SI700 – Programação para Dispositivos Móveis

Aula 4 – Widgets para Entrada de Dados do Usuário

Prof. Ulisses Martins Dias

2022

Faculdade de Tecnologia – Unicamp

Roteiro

Form

TextFormField

ElevatedButton

Checkbox

Switch

Slider

Radio

SnackBar

BottomSheet

DefaultTabController, TabBarView, TabBar

- Usados em conjunto com a **appBar** e o **body** do **Scaffold**, esses elementos implementam abas que ficam na parte superior da janela principal. A cada momento, apenas uma aba pode estar visível.
- O **DefaultTabController** deverá ser ser filho de **MaterialApp** no parâmetro **home** e pai de **Scaffold**.
- A função desse elemento é manter a sincronia das abas, sendo que nele você irá informar o número de abas no parâmetro **length**.

```
return MaterialApp(
  home: DefaultTabController(
    length: 3,
    child: Scaffold(),
  ),
);
```

- A criação das abas fica a cargo do widget **TabBar**. Em geral, você irá criar as abas e adicionar na **AppBar** que estára no parâmetro **appBar** do **Scaffold**.

```
appBar: AppBar(
  bottom: const TabBar(
    tabs: [
      Tab(icon: Icon(Icons.directions_car)),
      Tab(icon: Icon(Icons.directions_transit)),
      Tab(icon: Icon(Icons.directions_bike)),
    ],
  ),
)
```

DefaultTabController, TabBarView, TabBar

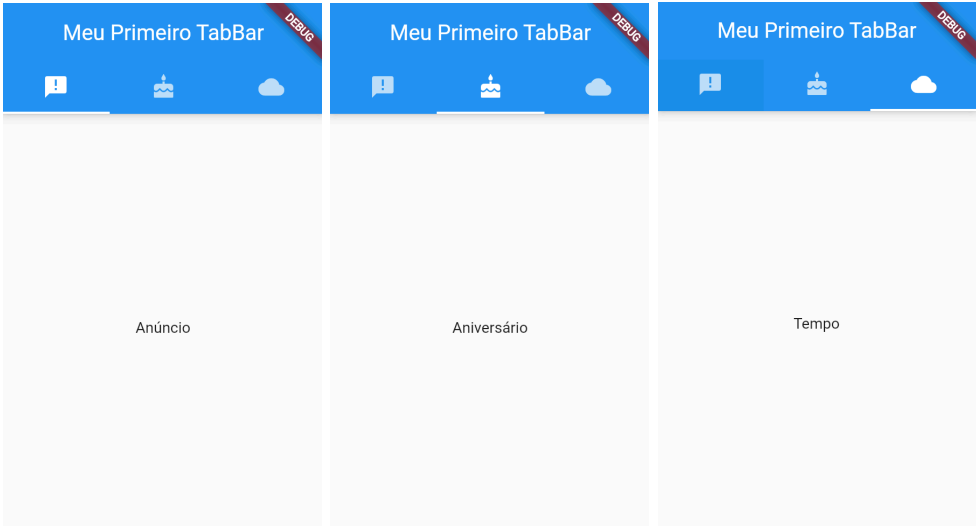
- Uma **TabBarView** é uma pilha de telas (ou views) em que uma está visível a cada momento.
- A maneira de tornar uma tela visível é interagindo com a **TabBar** criada na **AppBar**.

```
body: const TabBarView(
  children: [
    Icon(Icons.directions_car),
    Icon(Icons.directions_transit),
    Icon(Icons.directions_bike),
  ],
)
```

DefaultTabController, TabBarView, TabBar

```
MaterialApp(
 home: DefaultTabController(
  length: 3,
  child: Scaffold(
   body: const TabBarView(children: [
    Center(child: Text("Anúncio")),
    Center(child: Text("Aniversário")),
    Center(child: Text("Tempo")) ]),
   appBar: AppBar(
    title: const Text("Meu Primeiro TabBar"),
    bottom: const TabBar(tabs: [
     Tab(icon: Icon(Icons.announcement)),
     Tab(icon: Icon(Icons.cake)),
     Tab(icon: Icon(Icons.cloud)) ])),  ), ),);
```

DefaultTabController, TabBarView, TabBar



Form

Form

- **Form** é um widget opcional, mas que possui algumas utilidades importantes. Todos os campos de entrada dos dados serão filhos de um mesmo **Form**.
- A razão para usar **Form** é que permite salvar dados do formulário, resetar e validar o conteúdo.
- A propriedade **key** dos widgets permite associar uma chave global (**GlobalKey**) para identificar um widget, permitindo que a validação seja invocada em um passo posterior.

Form

```
// Como Atributo
final GlobalKey<FormState> _formKey = GlobalKey<FormState>();

Form(
  key: _formKey,
  child: Column(
    children: <Widget>[/*Widgets de entrada*/]
  )
);

// Posteriormente
if (formKey.currentState!.validate()) {
  formKey.currentState!.save();
}
```

TextFormField

TextFormField

Um **TextFormField** irá criar uma tela onde o usuário poderá inserir informações. Alguns atributos são importantes:

- **keyboardType**: permite dizer o que pretendemos receber do usuário e o sistema colocará um teclado com teclas apropriadas. Por exemplo, datas, urls, ...
- **validator**: permite adicionar uma função para validar a entrada do usuário. Se a entrada for inválida, o validator retornará uma string contendo a mensagem de erro. Caso contrário, o validator retornará **null**.
- **onSave**: permite criar uma função que informa como os dados serão salvos. Após essa função, podemos disparar uma ação que fará uso desses dados.

TextFormField

```
TextFormField(
  keyboardType: TextInputType.emailAddress,
  validator: (String? inValue) {
    if (inValue != null) {
      if (inValue.isEmpty) {
        return "Insira um nome de usuário";
      }    }
    return null;
  },
  onSaved: (String? inValue) {
    loginData.username = inValue ?? "";  },
  decoration: const InputDecoration(
    hintText: "user@domain.br",
    labelText: "Username (E-mail Address)",
  ) );
```

- Como segundo exemplo, a seguir criaremos um **TextFormField** para guardar a senha do usuário.
- Note o parâmetro **obscureText** sendo usado para colocar asterisco no lugar das teclas digitadas pelo usuário.
- O **validator** agora proibe senhas muito curtas.

TextFormField

```
TextFormField(
  obscureText: true,
  validator: (String? inValue) {
    if (inValue != null) {
      if (inValue.length < 10) {
        return "Mínimo de 10 letras";
      }    }
    return null;
  },
  onSaved: (String? inValue) {
    loginData.password = inValue ?? "";  },
  decoration: const InputDecoration(
    labelText: "Insira uma senha forte",
  )  );
```

ElevatedButton

ElevatedButton

- Um formulário não estará completo sem um botão, sendo esse o papel da classe **ElevatedButton**.
- Em geral, quando o usuário pressionar o botão, usaremos a referência que temos do formulário (a **GlobalKey** que mencionamos anteriormente na class **Form**).
- A função do botão será desencadear uma chamada para os métodos adicionados nos parâmetros **validator** e **onSave** de cada **TextFormField**, respectivamente.

ElevatedButton

```
ElevatedButton(
  child: const Text("Log In!"),
  onPressed: () {
    if (formKey.currentState!.validate()) {
      formKey.currentState!.save();
      // Faça algo com os dados
    }
  },
);
```

Checkbox

Checkbox

Vamos começar com uma classe que irá consolidar todos os valores do formulário.

```
class CompleteForm {
  bool checkboxValue = false;
  bool switchValue = false;
  double sliderValue = .5;
  int radioValue = 1;
  int radioValue2 = 1;
  int bottomSheetChoice = 0;
  doSomething() {
    print("CheckBox: $checkboxValue");
    print("Switch: $switchValue");
    print("Slider: $sliderValue");
    print("Radio: $radioValue");
  }
}
```

Checkbox

Feito isso, vamos assumir que temos um objeto completeForm em algum lugar visível do código. A seguir, um código de referência. Note o uso dos parâmetros **value** e **onChanged**.

```
Checkbox(
  value: completeForm.checkboxValue,
  onChanged: (bool? inValue) {
    if (inValue != null) {
      setState(() {
          completeForm.checkboxValue = inValue;
      });
    }
  },
);
```

Checkbox

Note que um **Checkbox** não possui um rótulo de texto, o que seria comum nesse tipo de componente. Nesse caso, você teria que criar um você mesmo colocando o **Checkbox** e um widget **Text** dentro de um **Row**.

Switch

Switch

- Um **Switch**, assim como o **CupertinoSwitch**, é um **Checkbox** com um visual diferente. Ele se parece com um botão de ligar e desligar que vemos em aparelhos eletrônicos.
- Um ponto a se notar é que se **onChange** for nulo, então o **Switch** aparecerá desabilitado na tela e não receberá interação do usuário.
- É muito fácil de entender os elementos do **Switch** associando com o que já vimos com o **Checkbox**

Switch

```
Switch(
  value: completeForm.switchValue,
  onChanged: (bool inValue) {
    setState(() {
        completeForm.switchValue = inValue;
    });
  },
);
```

Slider

Slider

- **Slider** mostra uma linha e uma espécie de alavanca (círculo no centro da linha) para que você possa mover para alguma posição. Isso permitirá definir um valor dentro de um intervalo.
- As propriedades **min** e **max** são usadas para gerenciar o intervalo, e a propriedade **value** define o valor atual.
- **onChange** é necessário para modificar o valor quando a maçaneta é movida. Outras propriedades podem ser úteis em casos específicos, como **onChangeStart** e **onChangeEnd** que permitem adicionar comportamento quando o usuário começa a mover o **Slider** e quando termina.

Slider

```
Slider(
  min: 0,
  max: 20,
  value: completeForm.sliderValue,
  onChanged: (double inValue) {
    setState(() => completeForm.sliderValue = inValue);
  },
);
```

Radio

Radio

- Uma barra de botões circulares semelhante aos rádios de carros antigos. Quando um botão é pressionado, todos os outros pulam de volta.
- Um **Radio** é muito semelhante ao **CheckBox**, exceto pelo fato de que as opções são excludentes dentro de um mesmo grupo que você deve definir dentro da propriedade **groupValue**.
- **Radios** não aparecem sozinhos, vários devem existir. Em especial, cada **Radio** deve ter uma propriedade **value** diferente dentro de um **groupValue**.

Radio

```
Widget myRadio(int value) {
  return Radio(
    value: value, // Valor deste botão
    groupValue: completeForm.radioValue, // Valor do grupo
    onChanged: (int? inValue) {
      if (inValue != null) {
        setState(() {
            completeForm.radioValue = inValue;
        });
      }
    },
  );
}
```

SnackBar

SnackBar

- Componente que mostra uma mensagem no rodapé da tela por um período de tempo. O usuário poderá clicar para fazer a mensagem sumir.
- Para mostrar uma mensagem no **SnackBar**, usamos uma chamada para ScaffoldMessenger.of para obter uma referência ao **ScaffoldMessenger**, que por sua vez possui um método chamado **showSnackBar**.
- Note que na chamada do método **of** passamos um **BuildContext** como parâmetro. Este **BuildContext** iniciará uma busca pela widget tree, deste ponto em diante em direção ao nó raiz até encontrar um **ScaffoldMessenger** na árvore.
- Este **ScaffoldMessenger**, como o próprio nome está dizendo, está associado ao **Scaffold** que você adicionou próximo do início da widget tree.

SnackBar

A propriedade **action** do **SnackBar**, quando colocada apresenta um texto clicável. Não use esse clique para algo importante, pois a maioria dos usuários não verá.

```
ElevatedButton(
  child: const Text("Mostrar SnackBar!"),
  onPressed: () {
    ScaffoldMessenger.of(context).showSnackBar(SnackBar(
        backgroundColor: Colors.green,
        duration: const Duration(seconds: 5),
        content: const Text("Obrigado"),
        action: SnackBarAction(
          label: "Volte Sempre! ${completeForm.radioValue}",
          onPressed: () {})));
}, );
```

BottomSheet

BottomSheet

- Uma evolução do **SnackBar**. O **BottomSheet** permite criar uma tela customizável da maneira que o programador bem entender.
- Da maneira como criaremos a seguir, a tela será persistente e só desaparecerá quando o usuário clicar em algum dos botões. O usuário poderá, no entanto, interagir com o restante do aplicativo.
- Se você quiser bloquear o restante do aplicativo, então você precisará utilizar um **ModalBottomSheet**.

BottomSheet

```
ElevatedButton(
  onPressed: () {
    showBottomSheet(
      context: context,
      backgroundColor: Colors.green,
      builder: (_) {
        return Row(
                children: [
                        Expanded(
                          child: Column(
                            children: [
```

```
const Text("Quem é o seu professor favorito:"),
TextButton(
  child: const Text("Guilherme Coelho"),
  onPressed: () {
    Navigator.of(context).pop();
}),
TextButton(
  child: const Text("Celmar Guimarães"),
  onPressed: () {
    Navigator.of(context).pop();
}),
TextButton(
  child: const Text("Ulisses Martins Dias"),
  onPressed: () {
    Navigator.of(context).pop();
```

```
              mainAxisSize: MainAxisSize.min,
            ),
          )
        ],
      );
    },
  );
  child: const Text("Vote no Professor Favorito")
);
```