Trabalho Prático | DGT2812 Desenvolvimento de Aplicativos Móveis com Flutter

Material de **orientações** para desenvolvimento do **Trabalho Prático** da disciplina DGT2812 Desenvolvimento de Aplicativos Móveis com Fluter

📍 **O trabalho prático deve ser feito individualmente.**

DGT2812 - Desenvolvimento de Aplicativos Móveis com Fluter

Objetivos do trabalho prático

- Instalar e configurar o Flutter SDK e o ambiente de desenvolvimento integrado (IDE) de acordo com as melhores práticas;
- Empregar Widgets fundamentais, como MaterialApp, Scaffold, AppBar, Text e RaisedButton;
- Aplicar diferentes Widgets para criar uma interface visual atraente e funcional;
- Aplicar o widget ListView para exibir e gerenciar listas de dados;
- Criar e implementar funcionalidades personalizadas para um Widget específico.

Entrega

- As microatividades irão dar suporte para o desenvolvimento do trabalho Prático. Elas têm apoio/gabarito para resolução no próprio documento;
- A entrega esperada é o Trabalho Prático, descrito neste documento após as Microatividades;

👉 Atividades práticas

Posso criar um App de outra forma!

Microatividade 1: Preparação do ambiente

- Material necessário para a prática

- Flutter SDK (Pode ser baixado no site oficial do Flutter);
- Android Studio e/ou um editor de texto (Intellij idea community);
- Git para controle de versionamento;
- Emulador Android ou iOS para testar o aplicativo.

- Procedimentos

1. **Procedimentos de Instalação do Flutter no Ambiente Windows:**

a) Instalação do Git:

a. Baixe o cliente do Git para Windows a partir da página oficial do Git.

b. Execute o instalador do Git, seguindo o processo padrão de instalação (Avançar > Avançar > Concluir).

b) Download e Extração do Flutter SDK:

a. Acesse a página oficial do Flutter e baixe o arquivo zip da última versão do Flutter SDK, geralmente nomeado como flutter_windows_x.x.x-stable.zip.

b. Extraia o conteúdo do arquivo zip para uma pasta de sua escolha; recomenda-se C:\flutter para evitar privilégios de administrador.

c) Configuração das Variáveis de Ambiente:

a. Abra o Painel de Controle do Windows e vá para "Contas de usuário" > "Alterar as variáveis do meu ambiente".

b. Na seção de variáveis de ambiente de usuário, encontre a variável chamada "Path".

c. Adicione o caminho para a pasta C:\flutter\bin à variável "Path", separando a nova entrada das entradas pré-existentes com um ponto e vírgula (;).

d) Reinicialização do Sistema:

a. Reinicie o Windows para aplicar as alterações nas variáveis de ambiente.

e) Verificação da Instalação:

a. Abra uma instância do Git Bash ou CMD e digite o comando **flutter**. Isso deve exibir a ajuda do Flutter, confirmando uma instalação bem-sucedida.

f) Execução do Comando flutter doctor:

a. Abra uma linha de comando do Windows e execute o comando **flutter doctor**. Este comando fornecerá um diagnóstico completo, identificando e sugerindo soluções para quaisquer pendências na instalação.

b. Certifique-se de seguir cada passo cuidadosamente para garantir uma instalação correta e funcional do Flutter no ambiente Windows.

2. Procedimentos de Instalação do Flutter no Ambiente Linux:

a) Instalação do Git:

a. Utilize o gerenciador de pacotes de sua distribuição Linux (como apt-get) para instalar o Git. Execute o comando:

i. sudo apt-get install git-all

b) Verificação da Arquitetura do Sistema:

a. Certifique-se de que sua distribuição Linux seja de 64 bits, pois o Flutter é compatível apenas com sistemas operacionais de 64 bits.

c) Download e Extração do Flutter SDK:

a. Acesse a página oficial do Flutter e baixe a última versão do Flutter SDK para Linux, disponível na seção "Get the Flutter SDK".

b. Extraia o conteúdo do Flutter para uma pasta de sua escolha; recomenda-se criar uma pasta chamada "flutter" na raiz do seu usuário para evitar privilégios de administrador.

d) Configuração das Variáveis de Ambiente:

a. Adicione o Flutter às variáveis de ambiente usando o comando:

export PATH=$(pwd)/flutter/bin:$PATH

Lembre-se de que este comando funcionará apenas enquanto o terminal estiver aberto. Para torná-lo permanente, adicione-o ao arquivo de perfil do seu shell (como ~/.bashrc ou ~/.zshrc).

e) Verificação da Instalação: ✅

a. Abra uma instância do terminal e execute o comando **flutter**. Isso deve exibir a ajuda do Flutter, indicando que a instalação foi bem-sucedida.

f) Execução do Comando flutter doctor:

a. Execute o comando **flutter doctor** em uma linha de comandos do seu Linux. Este comando fornecerá um diagnóstico completo, apontando quaisquer pendências na instalação e sugerindo soluções.

Certifique-se de seguir cada passo cuidadosamente para garantir uma instalação correta e funcional do Flutter no ambiente Linux.

3. Para instalar no MacOS:

a) Instalação do Git:

a. Utilize o gerenciador de pacotes Homebrew para instalar o Git no macOS. Execute o comando:

i. brew install git

b) Download e Extração do Flutter SDK:

a. Acesse a página oficial do Flutter e baixe a última versão do Flutter SDK para macOS, disponível na seção "Get the Flutter SDK".

b. Extraia o conteúdo do Flutter para uma pasta de sua escolha; recomenda-se criar uma pasta chamada "flutter" na raiz do seu usuário para evitar privilégios de administrador.

c) Configuração do PATH:

a. Atualize o PATH da máquina para apontar para o Flutter SDK. Execute o comando:

i. export PATH=$(pwd)/flutter/bin:$PATH

ii. Para tornar essa configuração permanente, siga as instruções no artigo "Update your path" na documentação oficial do Flutter.

d) Verificação da Instalação: ✅

a. Abra uma instância do Terminal e execute o comando **flutter**. Isso deve exibir a ajuda do Flutter, indicando uma instalação bem-sucedida.

e) Execução do Comando flutter doctor:

a. Execute o comando **flutter doctor** em uma linha de comandos do seu macOS.

Este comando fornecerá um diagnóstico completo, identificando quaisquer pendências na instalação e sugerindo soluções.

Certifique-se de seguir cada passo cuidadosamente para garantir uma instalação correta e funcional do Flutter no ambiente macOS.

4. Procedimentos de Instalação do IntelliJ IDEA em Diferentes Plataformas:

- **Instalação no Windows:**
    - Baixe o instalador do IntelliJ IDEA para Windows.
    - Execute o instalador, seguindo o padrão Avançar > Avançar > Concluir.
    - Ao final do processo, a IDE estará instalada e pronta para uso.

b. **Instalação no Linux:**

- Baixe o arquivo do IntelliJ IDEA para Linux.
- Extraia o arquivo baixado para uma pasta específica para aplicativos no sistema operacional. Geralmente, use /usr/local/ (para uso pessoal) ou /opt/ (para usuários compartilhados).
- Abra uma instância do Terminal e navegue até a pasta onde o IntelliJ IDEA foi descompactado.
- Na pasta /bin, execute o arquivo idea.sh para iniciar o IntelliJ IDEA.
- Opcionalmente, adicione a pasta /intellij-idea/bin à variável PATH para facilitar o acesso.

c. **Instalação no macOS:**

- Abra o arquivo DMG baixado.
- Arraste o IntelliJ IDEA para a pasta Applications.
- Execute o IntelliJ IDEA.

Ao seguir esses passos, o IntelliJ IDEA estará instalado em seu sistema, independentemente da plataforma. Lembre-se de que essas instruções são específicas para o IntelliJ IDEA, mas você pode adaptá-las para outros editores de texto ou IDEs de sua escolha, se preferir. Certifique-se de verificar a documentação específica de cada ferramenta para instruções detalhadas sobre sua instalação.

5. Procedimentos de Instalação do Android Studio em Diferentes Plataformas:

a) Instalação no Windows:

a. Baixe o instalador do Android Studio para Windows.

b. Execute o instalador, seguindo o padrão Avançar > Avançar > Concluir.

c. Ao final do processo, abra o Android Studio e clique em "Configure".

d. Selecione a opção "AVD Manager" para configurar um dispositivo virtual.

b) Instalação no Linux:

a. Baixe o arquivo do Android Studio para Linux.

b. Extraia o arquivo baixado para uma pasta específica para aplicativos no sistema operacional. Geralmente, use /usr/local/ (para uso pessoal) ou /opt/ (para usuários compartilhados).

c. Abra uma instância do Terminal e navegue até a pasta onde o Android Studio foi descompactado.

d. Execute o script de inicialização (studio.sh) para iniciar o Android Studio.

e. Configure um dispositivo virtual no "AVD Manager".

c) Instalação no macOS:

a. Baixe o arquivo DMG do Android Studio.

b. Abra o arquivo DMG e arraste o Android Studio para a pasta Applications.

c. Execute o Android Studio.

d. Clique em "Configure" e selecione a opção "AVD Manager" para configurar um dispositivo virtual.

⚙ Configuração do AVD Manager:

- Ao abrir o AVD Manager, clique em "Create Virtual Device" (+).
- Selecione o modelo de smartphone, como Pixel 2.
- Avance, escolha a imagem do Android para o emulador e prossiga até a conclusão.

Ao seguir esses passos, o Android Studio estará instalado e configurado em seu sistema. Certifique-se de adaptar os passos conforme necessário para a plataforma específica que você está utilizando. Além disso, esteja ciente de que essas instruções podem variar com as versões mais recentes do Android Studio, então sempre consulte a documentação oficial para obter as informações mais atualizadas.

- Resultados esperados ✨

Ao concluir esta microatividade, você estará habilitado a desenvolver aplicações de maneira ágil, segura e flexível. Iniciando com a configuração e instalação do Flutter em sua máquina, você terá os fundamentos necessários para criar aplicativos incríveis.

Microatividade 2: Utilização de Widgets Flutter Básicos - MaterialApp, Scaffold e AppBar

- Material necessário para a prática

- Editor de texto ou IDE (Sugestões: IntelliJ ou VS Code)
- Flutter SDK
- Android Studio e/ou xCode
- Simulador Android ou iOS
- Navegador Web: Google Chrome, Firefox, MS Edge, Safari ou Opera

- Procedimentos:

1. No editor de texto ou IDE, inicie um novo projeto Flutter utilizando o comando **flutter create my_app**.
2. Navegue até a pasta do projeto e abra o arquivo **main.dart** dentro da pasta **lib**.
3. No arquivo mencionado, introduza os Widgets básicos de um aplicativo Flutter:

a) Importe os pacotes necessários do Flutter.

b) Crie uma função que retorne um MaterialApp contendo um Scaffold e algum texto, conforme abaixo:

```
import 'package:flutter/material.dart';


void main() {
  return runApp(
    MaterialApp(
      home: StatelessWidgetExemplo("Olá Flutter - MaterialApp"),
    )
  );
}
class StatelessWidgetExemplo extends StatelessWidget {
  final String _appBarTitle;
  StatelessWidgetExemplo(this._appBarTitle) : super();
```

```
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(_appBarTitle),
      ),
      body: Center(
        child: Text('Macoratti .net'),
      ),
    );
  }
}
```

4. Salve as alterações no arquivo **main.dart**.

5. Inicie o servidor com o comando **flutter run**.

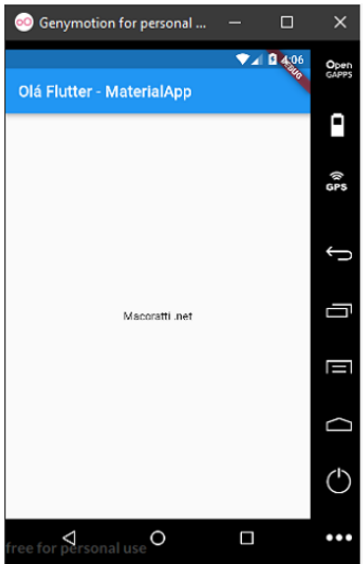
6. Verifique se o texto que você inseriu está sendo exibido no simulador. Se afirmativo, a tarefa foi concluída com sucesso.

7. Como etapa extra e opcional, considere utilizar um linter, como Dart Linter, para verificar a sintaxe e a validade do seu código.

- Resultados esperados ✨

Os resultados aguardados para esta microatividade incluem a visualização, no simulador Android ou iOS, do texto inserido pelo aluno durante a etapa 3 do roteiro. Além disso, de maneira subjetiva, espera-se que o aluno adquira uma compreensão da estrutura de Widgets básicos em um aplicativo Flutter. Essa compreensão deve abranger conceitos essenciais, como MaterialApp, Scaffold, AppBar, e outros elementos fundamentais para o desenvolvimento em Flutter.

Como referência visual do resultado esperado, a imagem a seguir apresenta um exemplo do aplicativo Flutter após a conclusão desta microatividade.



Em conjunto, esses resultados indicarão o êxito na aplicação prática dos conceitos abordados, proporcionando ao aluno uma base sólida para progredir no desenvolvimento de aplicativos utilizando o framework Flutter.

Microatividade 3: Criação de layouts básicos com os Widgets

- Material necessário para a prática

- Editor de texto ou IDE (Sugestões: IntelliJ ou VS Code)
- Flutter SDK
- Android Studio e/ou xCode
- Simulador Android ou iOS
- Navegador Web: Google Chrome, Firefox, MS Edge, Safari ou Opera

- Procedimentos

1. Crie um layout utilizando a composição de Widgets.
2. Construa o layout conforme a seguinte estrutura:

// Exemplo de uma coluna (Column) contendo um ícone (Icon) e um texto (Text).

Column(

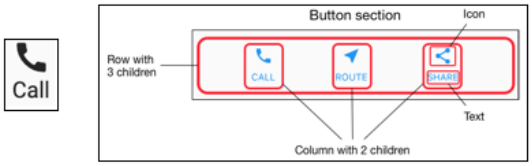
```
  children: [Icon(Icons.call), Text('Call')],
)
```

3. Utilize 3 instâncias dessa estrutura em uma linha horizontal (Row). A estrutura é a mesma, mudando apenas o ícone e o texto:

```
// Exemplo de uma linha (Row) com três colunas
Row(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: [
    Column(
      children: <Widget>[Icon(Icons.call), Text('Call')],
    ),
    Column(
      children: <Widget>[Icon(Icons.directions), Text('Route')],
    ),
    Column(
      children: <Widget>[Icon(Icons.share), Text('Share')],
    ),
  ],
)
```

- Resultados esperados ✨

Espera-se que, com esta atividade, o participante identifique alguns tipos de layouts disponíveis para a criação de aplicativos Flutter. Esses layouts ajudam a organizar os elementos na tela, utilizando Widgets como Column, Row, Text e Icon. A figura abaixo ilustra o resultado desejado, permitindo a escolha do widget de layout mais adequado com base nas necessidades do design.



Microatividade 4: Utilização do Widget ListView em Flutter

- Material necessário para a prática

- Editor de texto ou IDE (Sugestões: IntelliJ ou VS Code)
- Flutter SDK
- Android Studio e/ou xCode
- Simulador Android ou iOS
- Navegador Web: Google Chrome, Firefox, MS Edge, Safari ou Opera

- Procedimentos

1. Em certos casos, é necessário criar telas compostas por listas, como uma agenda de contatos ou uma lista de vários itens. Para isso, utilizamos o widget ListView:

```
ListTile(
  title: Text('Flutter'),
  subtitle: Text('Tudo é um widget'),
  leading: Icon(Icons.flash_on),
  trailing: Icon(Icons.keyboard_arrow_right),
)
```

2. Adicione esse widget à sua lista para criar cada item:

```
// Exemplo de uma ListView com três ListTile
ListView(
  children: [
    ListTile(
      title: Text('Flutter'),
```

```
      subtitle: Text('Tudo é um widget'),
      leading: Icon(Icons.flash_on),
      trailing: Icon(Icons.keyboard_arrow_right),
    ),
    ListTile(
      title: Text('Dart'),
      subtitle: Text('É fácil'),
      leading: Icon(Icons.mood),
      trailing: Icon(Icons.keyboard_arrow_right),
    ),
    ListTile(
      title: Text('Firebase'),
      subtitle: Text('Combina com Flutter'),
      leading: Icon(Icons.whatshot),
      trailing: Icon(Icons.keyboard_arrow_right),
    ),
  ],
)
```

- Resultados Esperados ✨

Esta microatividade possibilita a criação da interface do usuário (UI) por meio de widgets, descrevendo a aparência da visualização de acordo com sua configuração e estado atual. Ao adicionar interatividade ao seu aplicativo Flutter, você terá como resultado uma lista visualmente organizada e funcional, utilizando o widget ListView.

A imagem a seguir ilustra os dois procedimentos acima, mostrando a estrutura visual resultante da utilização do widget ListView.

Microatividade 5: Desenvolvimento de Outra Funcionalidade para o Widget em Flutter

- Material necessário para a prática

- Editor de texto ou IDE (Sugestões: IntelliJ ou VS Code)
- Flutter SDK
- Android Studio e/ou xCode
- Simulador Android ou iOS
- Navegador Web: Google Chrome, Firefox, MS Edge, Safari ou Opera

- Procedimentos

1. Em determinadas situações, é necessário empilhar um Widget sobre o outro. Observe que o texto está posicionado "acima" da imagem. O widget utilizado para criar essa pilha de widgets é o Stack:

```
Stack(
 children: [
   Container(
     width: 250,
     height: 250,
     color: Colors.blue,
   ),
   Container(
```

```
      width: 200,
      height: 200,
      color: Colors.red,
    ),
    Container(
      width: 150,
      height: 150,
      color: Colors.yellow,
    )
],
)
```

- Resultados esperados ✨

Visualize algumas definições básicas que devem ser feitas antes da customização. Utilizando o mecanismo de layout do Flutter, os Widgets, aprenderá a usá-los para construir layouts, definir itens na vertical e horizontal, listas, fotos fixas, entre outros. Este procedimento proporcionará uma compreensão prática da funcionalidade do widget Stack, permitindo a criação de sobreposições de widgets em seu aplicativo Flutter, como exemplificado nas imagens a seguir. A imagem da direita apresenta o resultado do código apresentado acima.

Trabalho Prático | Posso criar um App de outra forma!

Nesta atividade, revisaremos todos os conceitos e práticas abordados nas microatividades anteriores. Além disso, exploraremos a colocação de um único Widget na tela, abrangendo diversos aspectos presentes em alguns dos widgets de layout mais comuns que foram abordados ao longo do curso, entre outros tópicos relevantes.

Contextualização

A "Explore Mundo", uma Agência de Viagens, está em busca de melhorias para tornar seu aplicativo mais atrativo e funcional para os clientes. O objetivo é proporcionar uma experiência em que os usuários possam explorar destinos, consultar pacotes de viagens, efetuar reservas, entrar em contato com a equipe e obter informações detalhadas sobre a localização e as avaliações de estrelas para cada destino. As principais características desejadas para o app incluem:

1. **Banner de Destaque:** Um elemento visual destacado, como uma imagem ou slideshow, exibindo fotos irresistíveis dos destinos oferecidos pela agência. Cada imagem será interativa, direcionando os usuários para páginas específicas de destinos ao serem tocadas.
2. **Barra de Navegação:** Uma barra superior intuitiva, contendo links para diferentes seções do aplicativo, como "destinos", "pacotes de viagem", "contato" e "sobre nós". Essa navegação facilita o acesso dos usuários às áreas relevantes do app.
3. **Pesquisa Rápida:** Implementação de um campo de pesquisa que permitirá aos usuários inserir destinos específicos ou dados desejados, agilizando a busca por pacotes de viagem. 🔎

Além dessas melhorias para o aplicativo, a empresa tem o objetivo de ampliar seu alcance de mercado desenvolvendo um site complementar. Esse site oferecerá aos clientes acesso fácil às informações da agência e a praticidade de efetuar reservas de viagens diretamente por meio de seus dispositivos móveis. Essa estratégia visa proporcionar uma experiência integrada e conveniente para os clientes explorarem as ofertas da agência, tanto no aplicativo quanto no site.

Roteiro de prática

- Material necessário para a prática

- Editor de texto ou IDE (Sugestões: IntelliJ ou VS Code)
- Flutter SDK
- Android Studio e/ou xCode
- Simulador Android ou iOS
- Navegador Web: Google Chrome, Firefox, MS Edge, Safari ou Opera

- Procedimentos

1. **Configuração do Ambiente:**
    a. Certifique-se de ter seu ambiente configurado.
    b. Crie um novo aplicativo Flutter.
2. **Estrutura Inicial:**
    a. Substitua o conteúdo do arquivo **lib/main.dart** pelo código fornecido, o qual configura uma estrutura básica para o aplicativo.

```
import 'package:flutter/material.dart';


void main() => runApp(const MyApp());


class MyApp extends StatelessWidget {
  const MyApp({super.key});


  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter layout demo',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Flutter layout demo'),
        ),
        body: const Center(
          child: Text('Hello World'),
        ),
      ),
    );
```

}

}

3. **Divisão do Layout:**

    a. Analise o layout dividindo-o em elementos básicos.

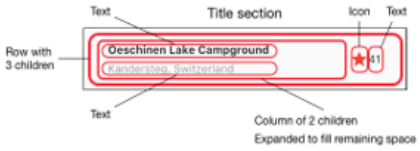
    b. Identifique linhas, colunas e áreas que requerem alinhamento, preenchimento ou bordas.

    c. Verifique a necessidade de grade, sobreposição de elementos ou guias de interface.

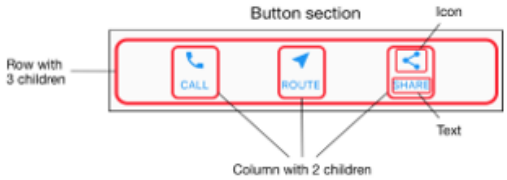
4. **Diagramação do Layout:**

a) Comece identificando os elementos principais do seu layout. Em nosso exemplo, temos quatro elementos dispostos em uma coluna: uma imagem, duas linhas e um bloco de texto.

b) Em seguida, faça um diagrama para cada uma das linhas. A primeira linha, que chamaremos de "Seção Título", possui três elementos filhos: uma coluna de texto, um ícone de estrela e um número. A coluna de texto tem duas linhas de texto dentro dela. Para garantir que essa primeira coluna ocupe o espaço adequado, é aconselhável envolvê-la em um widget "Expandido".



c) A segunda linha, que chamaremos de "Seção Button", também possui três elementos filhos. Cada um desses filhos é uma coluna que contém um ícone e um texto.



d) Uma vez que você tenha diagnosticado o layout do seu aplicativo, é aconselhável adotar uma abordagem de construção "de baixo para cima" para implementá-lo. Isso significa começar pelos elementos mais internos e, gradualmente, construir a estrutura do layout de forma progressiva.

e) Para manter o código organizado e evitar a confusão visual de um código de layout profundamente aninhado, considere a possibilidade de criar variáveis e funções para partes específicas da implementação. Isso ajudará a tornar o código mais legível e fácil de manter.

5. **Construção da Seção de Título:**

    a. Desenvolva a coluna esquerda na seção do título.

    b. Utilize o código fornecido para criar a estrutura, com destaque para o uso do Widget Expandido e do Container para gerenciar espaçamentos e estilo.

    c. Altere os textos e imagens de acordo com as necessidades do cliente.

    d. Utilize o código de exemplo a seguir para construir o seu próprio aplicativo:

```
// Passo 5: Construindo a Coluna da Seção Título
Widget titleSection = Container(
  padding: const EdgeInsets.all(32),
  child: Row(
    children: [
      Expanded(
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Container(
              padding: const EdgeInsets.only(bottom: 8),
              child: const Text(
                'Oeschinen Lake Campground',
                style: TextStyle(
                  fontWeight: FontWeight.bold,
                ),
              ),
            ),
            Text(
```

```
            'Kandersteg, Switzerland',
            style: TextStyle(
              color: Colors.grey[500],
            ),
          ),
        ],
      ),
    ),
    Icon(
      Icons.star,
      color: Colors.red[500],
    ),
    const Text('41'),
  ],
 ),
);
```

Neste passo, estamos construindo a coluna esquerda da seção de título. Usamos um **Container** para envolver os elementos da coluna e aplicar um espaçamento uniforme em todas as bordas. Dentro da coluna, usamos um **Expanded** para ocupar todo o espaço restante na linha horizontal. Isso é importante para garantir que a coluna ocupe o espaço disponível.

A propriedade **crossAxisAlignment** é definida como **CrossAxisAlignment.start** para alinhar o conteúdo da coluna à esquerda da linha.

Dentro da coluna, temos dois elementos de texto, onde o primeiro é exibido em negrito e o segundo em cinza. Em seguida, adicionamos um ícone de estrela vermelha e o texto "41" à direita da coluna.

Este trecho de código representa a construção da seção de título do aplicativo Flutter. Para incorporá-lo ao corpo do aplicativo, você pode fazer o seguinte:

```
return MaterialApp(
  title: 'Demonstração de layout Flutter',
  home: Scaffold(
    appBar: AppBar(
      title: const Text('Demonstração de layout Flutter'),
    ),
    body: Column(
      children: [
        titleSection,
        // Outros elementos do corpo do aplicativo podem ser adicionados aqui.
      ],
    ),
  ),
);
```

Assim, a seção de título é adicionada como parte do corpo do aplicativo.

6. **Construção da Seção de Botões:**
    a. Crie um método auxiliar, **_buildButtonColumn()**, para gerar colunas de botões de forma eficiente.
    b. Utilize esse método para construir a linha de botões, alinhando as colunas uniformemente com **MainAxisAlignment.spaceEvenly**.
    c. Utilize o código de exemplo a seguir para construir o seu próprio aplicativo:

```
// Passo 6: Construindo a Seção de Botões
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
```

```
@override
Widget build(BuildContext context) {
 // ...

 Column _buildButtonColumn(Color color, IconData icon, String label) {
  return Column(
   mainAxisSize: MainAxisSize.min,
   mainAxisAlignment: MainAxisAlignment.center,
   children: [
    Icon(icon, color: color),
    Container(
     margin: const EdgeInsets.only(top: 8),
     child: Text(
      label,
      style: TextStyle(
       fontSize: 12,
       fontWeight: FontWeight.w400,
       color: color,
      ),
     ),
    ),
   ],
  );
 }

 Color color = Theme.of(context).primaryColor;
```

```
    Widget buttonSection = Row(
      mainAxisAlignment: MainAxisAlignment.spaceEvenly,
      children: [
        _buildButtonColumn(color, Icons.call, 'CALL'),
        _buildButtonColumn(color, Icons.near_me, 'ROUTE'),
        _buildButtonColumn(color, Icons.share, 'SHARE'),
      ],
    );

    // ...

    return MaterialApp(
      title: 'Demonstração de layout Flutter',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Demonstração de layout Flutter'),
        ),
        body: Column(
          children: [
            titleSection,
            buttonSection,
            // Outros elementos do corpo do aplicativo podem ser adicionados aqui.
          ],
        ),
      ),
    );
  }
}
```

Neste passo, estamos construindo a seção de botões do aplicativo Flutter. Criamos um método auxiliar privado **_buildButtonColumn** que aceita uma cor, um ícone e um rótulo, e retorna uma coluna com esses elementos.

Em seguida, chamamos esse método para construir três colunas de botões usando o mesmo layout, mas com diferentes ícones e rótulos. Essas colunas são organizadas em uma linha (**Row**) e alinhadas uniformemente ao longo do eixo principal usando **MainAxisAlignment.spaceEvenly**.

Por fim, adicionamos a seção de botões como parte do corpo do aplicativo, abaixo da seção de título. O restante do código permanece inalterado.

7. **Definição da Seção de Texto:**
    a. Estabeleça a seção de texto como uma variável.
    b. Utilize um Container para gerenciar o preenchimento e adicione o texto desejado, configurando **softWrap** para quebras de linha automáticas.
    c. Utilize o código de exemplo a seguir para construir o seu próprio aplicativo:

```
// Passo 7: Definindo a Seção de Texto
Widget textSection = Container(
  padding: const EdgeInsets.all(32),
  child: const Text(
    'O Lago Oeschinen fica aos pés do Blüemlisalp nos Alpes Berneses. Situado a 1.578 metros acima do nível do mar, é um dos lagos alpinos mais amplos. Um passeio de teleférico a partir de Kandersteg, seguido por meia hora de caminhada por pastagens e floresta de pinheiros, leva você ao lago, que aquece até 20 graus Celsius no verão. As atividades desfrutadas aqui incluem remo e andar no tobogã de verão.',
    softWrap: true,
  ),
);
```

```
// ...

return MaterialApp(
  title: 'Demonstração de layout Flutter',
  home: Scaffold(
    appBar: AppBar(
      title: const Text('Demonstração de layout Flutter'),
    ),
    body: Column(
      children: [
        titleSection,
        buttonSection,
        textSection,
        // Outros elementos do corpo do aplicativo podem ser adicionados aqui.
      ],
    ),
  ),
);
```

Neste passo, estamos definindo a seção de texto do aplicativo Flutter. O texto é colocado em um **Container** com preenchimento ao longo de todas as bordas. Usamos o widget **Text** para exibir o texto e configuramos **softWrap** como verdadeiro para permitir que as linhas de texto preencham a largura da coluna antes de serem quebradas no limite da palavra.

Depois, adicionamos a seção de texto ao corpo do aplicativo, abaixo das seções de título e botões. Isso completa a construção da interface do aplicativo, e você pode continuar adicionando outros elementos ao corpo conforme necessário. O restante do código permanece inalterado.

8. **Adicionando uma Imagem:**

    a. Crie um diretório **images** e adicione a imagem **lake.jpg**.

    b. Atualize o arquivo **pubspec.yaml** para incluir a imagem como um recurso.

    c. Referencie a imagem no código, ajustando seu tamanho e comportamento com **BoxFit.cover**.

    d. Utilize os passos e o código de exemplo a seguir para construir o seu próprio aplicativo:

Para adicionar uma imagem ao seu aplicativo, siga estas etapas:

1. Crie um diretório chamado "images" no diretório principal do seu projeto, se ainda não existir.
2. Adicione o arquivo da imagem que você deseja colocar no aplicativo dentro da pasta “images”.
3. Atualize o arquivo "pubspec.yaml" para incluir uma seção "assets", que disponibiliza a imagem para o seu código. Certifique-se de que o arquivo "pubspec.yaml" fique semelhante ao seguinte exemplo (a imagem de exemplo foi chamada de lake.jpg):

```
flutter:
  assets:
    - images/lake.jpg
```

Agora que a imagem está disponível para o seu aplicativo, você pode referenciá-la em seu código da seguinte maneira:

```
// ...

body: Column(
  children: [
    Image.asset(
      'images/lake.jpg',
      width: 600,
```

```
      height: 240,
      fit: BoxFit.cover,
    ),
    titleSection,
    buttonSection,
    textSection,
    // Outros elementos do corpo do aplicativo podem ser adicionados aqui.
  ],
),

// ...
```

Neste código, usamos **Image.asset** para carregar a imagem selecionada (no exemplo acima o arquivo "lake.jpg") do diretório de ativos "images". Configuramos a largura e a altura da imagem e usamos **BoxFit.cover** para garantir que a imagem seja dimensionada para cobrir toda a caixa de renderização.

Com essas etapas, você adicionou com sucesso uma imagem ao seu aplicativo Flutter e a exibiu na interface do usuário. Você pode personalizar o posicionamento e o tamanho da imagem conforme necessário para atender aos requisitos do seu aplicativo.

Lembre-se de adaptar o código e as imagens de acordo com o seu projeto.

9. Organizando os Elementos em uma ListView:

a. Substitua o uso de **Column** por **ListView** para suportar rolagem em dispositivos menores.

b. Utilize os passos e o código de exemplo a seguir para construir o seu próprio aplicativo:

Nesta etapa final, organizaremos todos os elementos em uma **ListView** em vez de uma **Column**. Isso é útil quando o aplicativo precisa de suporte à rolagem do corpo em dispositivos com telas menores.

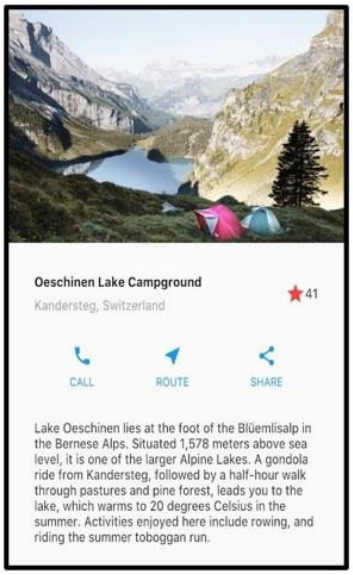
```
return MaterialApp(
  title: 'Demonstração de layout Flutter',
  home: Scaffold(
    appBar: AppBar(
      title: const Text('Demonstração de layout Flutter'),
    ),
    body: ListView(
      children: [
        Image.asset(
          'images/lake.jpg',
          width: 600,
          height: 240,
          fit: BoxFit.cover,
        ),
        titleSection,
        buttonSection,
        textSection,
        // Outros elementos do corpo do aplicativo podem ser adicionados aqui.
      ],
    ),
  ),
);
```

Neste código, substituímos a Column por uma ListView. A ListView permite que os elementos sejam rolados verticalmente quando a tela do dispositivo é pequena, garantindo

que todos os conteúdos sejam acessíveis. Os elementos, como a imagem, a seção do título, a seção de botões e a seção de texto, são adicionados como filhos da ListView. Agora, o corpo do aplicativo oferece suporte à rolagem quando necessário, proporcionando uma experiência de usuário mais flexível em diferentes tamanhos de tela.

- Resultados esperados ✨

Os passos anteriores visam a construção de um aplicativo completo com a estrutura básica do Flutter, garantindo que as interfaces de usuário sejam responsivas e acessíveis em dispositivos móveis, tablets e desktops. Essa plataforma permitirá que os clientes compartilhem avaliações e comentários sobre suas experiências de viagem. Além disso, a Agência de viagens utilizará widgets interativos do Flutter para exibir as avaliações dos clientes e interagir com eles de forma envolvente, compartilhando suas opiniões e proporcionando uma experiência interativa e informativa aos usuários do site. A imagem a seguir ilustra um exemplo de aplicativo desenvolvido. Adapte os códigos apresentados para que você possa criar a sua solução.



O aplicativo foi finalizado.

📌 *Referências*

Não foram utilizadas referências bibliográficas para a elaboração das atividades.

Entrega da prática

Chegou a hora, gamer!

✍ Armazene o projeto em um repositório no GIT.

✍ Anexar a documentação do projeto (PDF) no GIT.

✍ Compartilhe o link do repositório do GIT com o seu tutor para correção da prática, por meio da **Sala de Aula Virtual**, na aba "**Trabalhos**" do respectivo nível de conhecimento.

✍ **Ei, verifique o prazo de entrega deste trabalho pois não aceitamos trabalhos fora do prazo!**