Aula de widgets e formatações

- vá ao site do flutter e mostre parte da documentação dos widgets <https://flutter.dev/docs/development/ui/widgets>

- Abra o basic widgets

* + - Container é um widget de criado para armazenar outros widgets. Ele permite incluir uma caixa de tamanho limitado para inserir outros widgets dento dela.
    - Column é um widget que permite organizar os outros widgets uns abaixo dos outros
    - Row é um widget que permite organizar os outros widgets uns ao lado dos outros
    - Scaffold é um widget Implementa a estrutura básica de layout visual do material design.

- Por falar em material design todos os componentes do flutter são baseados no material design. O material design é um conjunto de boas práticas de design propostas pelo google. Entre no site <https://material.io>. Lá são descritas essas boas práticas descritas pelo Google.

- Crie um aplicativo vazio para exibir piadas curtas. **Nomeie o projeto como piadas**

**-** Crie a estrutura básica stateless para um aplicativo.

void main() {

  runApp(MaterialApp(home: MyApp()));

}

class MyApp extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return Container();

  }

}

- Vamos falar um pouco sobre os organizadores de layout colunas e linhas. Os layouts são normalmente são widgets invisíveis que nos auxiliam na organização dos demais widgets na interface da aplicação.

- Vamos fazer um exemplo com o widget de coluna. Esse widget lhe permite posicionar os widgets nele aninhados uns ao lado dos outros.

- Altere o código do seu App apagando o Container e inserindo no lugar dele um Column

return Column(

      children: [Text("texto 1"), Text("texto 2"), Text("texto 2")],

    );

- Vamos nos ater por agora simplesmente em entender esses widgets como controladores de fluxo, não vamos nos preocupar com as questões de formatação.

- Altere o Column, por row.

return Row(

      children: [Text("texto 1"), Text("texto 2"), Text("texto 2")],

    );

- Agora vamos iniciar algumas formatações básicas usando o Text

- Deixe somente um Text dentro do Row.

- Vamos inserir um style. Essa propriedade necessita de um Tipo de dado TextStyle.

- Segure o ctrl e mostre o conjunto de propriedades existentes nessa classe de formatação

- Vamos trabalhar com a propriedade style para definir o tamanho da tipografia.

   return Row(

      children: [

        Text(

          "texto 1",

          style: TextStyle(

              fontSize: 40,

              color: Colors.black,

              fontStyle: FontStyle.italic,

              fontWeight: FontWeight.bold,

              letterSpacing: 10,

              wordSpacing: 20,

              decoration: TextDecoration.lineThrough,

              decorationColor: Colors.greenAccent),

        ),

      ],

    );

- Explicar as propriedades e mostrar os efeitos de cada uma delas. Decoration são as linhas de sublinhado que aparecem no texto.

- Para facilitar a formatação repetitiva de texto o ideai é construir uma função que retorne o Text com as opções de formatação já preenchidas. Semelhante a um CSS

Text formata(String texto) {

    return Text(

      texto,

      style: TextStyle(

          fontSize: 40,

          color: Colors.black,

          fontStyle: FontStyle.italic,

          fontWeight: FontWeight.bold,

          letterSpacing: 10,

          wordSpacing: 20,

          decoration: TextDecoration.lineThrough,

          decorationColor: Colors.greenAccent),

    );

  }

  @override

  Widget build(BuildContext context) {

    return Row(

      children: [formata("texto1")],

    );

  }

- Agora vamos ver como utilizar botões

- Apague o formata que está dentro do children

- Vamos criar um ElevatedButton(onPressed: onPressed, child: child)

- Vamos definir um child que é o conteúdo do botão.

ElevatedButton(onPressed: (){},

                child: Text("Botão"))

- Clique com o ctrl no ElevatedButton e comente sobre as propriedades

- Lembre-se que todas as propriedades já aprendidas para formatar um texto são válidas para esse Text de conteúdo do botão.

- Vamos entender um pouco mais sobre a formatação geral do aplicativo.

- Vamos trabalhar com espaçamento.

- Apague o botão deixando somente um texto na sua aplicação

 return Row(

      children: [Text("Botão")],

    );

- Vamos criar um container que envolva o row.

  return Container(

        child: Row(

      children: [Text("Botão")],

    ));

O containe lhe permite customizar alguns espaçamentos e bordas dessa caixa invisível de organização.

 return Container(

        padding: EdgeInsets.all(10),

        margin: EdgeInsets.all(5),

        decoration: BoxDecoration(border: Border.all(width: 3)),

        child: Row(

          children: [Text("Botão")],

        ));

- Padding é o espaçamento interno, margin é o espaçamento externo e o decoration lhe permite altera a aparência da caixa do container.

- EdgeInsets é a classe que lhe permite configurar essas medidas. O método all insere a mesma configuração para todos os lados e fromLTRB configura individualmente esses parâmetros para cada lado da caixa, temos também o only que lhe permite configura somente algumas medidas.

  return Container(

        padding: EdgeInsets.all(10),

        margin: EdgeInsets.only(left: 10,top: 10),

        decoration: BoxDecoration(border: Border.all(width: 3)),

        child: Row(

          children: [Text("Botão")],

        ));

- O para fazer o espaçamento entre widgets você pode definir um widget Padding

child: Row(

          children: [

            Text("Botão"),

            Padding(

              padding: EdgeInsets.only(left: 10),

              child: Text("botão 2"),

            )

- Agora vamos falar um pouco sobre alinhamento

- Apague todo conteúdo dentro children:[]

- Vamos criar dentro do children 3 widget Text

 return Container(

        padding: EdgeInsets.all(10),

        margin: EdgeInsets.only(left: 10, top: 10),

        decoration: BoxDecoration(border: Border.all(width: 3)),

        child: Row(

          children: [

            Text("T1"),

            Text("T2"),

            Text("T3")

          ],

        ));

- Agora vamos definir alguns alinhamentos para o widget row. 0 MainAxisAlignment define o alinhamento do eixo principal do widget que no caso do row é o eixo horizontal, mas no caso do column é o vertical. O mainAxisAlignment pode ter configurações de center para centraliza, end para alinha a direita e spaceEvenly que dá um espaçamento uniforme entre os widgets da row. O spaceAround mantem um espaçamento entre os objetos do meio maior que os da borda e o spaceBetWeen ele remove os espaços das extremidades e mantem o espaçamento entre os objetos.

child: Row(

          children: [

            Text("T1"),

            Text("T2"),

            Text("T3")

          ],mainAxisAlignment: MainAxisAlignment.spaceEvenly,

- Agora vamos falar do crossAxisAligment que é o eixo oposto ao mainAxis. No caso da row ele é o vertical e no caso do colunm ele é o horizontal. Ele tem as mesmas propriedades do mainAxis

- Vamos falar da inserção de imagens no projeto.

- Crie uma pasta na raiz do projeto chamada imagens e coloque duas imagens do seu gosto.

- Abra o arquivo pubspec.yaml

- Procure assets dentro dos comentários do arquivo

  # assets:

  #   - images/a\_dot\_burr.jpeg

  #   - images/a\_dot\_ham.jpeg

- Para utilizar os recursos adicionados você deve habilitar isso no pubspec.yaml. Copie a palavra assets e cole abaixo do comentário. Digite – images/nomedaimage.jpg.

- clique para atualizar o pubspec

- clique para fazer o upgrade package

- Salve e feche o arquivo

- volte ao arquivo main.dart

- no child do container insira um widget Image.assets e coloque o caminho da imagem dentro do projeto.

 Container(

      padding: EdgeInsets.all(10),

      margin: EdgeInsets.only(left: 10, top: 10),

      decoration: BoxDecoration(border: Border.all(width: 3)),

      child: Image.asset("images/picapau.jpg"),

    );

- Clique com o ctrl no Image para mostrar as opções de construtores existentes. Mostre os parâmetros do construtor asset.

- Para controlar a exibição da imagem temos a opção do fit: essa propriedade nos permite alterar a forma como a imagem irá ocupar os espaço disponível no widget pai dela. O padrão de valor é o contain que ocupa o espaço disponível sem distorcer a imagem. Vamos utilizar o BoxFit.cover para modificar essa visualização.

 child: Image.asset(

        "images/picapau.jpg",

        fit: BoxFit.cover,

      ),

- Altere o BoxFit.fill esse preenche todo os espaço, mas irá distorcer a imagem.

- Para evitar a distorção use o BoxFit.fitWidth ou BoxFit.firHeight e escolha em qual direção deseja encaixar o tamanho da imagem.

child: Image.asset(

        "images/picapau.jpg",

        fit: BoxFit.fitWidth

      )

- Temos também o o BoxFit.scaleDown que ajusta a imagem mantendo as proporções de largura e altura originais da imagem

 child: Image.asset(

        "images/picapau.jpg",

        fit: BoxFit.scaleDown

      ),

- Vamos ver agora o widget Scaffold que é o esqueleto base da maior parte das aplicações. Ele já lhe permite inserir as 3 áreas principais de um app. Barra de titulo, área de conteúdo e rodapé.

- Vamos apagar o container da nossa aplicação principal e vamos inserir um scaffold.

- Insira as 3 áres appbar, body, buttonNavigationBar

 return Scaffold(

        appBar: AppBar(

          title: Text("título"),

          backgroundColor: Colors.amber,

        ),

        body: Text("conteúdo"),

        bottomNavigationBar: BottomAppBar(

          color: Colors.blueAccent,

          child: Padding(

            padding: EdgeInsets.all(20),

            child: Row(

              children: [

                ElevatedButton(onPressed: () {}, child: Text("voltar"))

              ],

            ),

          ),

        ));

**Entrada de dados**

Agora vamos iniciar nossos estudos sobre os campos de entrada. E o primeiro deles é o TextField.

Com um aplicativo padrão com scaffold no body vamos definir um TextField no lugar do contêiner que é o padrão. Escreva o código abaixo e troque os tipos TextInputType para os que estão no comentário do código

TextField(

              //text, number, emailAddress, datetime

              keyboardType: TextInputType.number,

              decoration: InputDecoration(

                labelText: "Digite um valor"

              ),

style: TextStyle(

                fontSize: 25,

                color: Colors.green

              ),

Mostre as propriedades enabled e maxLenght.

Inclua a propriedade obscureText:true para colocar máscara de senha no campo

TextField(

              //text, number, emailAddress, datetime

              keyboardType: TextInputType.number,

              decoration: InputDecoration(

                labelText: "Digite um valor"

              ),

style: TextStyle(

                fontSize: 25,

                color: Colors.green

              ),obscureText:false,

Agora precisamos saber como recuperar o conteúdo desse campo de texto. E para isso utilizaremos um evento. Perceba que nesse caso a função anônima possui um parâmetro String, pois é nele que será retornado o valor que será incluso no campo. O evento de onChanged executa a cada alteração da caixa, sendo assim ele será chamado várias vezes. Isso pode não ser interessante então vamos conhecer outros eventos.

TextField(

              //text, number, emailAddress, datetime

              keyboardType: TextInputType.number,

              decoration: InputDecoration(

                labelText: "Digite um valor"

              ),

style: TextStyle(

                fontSize: 25,

                color: Colors.green

              ),obscureText:false,

onChanged: (String texto){

                print("valor digitado:" +  texto);

              }

Assim vamos trocar o evento para OnSubmitted que so envia os dados após a confirmação do usuário que irá clicar no botão de conformar os dados digitados no teclado.

TextField(

              //text, number, emailAddress, datetime

              keyboardType: TextInputType.number,

              decoration: InputDecoration(

                labelText: "Digite um valor"

              ),

style: TextStyle(

                fontSize: 25,

                color: Colors.green

              ),obscureText:false,

 onSubmitted: (String texto){

                print("valor digitado:" +  texto);

              },

Ok, mas como acessar esse o conteúdo desse campo de texto fora desses eventos? Para isso precisamos de um controlador.Uutilizaremos o controlador, pois ele é mais flexível

Para isso precisaremos de uma vaiável do tipo TextEditingController.

Antes do @override declare a variável

 TextEditingController \_textEditingController = TextEditingController();

Agora defina a propriedade controller com TextField

onSubmitted: (String texto){

                print("valor digitado:" +  texto);

              },

              controller: \_textEditingController,

Pronto agora em qualquer lugar do seu código caso queira recuperar o valor do campo de texto basta acessa **textEditingController.text**

Agora vamos conhecer o Text Form Field que une as características de um campo de entrada de texto com um Text.

// Widget text

  \_editText(String field, TextEditingController controller) {

    return TextFormField(

      controller: controller,

      validator: (s) => \_validate(s, field),

      keyboardType: TextInputType.number,

      style: TextStyle(

        fontSize: 22,

        color: Colors.blue,

      ),

      decoration: InputDecoration(

        labelText: field,

        labelStyle: TextStyle(

          fontSize: 22,

          color: Colors.grey,

        ),

      ),

    );

  }

E a segunda parte é como acessar esse o conteúdo desse campo de texto é realizado, por informações enviadas pelo form ou por um controller associado ao campo e nesse exemplo utilizaremos o controlador, pois ele é mais flexível

 final \_tPeso = TextEditingController();

 final \_tAltura = TextEditingController();

Para isso eu devo declarar as instancias do TextEditingController e depois associar essas variáveis ao campo de texto através da propriedade controller.

Posso também criar validações para esse campo usando uma função e associando a mesma a propriedade validator do campo de texto

// PROCEDIMENTO PARA VALIDAR OS CAMPOS

  String \_validate(String text, String field) {

    if (text.isEmpty) {

      return "Digite $field";

    }

    return null;

  }

Vamos ver agora o checkBox esse campo permite que usuário selecione uma ou mais opções de um conjunto de opções disponíveis.

Crie uma aplicação padrão e configure o retorno para um scaffold. Veja o código abaixo.

 Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        title: Text("Entrada de dados"),

      ),

      body: Container(

        child: Column(

          children: <Widget Text("Comida Brasileira"),

            Checkbox(

              onChanged: (bool valor){

                print("Checkbox: " + valor.toString() );

              },

            )

          ],

        ),

      ),

    );

  }

Ao executar o código você perceberá que mesmo marcando e desmarcando o checkBox não se altera, mas ele executa o print mostrando que o valor do parâmetro está sendo alterado.

Para que isso ocorra precisamos armazenar o valor em uma variável e configurar o valor do checkBox para esse valor.

 bool \_estaSelecionado = false;

  @override

onChanged: (bool valor){

                setState(() {

                  \_estaSelecionado = valor;

                });

                print("Checkbox: " + valor.toString() );

              },

Agora teste a aplicação e perceba que a modificação do valor altera o estado da caixa.

Vamos conhecer uma variação do CheckBox que é o checkBoxListTile. Esse widget nos permitirá configurar mais algumas propriedades para esse objeto.

Então vamos alterar o nosso programa colocando o seguinte código no lugar do checkBox atual.

 bool \_comidaBrasileira = false;

  @override

child: Column(

          children: <Widget>[

            CheckboxListTile(

              title: Text("Comida Brasileira"),

                subtitle: Text("A melhor comida do mundo!!"),

                //activeColor: Colors.red,

                //selected: true,

                //secondary: Icon(Icons.add\_box),

                value: \_comidaBrasileira,

                onChanged: (bool valor){

                  setState(() {

                    \_comidaBrasileira = valor;

                  });

                }

            ),

Vamos ver agora o RadioButton esse campo permite que usuário selecione uma ou mais opções de um conjunto de opções disponíveis.

Crie uma aplicação padrão e configure o retorno para um scaffold. Veja o código abaixo.

  int \_escolhaUsuario;

  @override

body: Container(

        child: Column(

          children: <Widget>[

            RadioListTile(

              title: Text("Masculino") ,

                value: 1,

                groupValue: \_escolhaUsuario,

                onChanged: (int escolha){

                  setState(() {

                    \_escolhaUsuario = escolha;

                  });

                }

            ),

            RadioListTile(

                title: Text("Feminino") ,

                value: 2,

                groupValue: \_escolhaUsuario,

                onChanged: (int escolha){

                  setState(() {

                    \_escolhaUsuario = escolha;

                  });

                }

            ),

            EvelatedButton(

                child: Text(

                  "Salvar",

                  style: TextStyle(

                      fontSize: 20

                  ),

                ),

                onPressed: (){

                  print("Resultado: " + \_escolhaUsuario.toString() );

                }

            )

Aqui a propriedade groupvalue é quem define a escolha do usuário, por isso ele deve ser o mesmo para todo o grupo.

A diferença do Radio para o radioListTile é que o listTile já configura o layout para uma lista de itens uns abaixo dos outros. Se quiser conferir é só trocar de RadioLlistTile para Radio.

Por último vamos conhecer o o widget SwitchListTile e Switch.

Crie uma aplicação padrão e configure o retorno para um scaffold. Veja o código abaixo.

bool \_escolhaUsuario = false;

  @override

body: Container(

        child: Column(

          children: <Widget>[

            SwitchListTile(

              title: Text("Receber notificações?"),

                value: \_escolhaUsuario,

                onChanged: (bool valor){

                  setState(() {

                    \_escolhaUsuario = valor;

                  });

                }

            ),

            ElevatedButton(

                child: Text(

                  "Salvar",

                  style: TextStyle(

                      fontSize: 20

                  ),

                ),

                onPressed: (){

                  if( \_escolhaUsuario ){

                    print("escolha: ativar notificações");

                  }else{

                    print("escolha: NÃO ativar notificações");

                  }

                  //print("Resultado: " + \_escolhaUsuario.toString() );

                }

            )