Navegação de Telas

Computação para Dispositivos Móveis

Prof. Gustavo Custodio
gustavo.custodio@anhembi.br

Conteúdo

- O que faremos essa aula?
  - Adicionar estado para a aplicação.
  - Interagir com botões.
  - Navegar para a próxima tela.

Navegacao de Telas

StatefulWidget

StatefulWidget

- Até o momento só utilizamos o `StatelessWidget`, widgets estáticos, para criar telas de apps.
  - No entanto, a maioria dos aplicativos possui alguma forma de estado.
- Existem também o `StatefulWidget`, um widget que **pode armazenar informação** e pode ser recriado quando um estado muda.
  - Comparado com widgets *stateless*, os widgets *stateful* possuem partes mais complexas.

StatefulWidget

- Vamos criar um aplicativo cronômetro.

StatefulWidget

- Adicione o código no `main.dart`.
- Esse código é utilizado para criar a tela principal do aplicativo, mas o conteúdo principal ficará em um `StatefulWidget`.

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

void main() {
  runApp(AppCronometro());
}

class AppCronometro extends StatelessWidget {
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Cronometro(),
    );
  }
}
```

StatefulWidget

- Crie o arquivo `cronometro.dart` na pasta `lib` e adicione o código:

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

class Cronometro extends StatefulWidget {
  @override
  State<Cronometro> createState() {
    return _CronometroState();
  }
}
```

StatefulWidget

- Um `StatefulWidget` é divido em duas classes: o widget e o estado (`State`).
- Todo `StatefulWidget` possui um método `createState()` que retorna um `State`.
  - Os widgets `StatefulWidget` e `State` são tão dependentes um do outro que são colocados no mesmo arquivo.

State

- Vamos criar a classe privada `_CronometroState` logo abaixo de `Cronometro`.

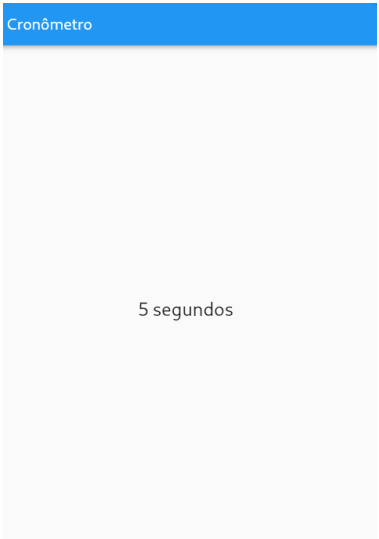
```dart
class _CronometroState extends State<Cronometro> {
  int segundos = 0;
  Timer? timer;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Cronometro'),
      ),
      body: Center(
        child: Text(
          '$segundos segundos',
          style: Theme.of(context).textTheme.headline5,
        ),
      ),
    );
  }
}
```

State

- Ao invés de digitar o código para criar um `StatefulWidget`, existem *snippets* que podem ser utilizados no VSCode.
  - `stful` - Cria a base de uma classe que herda de `StatefulWidget`.
  - `stless` - Cria a base de uma classe que herda de `StatelessWidget`.

State



- O `Timer` é o widget responsável por controlar o contador de tempo.
- A cada segundo ele executa uma função passada como parâmetro.

State

- Agora precisamos adicionar o comportamento do cronômetro.
  - Primeiro definimos uma função `initState`, responsável por definir o estado inicial do cronômetro.
  - Depois definimos uma função `_onTick(Timer time)`.
    - Dentro dessa função, temos um `setState`, responsável por **atualizar** a tela.
  - Por último, temos um método `dispose`, responsável por parar o contador quando o aplicativo é fechado.

State

```dart
@override
void initState() {
  super.initState();
  segundos = 0;
  // A cada segundo, o método _onTick échamado
  timer = Timer.periodic(Duration(seconds: 1), _onTick);
}

void _onTick(Timer time) {
  setState(() {
    ++segundos;
  });
}

@override
void dispose() {
  timer?.cancel();
  super.dispose();
}
```

State

- Cada vez que o `setState` é chamado, o widget é gerado novamente.
  - Widget chama o `setState()`;
  - Widget fica marcado para ser gerado novamente.
  - Flutter chama os métodos `build` desses widgets.

Navegação

- Vamos criar uma tela inicial onde você pode clicar em um botão e ser direcionado para a tela do cronômetro.
  - Mas antes de realizar a navegação entre telas, precisamos aprender a criar botões.
- Vamos criar um botão para resetar o cronômetro.
  - Fazemos isso utilizando o `ElevatedButton`.

Navegacao de Telas

Navegação

Navegação

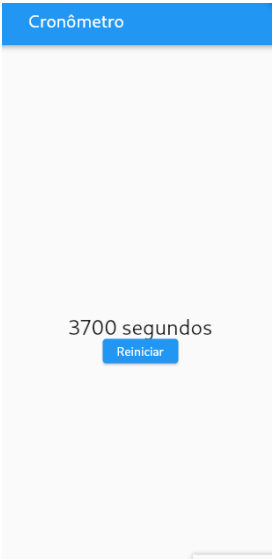
- Antes de criar o botao, vamos adicionar um widget `Column` em `_CronometroState`
  - Dessa forma, podemos adicionar o contador e um botão na mesma tela.

```
body: Column(
  // Alinha os elementos da coluna no centro da tela
  mainAxisAlignment: MainAxisAlignment.center,
  children: <Widget>[
    Center(
      child: Text(
        '$segundos segundos',
        style: Theme.of(context).textTheme.headline5,
      ),
    ),
    Center(
      child: _criarBotaoDeResetar(context),
    ),
  ],
),
```

Navegação

```
1  Widget _criarBotaoDeResetar(BuildContext context) {
2    return ElevatedButton(
3      onPressed: (){
4        setState(() {
5          segundos = 0;
6        });
7      },
8      child: Text('Resetar'),
9    );
10 }
```

Navegação



Navegação

- O atributo `onPressed` determina o que acontece quando o botão é pressionado.
  - No nosso caso, ele chama o método `setState` que zera o contador de segundos.
  - O `onPressed` está associado a uma função que é passada como parâmetro.
    - Quando o botão é pressionado, ele executa a função a qual ele está associado.

Navegação

- O Flutter utiliza uma classe chamada `Navigator` para gerenciar as telas do aplicativo.
  - O `Navigator` mantém um histórico de todas as telas chamadas e a ordem delas.

Navegação

- Dentro do arquivo `main.dart` crie uma classe chamada TelaPrincipal.

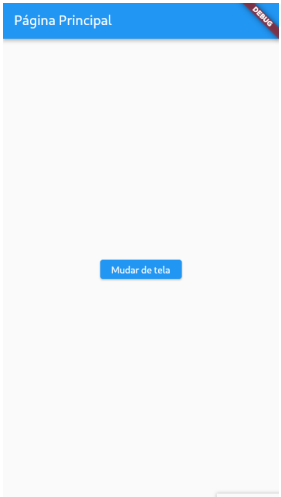
```dart
class TelaPrincipal extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Página Principal'),
      ),
      body: Center(
        child: ElevatedButton(
          child: Text('Mudar de tela'),
          onPressed: (){
            Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => Cronometro()),
            );
          },
        ),
      ),
    );
  }
}
```

Navegação

- E altere a classe AppCronometro:

```
class AppCronometro extends StatelessWidget {
  Widget build(BuildContext context) {
    return MaterialApp(
      home: TelaPrincipal(),
    );
  }
}
```

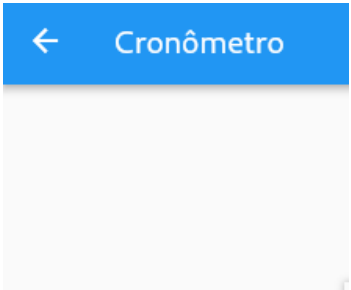
Navegação



Navegação

- `Navigators` funcionam como uma pilha:
  - Você pode adicionar componentes na pilha com push.
  - Você pode remover componentes da pilha com pop.
- Utilizamos o push para adicionar a tela com o cronômetro na lista de telas.
  - Agora vamos utilizar o pop para criar um botão de voltar.

Navegação



Navegação

- Vamos criar um método `_criarBotaoDeVoltar`.

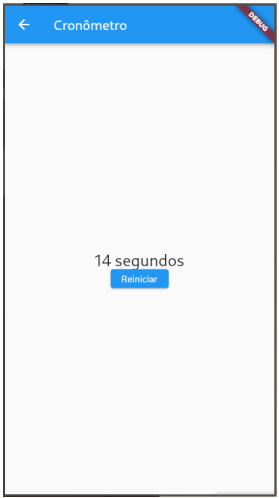
```
1  Widget _criarBotaoDeVoltar(BuildContext context) {
2    return IconButton(
3      icon: Icon(Icons.arrow_back),
4      onPressed: (){
5        Navigator.pop(context);
6      },
7    );
8  }
```

Navegação

- E vamos adicionar um botão no AppBar na tela do cronômetro:

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Cronometro'),
      leading: _criarBotaoDeVoltar(context),
    ),
  ...
```

Navegação



Navegação

- Além do `ElevatedButton`, existe também o `IconButton`.
  - Ele é uma imagem que reage a toques do usuário.
  - Além do botão de voltar, é possível criar botões de edição de texto, de salvar, etc.
- Usando o conhecimento que temos de navegação agora, podemos criar um menu de navegação.

Navegacao de Telas

Criando um Menu

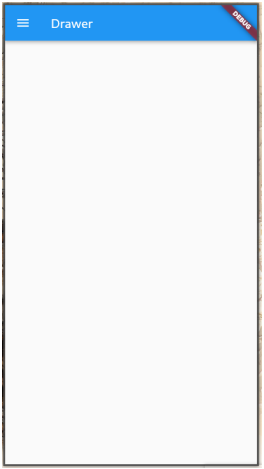
Drawer

- O Flutter conta com um recurso chamado `Drawer` (traduzido como gaveta).
  - Basicamente é o recurso que estende um menu do lado esquerdo da tela.
  - Ele é adicionado por meio da propriedade `drawer` no `Scaffold`.

Drawer

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

class AppDrawer extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      drawer: Drawer(),
      appBar: AppBar(
        title: Text('Drawer'),
      ),
    );
  }
}
```

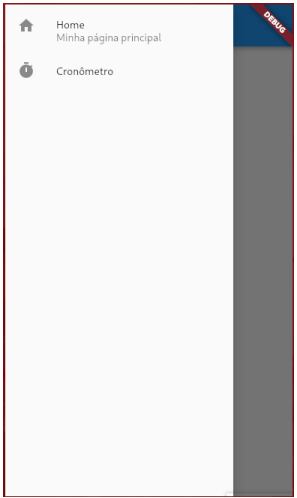


Drawer

- O `Drawer` mostrado está em branco, então precisamos adicionar conteúdo nele.
  - Um menu é um conteúdo vertical com vários itens.
  - Dessa forma, podemos inserir o conteúdo dentro do menu usando um `Column`.
    - Ou melhor, vamos utilizar um widget rolável chamado `ListView`.

Drawer

- Vamos tentar reproduzir o seguinte aplicativo usando os widgets `ListView` e `ListTile`.
- O `ListView` agrupa um conjunto de elementos.
  - O `ListTile` representa um único elemento.



Drawer

- Acrescente esse código no começo do `LisrView` para um menu visualmente mais agradável.

```
children: <Widget>[
  UserAccountsDrawerHeader(
      accountName: Text('nome'),
      accountEmail: Text('email')),
  ListTile(
    leading: Icon(Icons.home),
```

Drawer

- O código representa a função que cria o menu no `Drawer`.

```
Widget _criarMenuDrawer(BuildContext context) {
  return ListView(
    children: <Widget>[
      ListTile(
        leading: Icon(Icons.home),
        title: Text('Home'),
        subtitle: Text('Minha página principal'),
      ),
      ListTile(
        leading: Icon(Icons.timer),
        title: Text('Cronometro'),
      ),
    ],
  );
}
```

Drawer

- Agora chamamos o método `_criarMenuDrawer` na função `build` como filho do `Drawer`:

```
1  Widget build(BuildContext context) {
2    return Scaffold(
3      drawer: Drawer(
4        child: _criarMenuDrawer(context),
5      ),
6      appBar: AppBar(
7        title: Text('Drawer'),
8      ),
9    );
10 }
```

Drawer

- Para adicionar um comportamento ao `ListTile`, utilize o atributo onTap.
  - Ele funciona da mesma forma que o `onPressed` para botões.

Drawer

```
ListTile(
  leading: Icon(Icons.timer),
  title: Text('Cronometro'),
  onTap: (){
    Navigator.push(
      context,
      // Clicando nesse elemento direciona para outra página
      MaterialPageRoute(builder: (context) => Cronometro()),
    );
  },
),
```

Rotas

- Em Flutter, podemos utilizar **rotas nomeadas** para navegar pelo nosso aplicarivo.
  - Isso facilita a navegação entre diferentes telas.
  - Temos um caminho diferente associado a cada tela.

Rotas

- Dentro do arquivo `main.dart`, adicionamos ao `MaterialApp` o atributo `routes`:

```dart
class MyApp extends StatelessWidget {
 const MyApp({Key? key}) : super(key: key);

 @override
 Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Flutter Demo',
    home: AppDrawer(),
    initialRoute: '/',
    routes: {
      '/cronometro': (context) => Cronometro(),
    },
  );
 }
}
```

Rotas

- Dessa forma, a rota “/cronometro” será o caminho para chegar no nosso cronômetro.
- Para navegar para a rota específica, ainda utilizamos o `Navigator`.
- Seguimos a sintaxe:
  - `Navigator.of(context).pushNamed('/rota')`

Rotas

- Vamos modificar o exemplo anterior:

```
1       onTap: (){
2         Navigator.of(context).pushNamed('/cronometro');
3       },
```

Exercício

- Crie um aplicativo com três rotas:
  - Home (uma página escrita Bem-vindo).
  - Cronômetro.
  - Galeria (mostra um conjunto de fotos).
- Faça com que o menu apareça em toda as rotas.

Referências

Simone Alessandria and Brian Kayfitz.
Flutter Cookbook: Over 100 proven techniques and solutions for app development with Flutter 2.2 and Dart.
Packt Publishing Ltd, 2021.

Conteúdo

https://gustavotcustodio.github.io/dispomoveis.html

Obrigado

gustavo.custodio@anhembi.br