Flutter

Bootcamp: Desenvolvedor Mobile Apps

Guilherme Assis

2020

Flutter

Bootcamp: Desenvolvedor Mobile Apps

Guilherme Assis

Sumário

Capítulo 1. Introdução

Esta apostila é um material complementar as vídeoaulas. Ela não deve ser considerada como fonte principal de estudo, e sim como uma fonte de apoio as aulas gravadas, que são o foco principal do Bootcamp.

Neste capítulo faremos uma breve introdução ao Flutter, passando por seu funcionamento, ambiente de desenvolvimento e uma visão geral da linguagem de programação utilizada para o desenvolvimento, o Dart.

Flutter

O Flutter é um framework para desenvolvimento mobile multiplataforma criado pelo Google. Com uma mesma base de código o desenvolvedor consegue criar aplicações que rodam em Android e iOS. A equipe do Flutter tem trabalhado para que também seja possível criar aplicações web e desktop com esta mesma base de código, porém este assunto foge do nosso escopo atual.

Figura 1 – Flutter.

Fonte: https://flutter.dev/.

O Flutter utiliza a linguagem de programação Dart para o desenvolvimento. O framework faz a conversão do código desenvolvido em Dart para um código nativo de acordo com a plataforma, Android ou iOS. Isso garante ao Flutter uma performance

muito boa, pois o código elaborado irá rodar de forma nativa, garantindo assim performance similar ao desenvolvido nativamente em cada plataforma.

Este ponto é uma das grandes vantagens do Flutter frente a outros frameworks como por exemplo React Native e Ionic, pois o React Native depende de uma ponte, na qual é transportado o código JavaScript para que possa ser executado, enquanto o Ionic depende de uma WebView. O fato de o Flutter não depender de estratégias desse tipo para a execução lhe confere um ganho de performance frente aos concorrentes.

Em relação a renderização dos elementos de interface, o Flutter não utiliza os componentes nativos de cada plataforma. Ao invés disso ele utiliza uma engine de renderização muito usada em jogos, chamada Skia. Essa engine é a responsável por desenhar os componentes em tela, diretamente em um Canvas.

Esses componentes são desenvolvidos pela própria equipe do Flutter, fazendo com que eles sejam idênticos aos renderizados nativamente pelas plataformas. Essa utilização da engine faz com que animações também sejam bem fluidas. Esses componentes são chamados de widgets, e já estão embutidos no próprio framework, que podem ser utilizados sem a necessidade de instalação de bibliotecas adicionais para a maioria das situações.

A documentação do Flutter pode ser acessada pelo site oficial (https://flutter.dev/) e é bem completa e explicativa, sendo uma excelente fonte de consulta. A instalação do Flutter é bem simples, basta seguir o passo a passo do próprio site. Nas aulas gravadas é realizada a instalação no Windows. Para os demais ambientes o recomendado é seguir as instruções da documentação.

O Flutter possui plug-ins para os ambientes de desenvolvimento do Android Studio, IntelliJ IDEA e Visual Studio Code. Nas aulas gravadas foram utilizados o Android Studio para desenvolvimento, mas fica a critério de cada aluno utilizar algum dos outros ambientes suportados.

Dart

A linguagem de programação utilizada para o desenvolvimento em Flutter é o Dart. Ela é uma linguagem tipada, que foi criada pelo Google em 2011, e que tinha como principal objetivo substituir o JavaScript nos navegadores. Esse objetivo acabou não se concretizando, porém, a linguagem foi utilizada para o desenvolvimento com o Flutter, ganhando muito assim em popularidade.

O Dart é muito flexível, as suas formas de compilação permitem que ele seja executado em ambientes nativos, no caso dos aplicativos mobile, como também em ambientes web. Sua sintaxe é parecida com a de linguagens muito conhecidas, como por exemplo Java e C#. Além disso ele tenta simplificar a utilização de várias estruturas, apresentando também características de linguagens funcionais.

O Dart permite que o desenvolvedor trabalhe com o conceito de programação orientada a objetos, possuindo mecanismos para criação de classes, herança e polimorfismo.

Figura 2 – Dart.

Fonte: https://dart.dev/.

A imagem abaixo representa um trecho de código Dart. Nela podemos ver o método void main que é o ponto de entrada da execução, e dentro desse método é instanciado um objeto do tipo Aluno. A classe Aluno é criada herdando da classe Pessoa seus atributos e métodos. Na classe Aluno ainda é criada um atributo a mais,

o de matrícula. Além disso essa classe realiza uma sobrescrita de um método, substituindo a implementação original da classe mãe.

Figura 3 – Código Dart.

```dart
void main() {
  var a = Aluno();
  a.nome = "Fulano";
  a.cpf = 12345678911;
  a.matricula = 123456;
  a.ola();
}

abstract class Pessoa {
  String nome;
  int cpf;

  ola() {
    print("Meu nome é $nome");
  }
}

class Aluno extends Pessoa {
  int matricula;

  ola() {
    print("Nome do aluno: $nome");
  }
}
```

Nas aulas gravadas, o Dart é visto com mais detalhes. É realizado um overview da linguagem, passando pela maioria das suas estruturas, antes de iniciar as aulas sobre Flutter.

Capítulo 2. Widgets do Flutter

No Flutter, as interfaces são construídas a partir da composição de widgets, que podem ser vistos como componentes. A documentação do Flutter diz que este conceito foi inspirado no React. Neste capítulo, faremos uma breve introdução a alguns dos widgets mais utilizados em Flutter. Maiores informações a respeito dos widgets podem ser encontradas na própria documentação do Flutter a partir do seguinte link: https://flutter.dev/docs/development/ui/widgets.

StatelessWidget

As telas no Flutter podem ser construídas utilizando o StatelessWidget ou o StatefulWidget. O StatelessWidget é utilizado quando a tela não precisa ser alterada de acordo com algum estado.

A imagem abaixo ilustra a criação de uma classe chamada Hello, que estende um StatelessWidget. Essa classe irá mostrar os elementos retornados no método build de forma estática, sem realizar alterações de acordo com modificações no estado.

Figura 4 – StatelessWidget.

```
class Hello extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Material(
        color: Colors.white,
        child: Column(
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
          children: <Widget>[
            Image.asset(
              "assets/logo.png",
              width: 90.0,
            ), // Image.asset
```

No Flutter, estado quer dizer uma informação que pode ser lida durante a montagem do widget e que também pode ser modificada durante a sua existência. O StatefulWidget tem essa capacidade de ser atualizado quando algum estado for modificado. Para isso, o programador deve chamar o método setState quando quiser notificar alguma atualização para a interface.

A imagem abaixo ilustra a criação de um StatefulWidget. Podemos observar que ele é criado a partir de uma herança utilizando a palavra-chave extends. Podemos ver também a utilização do método setState, que tem por função neste exemplo notificar a tela que a variável quantidade teve seu valor alterado. Neste momento os widgets que estiverem utilizando essa variável serão notificados e atualizarão seu comportamento.

Figura 5 – StatefulWidget.

```
class Contador extends StatefulWidget {
  @override
  _ContadorState createState() => _ContadorState();
}

class _ContadorState extends State<Contador> {
  var quantidade = 0;

  incrementar() {
    setState(() {
      quantidade++;
    });
  }

  decrementar() {
    setState(() {
      quantidade--;
    });
  }
```

Column

O widget Column é utilizado para dispor elementos na vertical. Ele possui uma propriedade chamada children, na qual é possível inserir os elementos ficarão dentro dele, sendo possível inserir vários elementos.

A disposição desses elementos no eixo principal, ou seja, na vertical, pode ser controlada a partir da propriedade chamada “mainAxisAlignment”. Esta propriedade aceita as seguintes opções:

- center: dispõe os elementos filhos no centro da Column.
- start: dispõe os elementos filhos no início da Column.
- end: dispõe os elementos filhos no fim da Column.
- spaceAround: dispõe o espaço vazio igualmente entre os filhos, e também metade desse espaço antes do primeiro e depois do último filhos.
- spaceBetween: dispõe os elementos filhos dentro da Column com espaçamento somente entre os filhos.
- spaceEvenly: dispõe o espaço livre igualmente entre os elementos filhos, incluindo antes e depois do primeiro e último elemento.

Em relação a disposição dos elementos no eixo secundário, no caso da Column o eixo horizontal é feito utilizando a propriedade CrossAxisAlignment. Os valores possíveis para essa propriedade são iguais aos do MainAxisAlignment, com a diferença que atuaram no outro eixo. A imagem abaixo ilustra a utilização de uma Column.

Figura 6 – Column.

```
Column(
  children: <Widget>[
    ListTile(
      title: Text('Quantidade', style: TextStyle(fontSize: 30),)
    ), // ListTile
    Text('$quantidade', style: TextStyle(fontSize: 40)),
    ButtonBar(
      children: <Widget>[
        FlatButton(
          child: Text('Incrementar', style: TextStyle(fontSize: 20)),
          onPressed: incrementar,
        ), // FlatButton
        FlatButton(
          child: Text('Decrementar', style: TextStyle(fontSize: 20)),
          onPressed: decrementar,
        ) // FlatButton
      ], // <Widget>[]
    ) // ButtonBar
  ], // <Widget>[]
), // Column
```

Row

O widget Row é utilizado para dispor elementos na horizontal. Assim como o Column, ele também possui uma propriedade chamada children, na qual é possível inserir vários elementos que ficarão dentro dele.

A disposição dos elementos, tanto na horizontal (eixo principal), quanto na vertical (eixo secundário), ocorre de forma similar ao dito no tópico da Column, com os mesmos valores para as propriedades MainAxisAlignment e CrossAxisAlignment. A diferença aqui é que na Row o eixo principal é o eixo horizontal, enquanto no Column o principal é o vertical. A imagem abaixo ilustra a utilização de um Row.

Figura 7 – Row.

```
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: <Widget>[
    Text('Hello', style: TextStyle(
        fontSize: 40.0,
        fontStyle: FontStyle.italic,
    )), // TextStyle, Text
    Padding(
      //padding: const EdgeInsets.all(0.0),
      padding: const EdgeInsets.only(left: 20, right: 20),
      child: Text('World', style: TextStyle(
          fontSize: 40.0,
          fontWeight: FontWeight.w800
      )), // TextStyle, Text
    ), // Padding
    Text('John!',
        style: TextStyle(
            fontSize: 40.0,
            backgroundColor: Colors.black,
            color: Colors.white
        )) // TextStyle, Text
  ], // <Widget>[]
), // Row
```

Center

O Center é um widget do Flutter que centraliza seu filho dentro dele. Caso não seja definido suas dimensões, ele assume o maior tamanho possível, tanto na largura quanto na altura.

As imagens abaixo, extraídas da própria documentação do Flutter ilustram essa situação. O widget Center recebe apenas um elemento, através da propriedade

child. Caso o desenvolvedor queira colocar mais filhos nessa estrutura, ele pode utilizar outro widget de layout que aceita vários filhos, como por exemplo um Column.

Figura 8 – Center.

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Container(
      decoration: BoxDecoration(color: Colors.white),
      child: Center(
        child: Text(
          'Hello World',
          textDirection: TextDirection.ltr,
          style: TextStyle(
            fontSize: 32,
            color: Colors.black87,
          ),
        ),
      ),
    );
  }
}
```

Fonte: https://flutter.dev/.

Figura 9 – Center

Fonte: https://flutter.dev/.

Container

O Container é um widget que pode ser utilizado para auxiliar na estilização de outro elemento. Ele pode receber um elemento filho como parâmetro, por exemplo um Text. Caso nenhum outro parâmetro seja passado a ele, o layout não mostrará diferença em relação a ter inserido somente este Text na tela. O desenvolvedor tem a opção de informar parâmetros no Container, que irão refletir diretamente no Text que está dentro dele. Dentre outras coisas, é possível informar uma borda e cor de fundo, por exemplo.

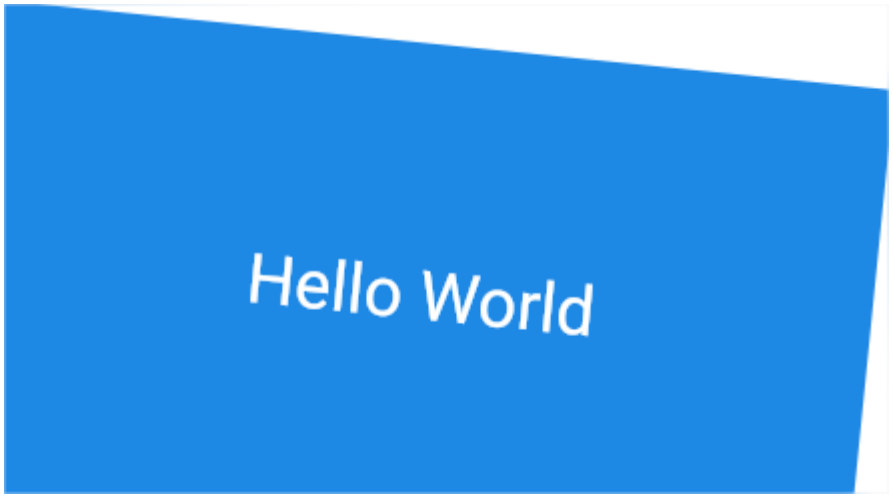
O código abaixo, retirado da própria documentação do Flutter, ilustra a utilização do container. Nele é criado um container que possui como filho um Text, e nesse container é configurado um plano de fundo azul com uma rotação, além de centralizar seu elemento filho, no caso o Text, no centro do container.

Figura 20 – Container.

```
Container(
  constraints: BoxConstraints.expand(
    height: Theme.of(context).textTheme.headline4.fontSize * 1.1 + 200.0,
  ),
  padding: const EdgeInsets.all(8.0),
  color: Colors.blue[600],
  alignment: Alignment.center,
  child: Text('Hello World',
    style: Theme.of(context)
        .textTheme
        .headline4
        .copyWith(color: Colors.white)),
  transform: Matrix4.rotationZ(0.1),
)
```

Fonte: https://flutter.dev/.

Figura 31 – Container.



Fonte: https://flutter.dev/.

Scaffold

O widget Scaffold é de grande utilidade no desenvolvimento de aplicativos com Flutter, pois ele já fornece ao desenvolvedor a implementação de um layout

básico pronto. Componentes utilizados com frequência em aplicativos já vêm configurados neste widget, bastando ao desenvolvedor informar o que deseja inserir cada um deles.

Exemplos de elementos que já vem configurados são o AppBar, Drawers (menu do tipo “hambúrguer”), snack bars (mensagens), barras de botões, body e botões flutuantes. As imagens abaixo, retiradas da própria documentação do Flutter, ilustra a utilização deste widget. No código são utilizados os parâmetros do AppBar (barra superior do aplicativo), body (conteúdo que fica no corpo do aplicativo) e um botão flutuante que fica no canto inferior direito.

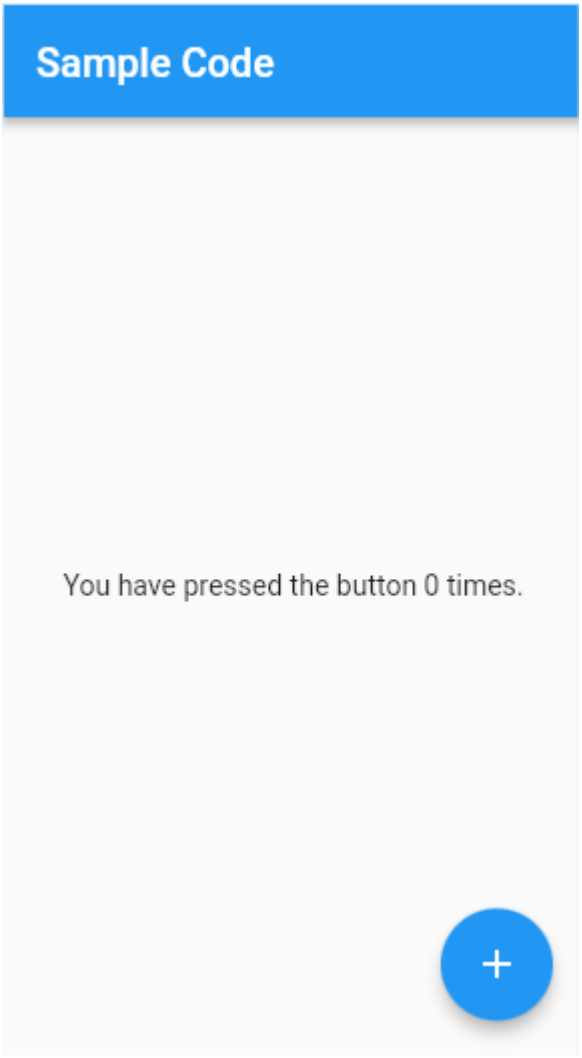
Figura 42 – Scaffold.

```
int _count = 0;

Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Sample Code'),
    ),
    body: Center(
      child: Text('You have pressed the button $_count times.')
    ),
    floatingActionButton: FloatingActionButton(
      onPressed: () => setState(() => _count++),
      tooltip: 'Increment Counter',
      child: const Icon(Icons.add),
    ),
  );
}
```

Fonte: https://flutter.dev/.

Figura 53 – Scaffold.



Fonte: https://flutter.dev/.

Text

O widget Text é utilizado para mostrar um texto na tela. O texto que o desenvolvedor deseja mostrar pode ser passado como primeiro parâmetro. Para estilizar o texto pode ser utilizado um TextStyle, permitindo alterar informações como, por exemplo, tamanho e cor do texto.

A imagem abaixo ilustra a utilização do Text juntamente com o TextStyle. Na imagem é criado um “Hello”, e o TextStyle utilizado para colocar o tamanho da fonte como 40 e o estilo da fonte como itálico.

Figura 64 – Text.

```
Text('Hello', style: TextStyle(
    fontSize: 40.0,
    fontStyle: FontStyle.italic,
)), // TextStyle, Text
```

Image

O widget Image é utilizado para colocar uma imagem na tela. Antes de utilizar a imagem, é preciso copiá-la para dentro do projeto, em uma pasta escolhida, e informá-la dentro do arquivo pubspec.yaml. A imagem abaixo ilustra o local que essa imagem deve ser configurada, devendo ficar dentro da seção chamada flutter. Nesse arquivo, a indentação é importante, se estiver incorreta o arquivo não será compilado. Após a alteração pode ser realizar um “pub get” pela própria IDE, para atualizar os conteúdos.

Figura 75 – Image.

```
flutter:
  uses-material-design: true
  assets:
    - assets/logo.png
```

Após ter realizado a configuração da imagem no pubspec.yaml, o widget Image pode então ser utilizado para colocar uma imagem na tela. Para isso basta

informar como primeiro parâmetro o caminho da imagem que foi configurada. A imagem abaixo ilustra essa utilização.

Figura 86 – Image.

```
Image.asset(
  "assets/logo.png",
  width: 90.0,
), // Image.asset
```

Button

No Flutter, um botão muito utilizado RaisedButton, que gera um botão com certa elevação. Uma variação dele é o FlatButton, que gera o botão sem elevação. Ele possui uma propriedade chamada onPressed, que recebe uma função como parâmetro que será executada sempre que o botão for clicado. A imagem abaixo ilustra esta utilização.

Figura 97 – Button.

```
FlatButton(
  onPressed: testeButton,
  color: Colors.lightGreen,
  child: Text(
    'Teste',
    style: TextStyle(fontSize: 20.0),
  ), // Text
), // FlatButton
```

ListView

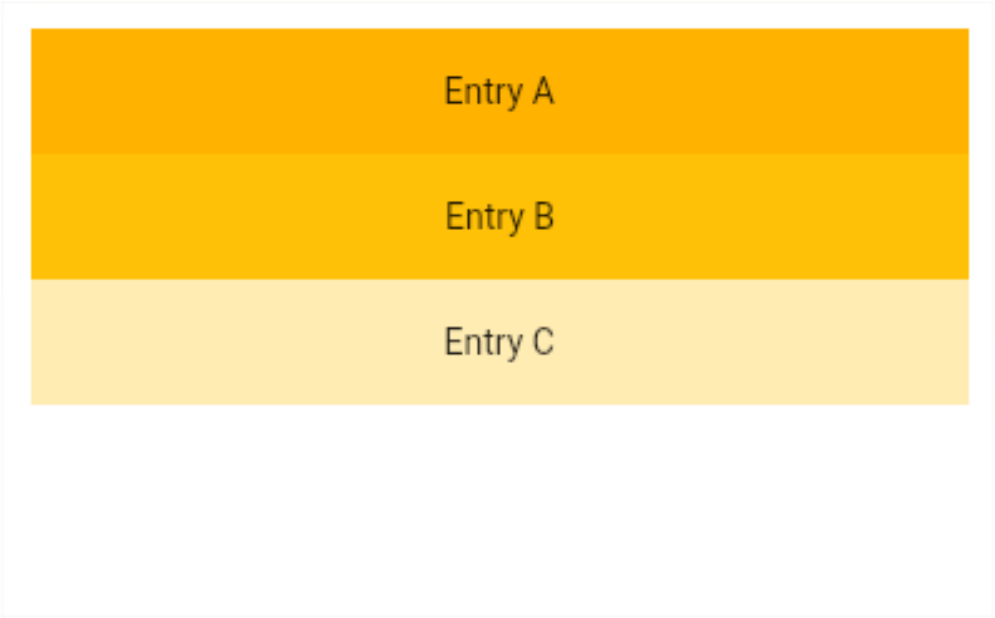
O widget ListView é utilizado para mostrar uma lista de itens na tela, permitindo que o usuário consiga realizar um scroll para ver todos eles. Outro cenário em que ele pode ser utilizado é quando vários campos serão colocados na tela, e o desenvolvedor quer garantir que mesmo se a tela do celular do usuário for muito pequena, ele ainda conseguirá efetuar um scroll para utilizar todos os elementos. As imagens abaixo ilustram a utilização desse widget.

Figura 108 – ListView.

```
ListView(
  padding: const EdgeInsets.all(8),
  children: <Widget>[
    Container(
      height: 50,
      color: Colors.amber[600],
      child: const Center(child: Text('Entry A')),
    ),
    Container(
      height: 50,
      color: Colors.amber[500],
      child: const Center(child: Text('Entry B')),
    ),
    Container(
      height: 50,
      color: Colors.amber[100],
      child: const Center(child: Text('Entry C')),
    ),
  ],
)
```

Fonte: https://flutter.dev/.

Figura 119 – ListView.



Fonte: https://flutter.dev/.

Referências

FLUTTER. Disponível em: <https://flutter.dev/>. Acesso em: 19 out. 2020.

DART. Disponível em: <https://dart.dev/>. Acesso em: 19 out. 2020.