uma gaveta do Material Design. Uma curiosidade bem pontual: os outros dois frameworks que podemos usar para desenvolver layouts responsivos e leves no Flutter são: [Foundation](https://get.foundation/) e Bootstrap - [flutter\_bootstrap | Flutter Package](https://pub.dev/packages/flutter_bootstrap), sempre podemos fazer um intenso discovery na documentação destes recursos e utilizar no App.

Para adicionar conteúdo ao Drawer, é possível usar os recursos de Column Widget ou ListView. O ListView é útil porque permite que os usuários percorram o drawer sem que o conteúdo possa ocupar mais espaço do que a tela suporta.

|  |  |
| --- | --- |

**Exemplo de Código:**



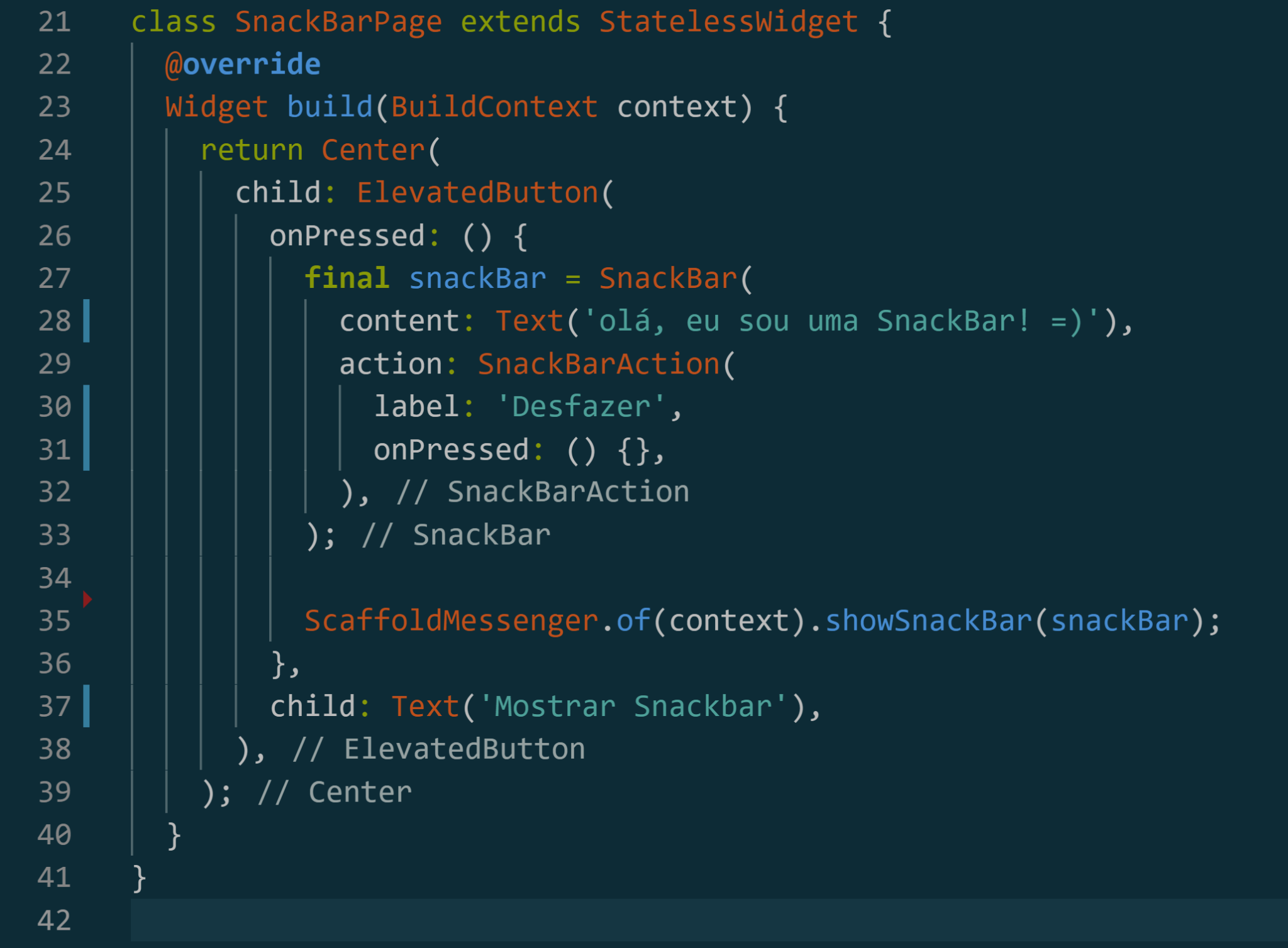
**Snackbar**

A preocupação ao criar aplicativos que seguem as diretrizes do Material Design, é aplicar aos Apps uma estrutura visual consistente e acima de tudo, agradável. Neste exemplo, o SnackBar é uma barra de informações que tem a função de sobrepor outros widgets importantes, como o FloatingActionButton e garante a entrega da mensagem.

Com o Scaffold declarado, exiba a SnackBar. Cria-se o SnackBar, em seguida, exibimos usando Scaffold Messenger. Podemos também inserir uma ação ao usuário quando a SnackBar for exibida. Por exemplo, se o usuário excluir acidentalmente uma mensagem, ele pode usar uma ação opcional no **SnackBar** para recuperar a mensagem.

|  |  |
| --- | --- |

**Exemplo de código:**



**UI se adapta a orientação do dispositivo - OrientationBuilder**

Para determinar o fluxo da orientação do aplicativo, usamos o OrientationBuilder widget. O OrientationBuilder calcula o fluxo da orientação comparando a largura e a altura disponíveis para o widget pai e reconstrói quando o tamanho do pai muda (device - contexto da aplicação).

Usando o Orientation, podemos construir uma lista que exibe duas colunas no modo retrato ou três colunas no modo paisagem, ou de acordo com o desejo.

**Classe GridView()**: Os layouts de grade mais comumente usados ​​são **GridView.count** , que cria um layout com um número fixo de blocos no eixo cruzado, e **GridView.extent** , que cria um layout com blocos que têm uma extensão máxima do eixo cruzado.

*Fonte:* [*GridView class - widgets library - Dart API*](https://api.flutter.dev/flutter/widgets/GridView-class.html)

|  |  |
| --- | --- |

**Ajustes de conteúdo na orientação retrato e paisagem.**

**Exemplo de código:**

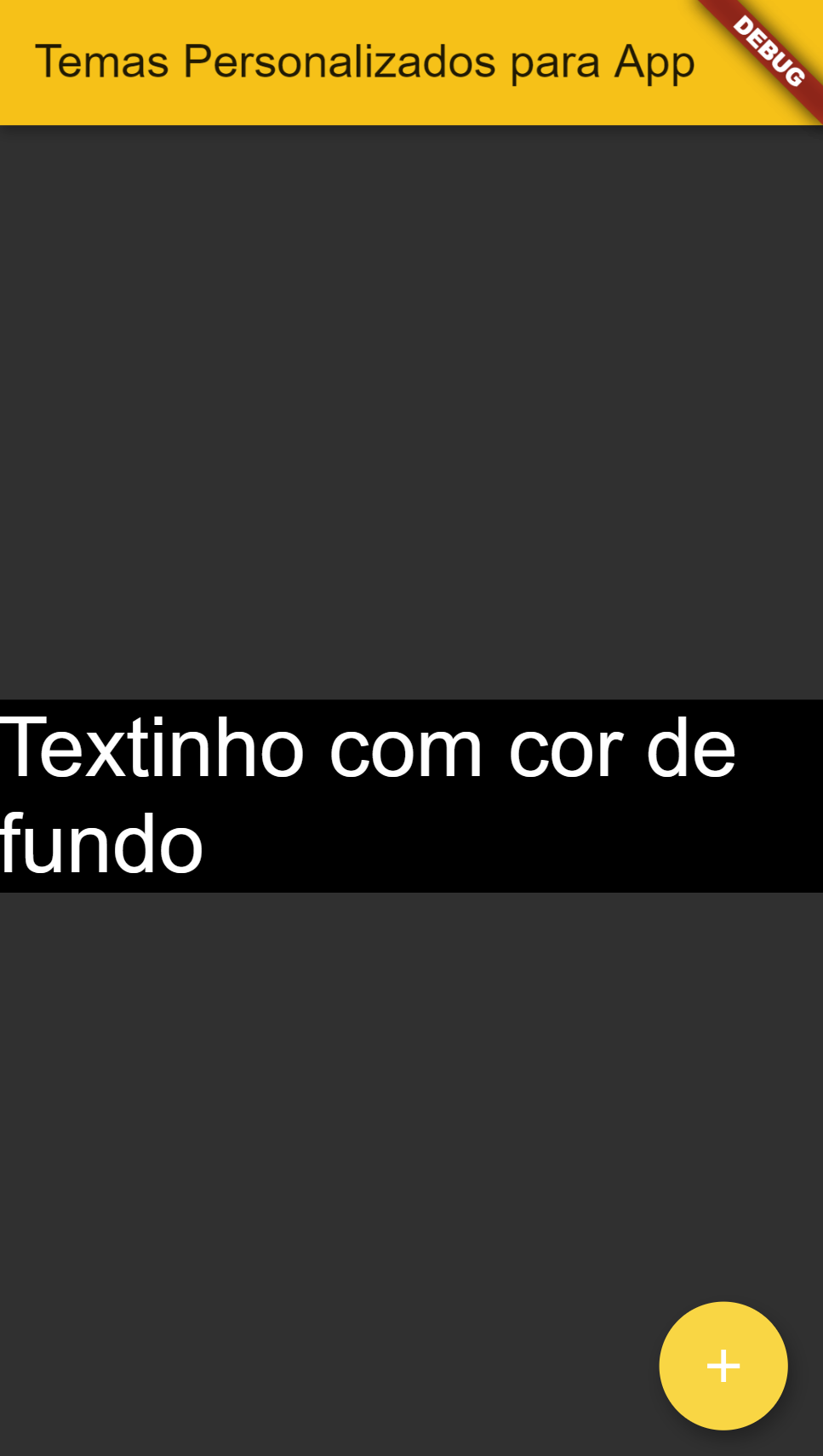


**Theme - Use temas para compartilhar cores e estilos de fonte**

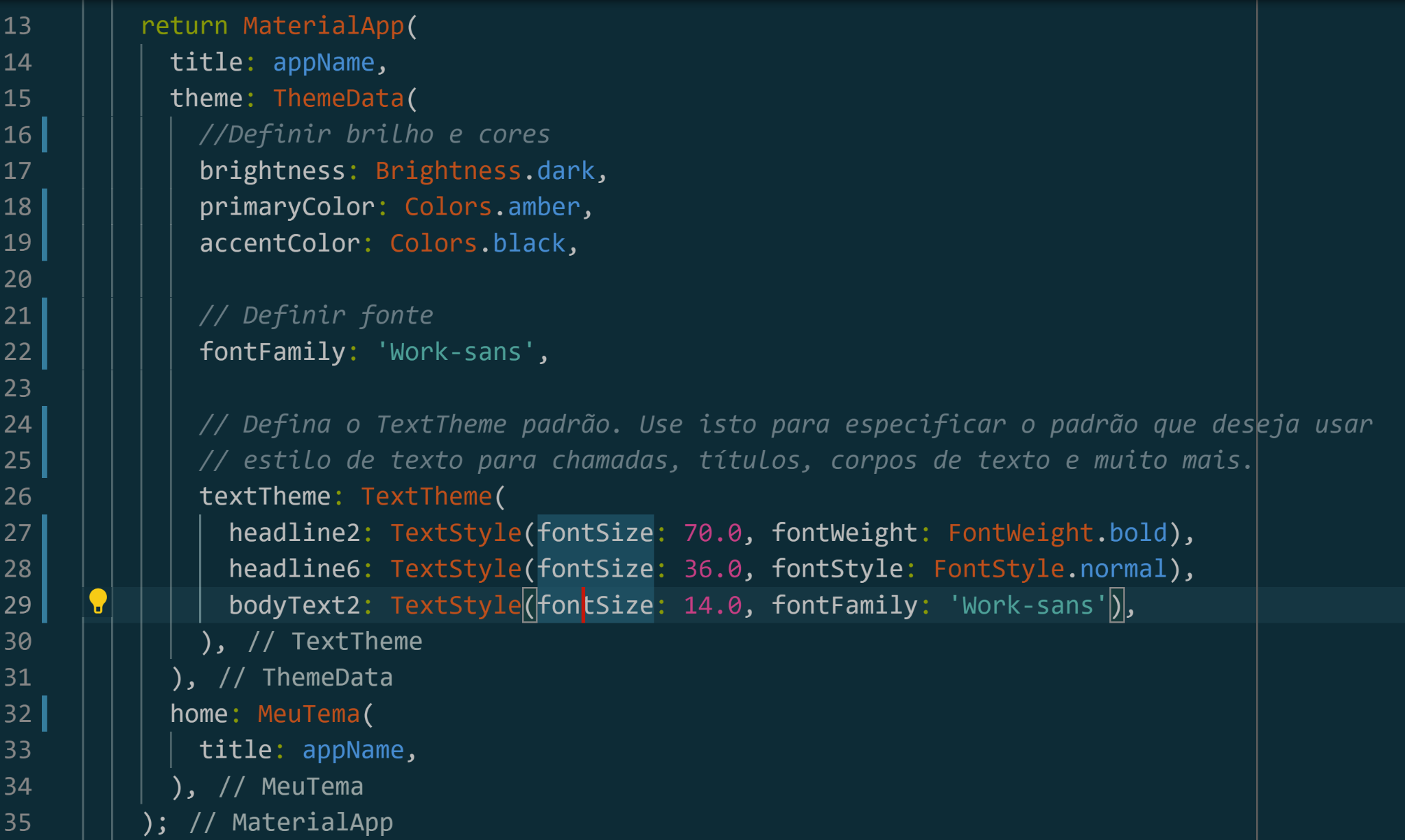
Um recurso bastante útil para desenvolvimento de layout de App é criar uma configuração do tema visual geral e personalizado para um MaterialApp ou algum widget dentro do aplicativo.

A propriedade do tema MaterialApp pode ser usada para configurar a aparência de todo o aplicativo. A subárvore de widget dentro de um aplicativo pode substituir o tema do aplicativo incluindo um widget de tema na parte superior da subárvore. Não esqueçam que os componentes de Layout do Flutter são altamente hierárquicos.

**Dicas by** [**ThemeData class - material library - Dart API**](https://api.flutter.dev/flutter/material/ThemeData-class.html)**:**

1. Os widgets cuja aparência deve estar alinhada com o tema geral podem obter a configuração do tema atual com **Theme.of**.
2. Os componentes do material geralmente dependem exclusivamente de **colorScheme** e **textTheme** .
3. Essas propriedades têm a garantia de ter valores não nulos.

**Exemplo de código:**



**Fontes Personalizadas - Google Fonts**

Flutter funciona com fontes personalizadas e podemos aplicar uma fonte personalizada em um aplicativo inteiro ou em widgets individuais! O que precisamos fazer:

1. Importe os arquivos de fontes, fazendo download do site Google Fonts
2. Declare a fonte no pubspec.
3. Defina uma fonte como padrão.
4. Use uma fonte em um widget específico.

Para trabalhar com uma fonte, vamos importar os arquivos de fonte para o projeto. É prática comum colocar arquivos de fonte em uma pasta **fonts** ou **assets** na raiz de um projeto Flutter.

Por exemplo, para importar os arquivos de fonte Raleway e Roboto Mono para um projeto, a estrutura da pasta pode ter a seguinte aparência:

| **app/**  **fonts/**  **Raleway-Regular.ttf**  **Raleway-Italic.ttf**  **RobotoMono-Regular.ttf**  **RobotoMono-Bold.ttf** |
| --- |

**Depois, precisamos declarar a(s) fonte(s) no pubspec**



