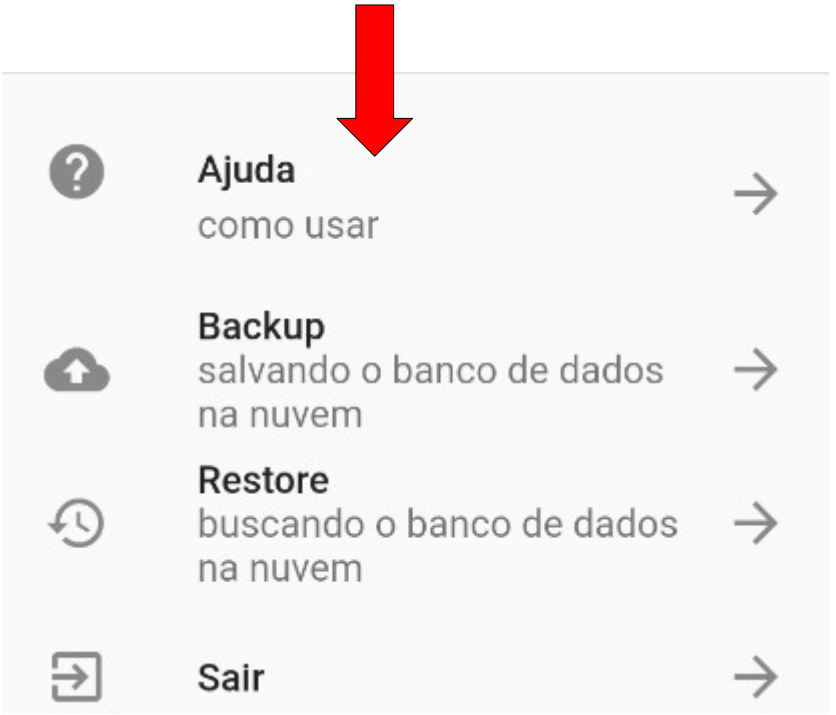
Vídeo e calendário

Usando vídeos em um app

- No App FIIs Plan utilizamos um vídeo para explicar o funcionamento do mesmo. Isso pode ser feito acessando o menu lateral na opção **Ajuda**.

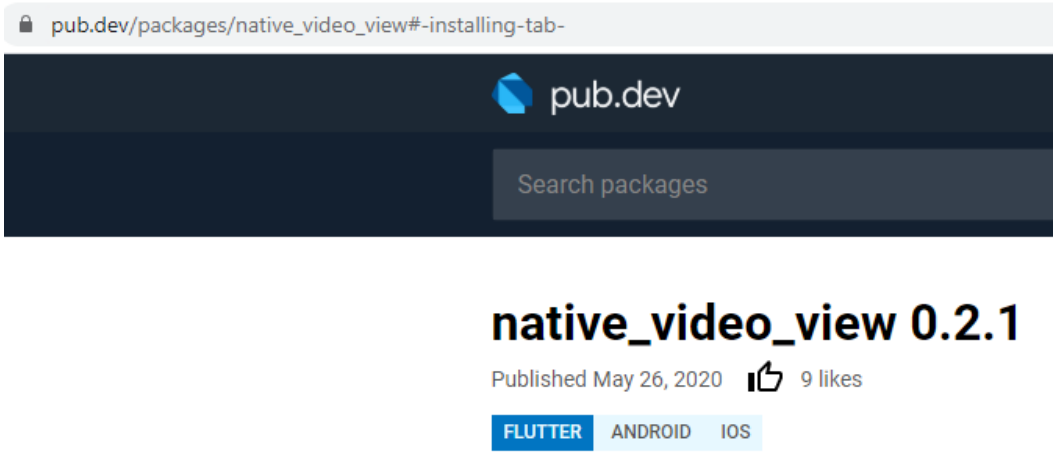
```
ListTile _ajuda(BuildContext context) {
  return ListTile(
    leading: Icon(Icons.help),
    title: Text("Ajuda"),
    subtitle: Text("como usar"),
    trailing: Icon(Icons.arrow_forward),
    onTap: () {
      // Fechando o menu lateral
      pop(context);

      push(context, TelaAjuda());
    },
  ); // ListTile
}
```



TelaAjuda

- A TelaAjuda basicamente exibe o vídeo de explicação do uso do App FIIs Plan.
- Para tanto ela utiliza o plugin **native_video_view** cujas instruções de instalação se encontram em:

  https://pub.dev/packages/native_video_view#-installing-tab-

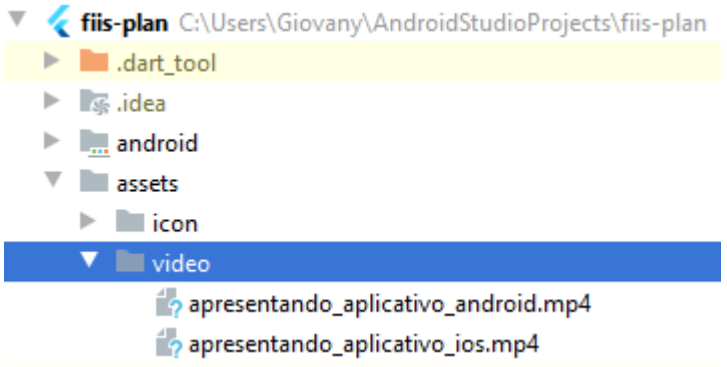


Configurações Android

- As configurações complementares do uso do plugin **native_video_view** se encontram em:

  https://pub.dev/packages/native_video_view#-readme-tab-

- Como os vídeos que iremos utilizar estão na pasta assets, precisaremos adicionar a seguinte permissão no AndroidManifest.xml: `<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />`

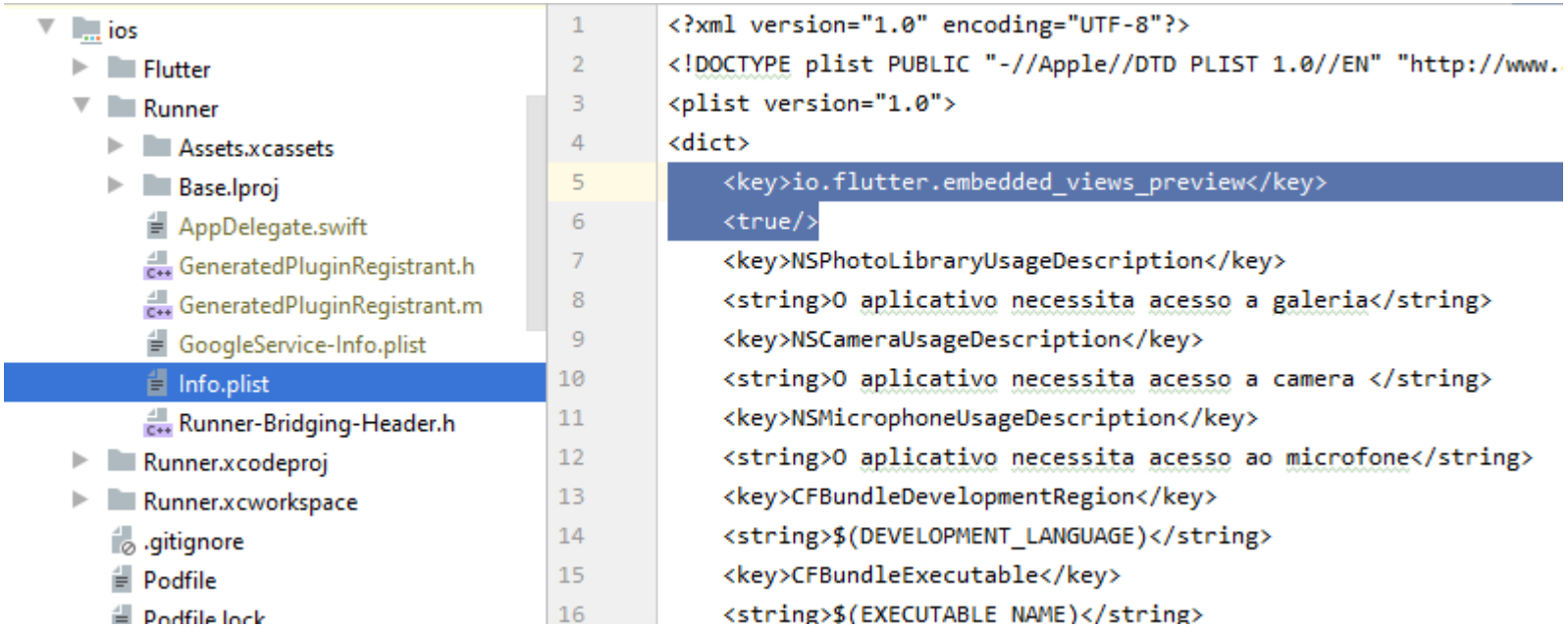



```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.fundos.fundosimobiliarios">
    <!-- io.flutter.app.FlutterApplication is an android.app.Application that
         calls FlutterMain.startInitialization(this); in its onCreate method.
         In most cases you can leave this as-is, but you if you want to provide
         additional functionality it is fine to subclass or reimplement
         FlutterApplication and put your custom class here. -->
    <application
        android:name="io.flutter.app.FlutterApplication"
        android:label="FIIs Plan"
        android:icon="@mipmap/launcher_icon">
...
    </application>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
</manifest>
```

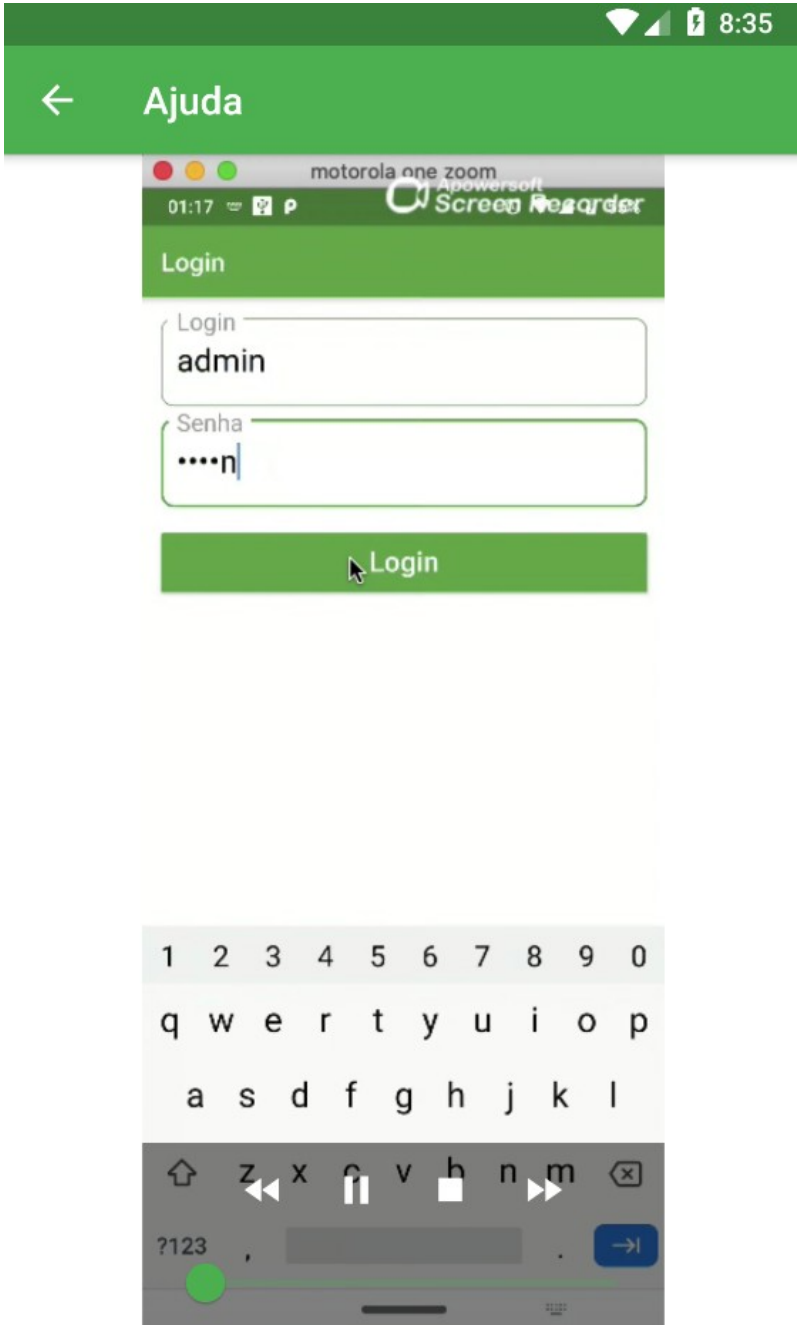
Configurações iOS

- Precisaremos ir no arquivo **Info.plist** e adicionar as seguintes linhas:

```
<key>io.flutter.embedded_views_preview</key>
<true/>
```



TelaAjuda



- A TelaAjuda exibe o vídeo explicando o uso do App FIIs Plan (imagem ao lado).
- Para a confecção dos vídeos foi utilizado o aplicativo Apowersoft Screen Recorder em versão não paga (que mantém a logo no vídeo gerado).
- TelaAjuda é um **StatelessWidget** e no próximo slide veremos o método **build** que desenha a tela.
- Para utilizar o plugin native_video_view foi necessário fazer a seguinte importação:

```
import 'package:native_video_view/native_video_view.dart';
```

TelaAjuda

```dart
class TelaAjuda extends StatelessWidget {

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Ajuda'),
      ), // AppBar
      body: Container(
        alignment: Alignment.center,
        child: NativeVideoView(
```

- Basicamente temos um Scaffold com uma AppBar simples e um Container centralizado no **body**.
- No próximo slide veremos o widget **NativeVideoView** que efetivamente irá exibir nossos vídeos.

TelaAjuda

- As explicações sobre cada um dos atributos de **NativeVideoView** se encontram em (final da página):

  https://pub.dev/packages/native_video_view#-readme-tab-

- Os métodos mais importantes são o **onCreated** e o **onPrepared**.
- Em **onCreated**, de acordo com a plataforma (Android ou iOS), um vídeo é carregado da pasta **assets**. Essa diferenciação é importante pois o aplicativo iOS não tem os menus de Backup e Restore.
- **onPrepared** é chamado quando o vídeo foi carregado, nesse contexto, basta então acionar o **play** para rodá-lo.

```
child: NativeVideoView(
  keepAspectRatio: true,
  showMediaController: true,
  onCreated: (controller) {
    if(Platform.isAndroid){
      controller.setVideoSource(
        'assets/video/apresentando_aplicativo_android.mp4',
        sourceType: VideoSourceType.asset,
      );
    } else {
      controller.setVideoSource(
        'assets/video/apresentando_aplicativo_ios.mp4',
        sourceType: VideoSourceType.asset,
      );
    }
  },
  onPrepared: (controller, info) {
    controller.play();
  },
  onError: (controller, what, extra, message) {
    print('Player Error ($what | $extra | $message)');
  },
  onCompletion: (controller) {
    print('Video completed');
  },
), // NativeVideoView
```


Flutter 2.0

```dart
class _TelaAjudaState extends State<TelaAjuda> {
  late VideoPlayerController _videoPlayerController;
  bool startedPlaying = false;

  @override
  void initState() {
    super.initState();
    String string_video;

    if(Platform.isAndroid){
      string_video = 'assets/video/apresentando_aplicativo_android.mp4';
    } else {
      string_video = 'assets/video/apresentando_aplicativo_ios.mp4';
    }

    _videoPlayerController =
        VideoPlayerController.asset(string_video);
    _videoPlayerController.addListener(() {
      if (startedPlaying && !_videoPlayerController.value.isPlaying) {
        Navigator.pop(context);
      }
    });
  }
```

- Basicamente foi necessário trocar o plugin de uso de vídeos. O native_video_view não foi atualizado para o Flutter 2.0 é ainda não possui suporte a null-safe.
- Em seu lugar utilizamos o video_player. Para instalar esse plugin basta ir em https://pub.dev/packages/video_player/install e seguir as instruções.
- No initState de acordo com a plataforma selecionamos o caminho do vídeo.
- Na sequência inicializamos o _videoPlayerController com as configurações necessárias.

Flutter 2.0

- Aqui liberamos o **_videoPlayerController** no **dispose** e definimos o método que irá inicializar o vídeo no FutureBuilder definido (próximo slide):

```
@override
void dispose() {
  _videoPlayerController.dispose();
  super.dispose();
}

Future<bool> started() async {
  await _videoPlayerController.initialize();
  await _videoPlayerController.play();
  startedPlaying = true;
  return true;
}
```


Flutter 2.0

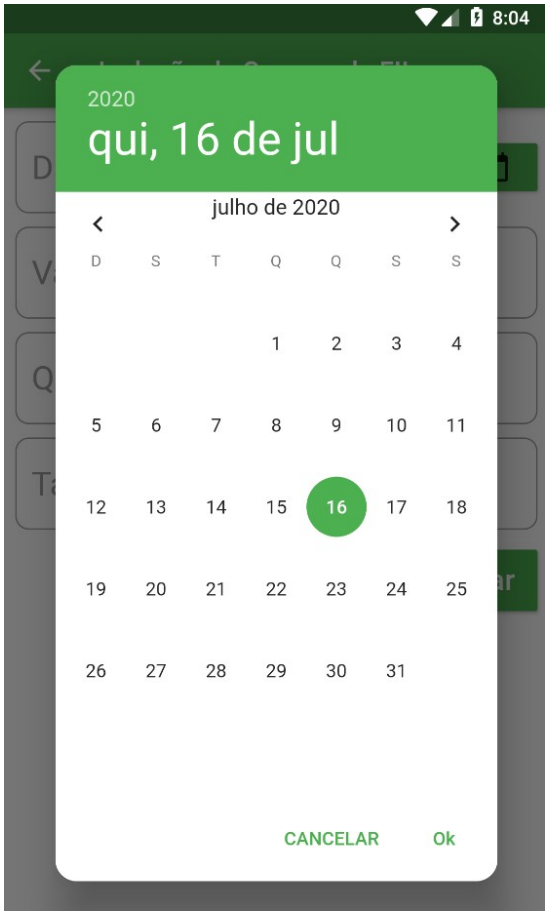
```dart
@override
Widget build(BuildContext context) {
  return Material(
    elevation: 0,
    child: Center(
      child: FutureBuilder<bool>(
        future: started(),
        builder: (BuildContext context, AsyncSnapshot<bool> snapshot) {
          if (snapshot.data == true) {
            return AspectRatio(
              aspectRatio: _videoPlayerController.value.aspectRatio,
              child: VideoPlayer(_videoPlayerController),
            );  // AspectRatio
          } else {
            return const Text('Aguarde o vídeo carregar');
          }
        },
      ),  // FutureBuilder
    ),  // Center
  );  // Material
}
```

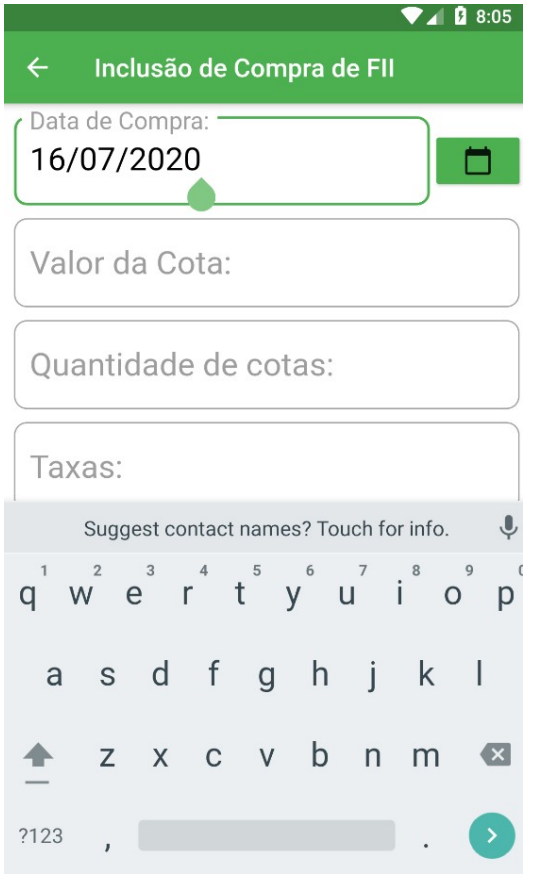
- Assim que started() terminar (slide anterior) o **FutureBuilder** poderá apresentar o VideoPlayer.
- Se **snapshot.data == true** o VideoPlayer é carregado utilizado o **_videoPlayerController** definido anteriormente.
- Enquanto o vídeo não é carregado uma mensagem de texto é exibida.

Calendário

```
CampoEdicaoData(
  "Data de Compra:",
  controlador: _controle.controlador_data_compra,
  recebedor_foco: _controle.focus_valor_cota,
), // CampoEdicaoData
```

- Para trabalhar com datas foi criado um widget chamado **CampoEdicaoData**.
- Esse widget foi utilizado em diversos lugares. Ao lado vemos o código da **TelaEdicaoCompra**.
- **controlador_data_compra** é o **TextEditController** que guardará o valor da data selecionada ou escrita no campo de edição.
- **focus_valor_cota** é o **FocusNode** que irá indicar para onde irá o cursor se o usuário clicar para avançar.





Plugin flutter_rounded_date_picker

- Para trabalhar com calendários utilizaremos o plugin **flutter_rounded_date_picker** que pode ser obtido em:

  https://pub.dev/packages/flutter_rounded_date_picker/install

- Uma configuração importante do plugin é a questão da internacionalização (suporte a várias línguas). Essa configuração permite que o plugin utilize a língua utilizada no smartphone.
- Também é possível customizar bastante o design do calendário. Para maiores detalhes:

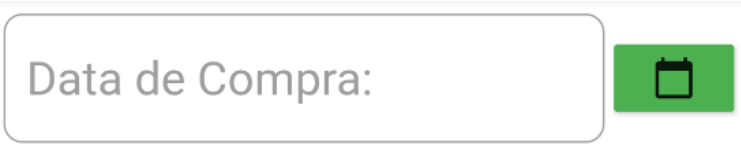
  https://pub.dev/packages/flutter_rounded_date_picker#-readme-tab-

Suporte a várias línguas

- Ao lado vemos o arquivo **main.dart**.
- Em **locale** podemos colocar o local a ser utilizado pelo App. Deixar essa opção como feito ao lado pode prejudicar a internacionalização (um smartphone em inglês iria abrir o calendário em português).
- **supportedLocales** são os locais/línguas suportados pelo aplicativo.

```dart
import 'package:flutter/material.dart';
import 'package:fundosimobiliarios/telas/tela_abertura.dart';
import 'package:flutter_localizations/flutter_localizations.dart';

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      localizationsDelegates: [
        GlobalMaterialLocalizations.delegate,
        GlobalWidgetsLocalizations.delegate,
        GlobalCupertinoLocalizations.delegate,
      ],
      // Força português independente o idioma do smartphone
      // Deixei essa linha apenas para mostrar, num aplicativo a ser
      // disponibilizado no AppStore, ela deve ser retirada
      locale: Locale('pt', 'BR'),  // Current locale
      supportedLocales: [
        const Locale('en', 'US'), // English
        const Locale('pt', 'BR'), // Brasil
      ],
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        primarySwatch: Colors.green,
        // Define o tema para claro ou escuro
        brightness: Brightness.light,
        // Define a cor de fundo padrão para Containers
        scaffoldBackgroundColor: Colors.white,
      ), // ThemeData
      home: TelaAbertura(),
    ); // MaterialApp
  }
}
```


CampoEdicaoData




```
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:fundosimobiliarios/util/formatacao.dart';
import 'package:fundosimobiliarios/util/widgets/botao_icone.dart';
import 'package:flutter_rounded_date_picker/rounded_picker.dart';


class CampoEdicaoData extends StatefulWidget {
  String texto_label;
  String texto_dica;
  bool  passaword;
  TextEditingController controlador;
  FormFieldValidator<String> validador;
  TextInputType teclado;
  FocusNode marcador_foco;
  FocusNode recebedor_foco;
```

- A importação do arquivo **formatacao.dart** é importante para gerar o formato de data.
- A importação de **botao_icone.dart** é necessária pois o utilizamos na montagem desse widget (ele usa o widget BotaoIconce).
- Os atributos que aparecem são normalmente os mesmos utilizados em outros campos de edição.

```
class CampoEdicao extends StatelessWidget {
  String texto_label;
  String texto_dica;
  bool  passaword;
  TextEditingController controlador;
  FormFieldValidator<String> validador;
  TextInputType teclado;
  FocusNode marcador_foco;
  FocusNode recebedor_foco;
```

CampoEdicaoData

```
CampoEdicaoData(
    this.texto_label,
    {this.texto_dica = "",
      this.passaword = false,
      this.controlador = null,
      this.teclado = TextInputType.text,
      this.marcador_foco = null,
      this.recebedor_foco = null}){
    this.validador = (String text){
      if(text.isEmpty)
        return "O campo '$texto_label' está vazio e necessita ser preenchido";
      if(gerarDateTime(text) == null) {
          return "Formato de data inválido (deve ser dd/mm/aaaa)";
      }
      return null;
    };
  }
```

- No construtor o destaque é a questão do **validador**. Note que esse campo de edição não permitirá valor vazio e verificará o formato da data através de **gerarDatetime**.
- Esse comportamento é interessante visto que nos poupa tempo de implementá-lo nos diversos usos desse widget.

- Utilizamos uma **Row** para dispor 2 widgets, um **TextFormField** e um **BotaoIcone** (próximo slide).
- A proporção desses elementos na **Row** foi feita usando **Expanded** e o atributo **flex** (5 para o TextFormField e 1 para o BotaoIcone).
- Ao lado vemos diversas configurações do **TextFormField** para termos o design apresentado.

```
class _CampoEdicaoDataState extends State<CampoEdicaoData> {
  @override
  Widget build(BuildContext context) {
    return Row(
      mainAxisAlignment: MainAxisAlignment.spaceBetween,
      children: <Widget>[
        Expanded(
          flex: 5,
          child: TextFormField(
            validator: widget.validador,
            obscureText: widget.passaword,
            controller: widget.controlador,
            keyboardType: widget.teclado,
            textInputAction: TextInputAction.next,
            focusNode: widget.marcador_foco,
            onFieldSubmitted:(String text){
              FocusScope.of(context).requestFocus(widget.recebedor_foco);
            },
            // Estilo da fonte
            style: TextStyle(
              fontSize: 25,
              color: Colors.black,
            ), // TextStyle
            decoration: InputDecoration(
              border: OutlineInputBorder(
                borderRadius: BorderRadius.circular(10),
              ), // OutlineInputBorder
              labelText: widget.texto_label,
              // Estilo de labelText
              labelStyle: TextStyle(
                fontSize: 25,
                color: Colors.grey,
              ), // TextStyle
              hintText: widget.texto_dica,
              // Estilo do hintText
              hintStyle: TextStyle(
                fontSize: 10,
                color: Colors.green,
              ), // TextStyle
            ), // InputDecoration
          ), // TextFormField
        ), // Expanded
```

- Antes do **BotaoIcone** temos um **SizedBox** para dar um pequeno espaçamento entre os widgets.
- Aqui ocorre o chamamento a funcionalidade do plugin quando o botão é clicado.
- Assim que datatime é obtido ele é transformado no formato de String e setado no controlador do **TextFormField**. Como ocorre a chamada do setState, o widget é redesenhado com o valor informado.

```
SizedBox(
  width: 5,
), // SizedBox
Expanded(
  flex: 1,
  child: BotaoIcone(
    ao_clicar: () async {
      DateTime datatime = await showRoundedDatePicker(
          context: context,
          theme: ThemeData(
            primarySwatch: Colors.green
          ), // ThemeData
      );
      setState(() {
        widget.controlador.text = formatarDateTime(datatime);
      });
    },
    cor: Colors.green,
    icone: Icons.calendar_today,
  ), // BotaoIcone
), // Expanded
], // <Widget>[]
); // Row
```

Dúvidas?