***CURSO DE FLUTTER***

**2. Noções básicas de Flutter**

31. Construindo o Primeiro Widget #02

***CURSO DE FLUTTER***

**0:21**

**2. Noções básicas de Flutter**

31. Construindo o Primeiro Widget #02

***Função***main()

A função main() é a porta de entrada de uma aplicação ***Flutter.*** É dentro dela que é chamada a função runApp() que roda a aplicação em si.

**0:36**

**2. Noções básicas de Flutter**

31. Construindo o Primeiro Widget #02

Para isso, será criado uma ***Classe*** *"que é o Widget"* e o nome da Classe será ***PerguntaApp***class PergutaApp  que estenderá, extends  ou seja, há uma herança de StatelessWidget . ***Exemplo:*** class PergutaApp extends StatelessWidget{ }

**1:17**

**2. Noções básicas de Flutter**

31. Construindo o Primeiro Widget #02

Será implementado um *Método*que precisa retornar um ***Widget***e esse *Método*é o ***build()***  e ele irá receber um ***BuildContext*** que será chamado de ***context***,  que irá retornar um ***Widget***. ***Exemplo:***build(BuildContext context){}.

Dentro das chaves do *Método* ***build***, o ***Widget***retornado ***return***, será o ***MaterialApp(); . Exemplo:*** return MaterialApp();.

Digitando *CTRL+ESPAÇO*no teclado dentro dos parênteses de *MaterialApp*, irá mostrar todos os ***atributos nomeados*** que poderão ser usados, que no caso em questão, será o ***home*** e será passado um outro ***Widget*** de texto ***Text()*** para colocar o nome da ***home. Exemplo:*** home: Text('Olá Flutter!!!'),.

**2:50**

**2. Noções básicas de Flutter**

31. Construindo o Primeiro Widget #02

***Código de Exemplo:***

1. class PerguntaApp extends StatelessWidget {
3. Widget build(BuildContext context) {
4. return MaterialApp(
5. home: Text('Olá Flutter!!!'),
6. );
7. }
8. }

**3:59**

**2. Noções básicas de Flutter**

31. Construindo o Primeiro Widget #02

Para chamar o ***App***, será usada a função ***runApp()*** passando como parâmetro uma instância do componente ***PerguntaApp()***.

***Dica:****ele não recebe nenhum parâmetro por usar o construtor padrão.*

***Exemplo:***

1. main(){
2. runApp(PerguntaApp());
3. }

**4:16**

**2. Noções básicas de Flutter**

31. Construindo o Primeiro Widget #02

***CÓDIGO COMPLETO***

1. import 'package:flutter/material.dart';
3. main() {
4. runApp(PerguntaApp());
5. }
7. class PerguntaApp extends StatelessWidget {
8. Widget build(BuildContext context) {
9. return MaterialApp(
10. home: Text('Olá Flutter'),
11. );
12. }
13. }

**0:09**

**2. Noções básicas de Flutter**

32. Resumo até Aqui

***BREVE RESUMO:***

No *Método****main()***, ao invés de usar a *abertura e fechamento* de chaves, pode-se usar uma ***arrow function***, sendo mais comum ser encontrada nos códigos.

***Exemplo:*** main() => runApp(PerguntaApp());

O *Decorator* ***@override*** é um *Método* que obrigatoriamente o *Componente****Stateless***precisa implementar.

O *Método* ***@override***, estará sobrescrevendo o *Método* ***build()***, sendo, que esse *Método* que irá chamar é o *Flutter.*

**1:09**

**2. Noções básicas de Flutter**

32. Resumo até Aqui

O ***Nó Raiz*** da *árvore de componentes*, é todo o resto estará debaixo deste componente raiz é o primeiro componente passado por parâmetro ao ***runApp()***que no caso é o ***PergntaApp().***

E o Componente terá o *Método****build()*** que recebe um ***BuildContext***quem passará esse contexto de construção é o próprio *Flutter*.

Cada Componente tem seu próprio contexto, e o componente ***Filho****(já que temos uma árvore de componentes)* ele tem como a partir do contexto, ter uma referência para o contexto do ***Pai***e ele consegue se comunicar através desses contextos, mas cada componente tem seu próprio contexto.

**2:09**

**2. Noções básicas de Flutter**

32. Resumo até Aqui

O ***Widget principal*** no caso ***"Pai"***, é o ***PerguntaApp()*** e o *Widget* ***Filho*** direto é o ***MaterialApp()***, também está sendo usando um parâmetro nomeado **home:***e* no Text() foi passado um parâmetro posicional, ou seja, o primeiro parâmetro passado é exatamente o valor do texto que ele vai assumir, eventualmente se for colocado uma vírgula após as aspas, se terá acesso a outros parâmetros nomeados.

Os parâmetros posicionais são obrigatórios, os nomeados são opcionais e eventualmente podem ter um valor padrão.

**0:21**

**2. Noções básicas de Flutter**

33. Árvore de Widgets

***CONSTRINDO A ÁRVORE DE COMPONENTES***

Modificando a ***home:*** para ao invés de só usar o *Componente****Text()*** para colocar um texto simples, irá ser colocado outro *Componente,*o **Scaffold()** que cria a *estrutura do Aplicativo*, onde serão colocados outros *Widgets*para dar forma ao ***App***.

**0:51**

**2. Noções básicas de Flutter**

33. Árvore de Widgets

Digitando ***CTRL+ESPAÇO*** vai abrir um *menu de opções* com parâmetros nomeados que poderão ser usados dentro do ***Scaffold(),*** será usado o*parâmetro nomeado* ***appBar:*** e o *Componete****AppBar()*** que cria a barra superior do App.

Dentro de ***AppBar()***, será criado outro *parametro nomeado* e outro *componente*dando assim forma a ***Árvore de Widgets.***

Digitando ***CTRL+ESPAÇO*** dentro de ***AppBar()***, aparecerá a opção de criar o título através do *p.arâmetro nomeado* ***title:*** sendo que ele espera como parâmetro um *Widget*, tendo então de se prestar muita atenção para não se tentar colocar uma *string* no lugar ***Ex.: title: "Título de Exemplo"***, pois será um erro gravíssimo.

O correto é usar o *Widget****Text()***, title: const Text(`Perguntas`)

**0:53**

**2. Noções básicas de Flutter**

33. Árvore de Widgets

***Dica****: usando****const****no****title:****, evita uso de memória e processamento  desnecessários.*

**2:46**

**2. Noções básicas de Flutter**

33. Árvore de Widgets

E por último, saindo de ***appBar:*** e voltando para ***home:*** clicando em ***CTRL+ESPAÇO*** vamos encontrar *parâmetro nomeado* ***body:*** que é o *corpo da aplicação* e que também espera um *Widget*como parâmetro. Que o caso vai ser usado um texto simples por enquanto. ***Ex.:***body: Text('Olá Flutter!!!'), .

***EXEMPLO DO CÓDIGO ALTERADO***

1. // ignore\_for\_file: use\_key\_in\_widget\_constructors
3. import 'package:flutter/material.dart';
5. main() => runApp(PerguntaApp());
7. class PerguntaApp extends StatelessWidget {
8. @override
9. Widget build(BuildContext context) {
10. return MaterialApp(
11. home: Scaffold(
12. appBar: AppBar(
13. title: const Text('Perguntas'),
14. ),
15. body: Text('Olá Flutter'),
16. ));
17. }
18. }

**0:04**

**2. Noções básicas de Flutter**

37. Usando Column para Organizar o Widgets

***USANDO COLUNAS***Column()***PARA ORGANIZAR OS WIDGETS***

**0:50**

**2. Noções básicas de Flutter**

37. Usando Column para Organizar o Widgets

Um dos *Componentes*usados para organizar *layouts*dentro do *Flutter*é o Column()  e dentro de Column irá receber um *atributo* chamado *children* que recebe como parâmetro um *Widget* que por sua vez retorna uma ***lista de Widgets****.*

***Exemplo:***

1. Column(
2. children: <Widget>[
3. Text(`Linha 1`),
4. Text(`Linha 2`),
5. Text(`Linha 3`),
6. ],
7. )

**1:54**

**2. Noções básicas de Flutter**

37. Usando Column para Organizar o Widgets

***.FAZENDO O MESMO PROCEDIMETO MAIS RÁPIDO:***

Usando o atalho ***CTRL+PONTO*** vai abrir um menu várias opções para ***envolver (wrap)*** esse trecho de código e nesse caso será envolvido por uma coluna ***Column()***.

Dentro da Aplicação será criado uma lista de Perguntas:

***Exemplo:***

1. Final pergutas = [
2. 'Qual é a sua cor favorita?',
3. 'Qual é o seu animal favorito?',
4. ];

**3:46**

**2. Noções básicas de Flutter**

37. Usando Column para Organizar o Widgets

Agora como as *Perguntas*estão em uma em uma *lista*em uma constante Final, ela pode ser acessada pelo *Widget*Text() através da ***notação ponto:*** Text(perguntas.elements(0)), ou acessando diretamente a lista: Text(perguntas[0]), acessando assim, o primeiro elemento da lista que é a primeira pergunta.

**4:10**

**2. Noções básicas de Flutter**

37. Usando Column para Organizar o Widgets

Para se criar um botão RaisedButton(), será preciso dois ***parâmetros/atributo***.

O primeiro atributo é o ***child*** (filho) child: , onde, ele recebe um outro *Widget*: ***Exemplo:****uma coluna, uma linha, um ícone, uma lista, um texto, um ícone e um texto ao mesmo tempo, ...*

Nesse caso o child: irá receber um texto simples: child: Text(`Resposta 1`), .

O outro atributo que RaisedButton() recebe é a ação do click do botão ele executar uma função, o ***onPressed:***onPressed: null,  está nulo por ñ ter uma função ainda.

**5:34**

**2. Noções básicas de Flutter**

37. Usando Column para Organizar o Widgets

***CÓDIGO DE EXEMPLO ATÉ O MOMENTO:***

*Já com alterações Dart 2.14*

1. // ignore\_for\_file: use\_key\_in\_widget\_constructors
3. import 'package:flutter/material.dart';
5. main() => runApp(PerguntaApp());
7. class PerguntaApp extends StatelessWidget {
8. @override
9. Widget build(BuildContext context) {
10. final perguntas = [
11. 'Qual é a sua cor favorita?',
12. 'Qual é o seu animal favorito?',
13. ];
15. return MaterialApp(
16. home: Scaffold(
17. appBar: AppBar(
18. title: const Text('Perguntas'),
19. ),
20. body: Column(children: [
21. Text(perguntas.elementAt(0)),
22. const ElevatedButton(
23. onPressed: null,
24. child: Text('Resposta 1'),
25. ),
26. const ElevatedButton(
27. onPressed: null,
28. child: Text('Resposta 2'),
29. ),
30. const ElevatedButton(
31. onPressed: null,
32. child: Text('Resposta 3'),
33. ),
34. ]),
35. ),
36. );
37. }
38. }

**0:03**

**2. Noções básicas de Flutter**

39. Funções Anônimas

***FUNÇÕES ANÔNIMAS***

Para habilitar um botão, e preciso criar uma ***função anônima*** no atributo *onPressed*. ***Exemplo:*** onPressed: () {} .

**0:29**

**2. Noções básicas de Flutter**

39. Funções Anônimas

***DICA: Quando se passa uma função como parâmetro, ñ se coloca os parênteses da Função, passe-se somente o nome da Função.***

**4:00**

**2. Noções básicas de Flutter**

39. Funções Anônimas

Pode se habilitar o click de um botão onPressed: de 4 formas diferentes:

1. Declarando uma Função vazia: ***Exemplo:***onPressed; () {},
2. Declarando uma Função que ñ retorna nada e a chamando por parâmetro.
   1. void responder(){
   2. print('Resposta 1');
   3. }
   5. onPressed: responder,
3. Declarando a Função diretamente no parâmetro ***onPressed.***

***Exemplo:***

* 1. onPressed: () {
  2. print('Resposta 2 Selecionada');
  3. }

1. Declarar diretamente usando a ***Função Arrow***.

***Exemplo:***onPressed: () => print('Resposta 3!!!!'),

**6:14**

**2. Noções básicas de Flutter**

42. Usando StatefulWidget #02

***CONVERTENDO UM STATELESSWIDGET EM STATEFULWIDGET***

**6:23**

**2. Noções básicas de Flutter**

42. Usando StatefulWidget #02

***Primeiro passo:*** trocar a herança extends  ao invés do componente ***PerguntaApp*** herdar de um *StatelessWidget*, irá herdar de um ***StatefulWidget.***

***Segundo passo:*** foi colocado dentro do estado State  a variável de instancia da Classe que irá ser alterada a todo instante, pois, será o estado que irá controlar essa variável.

***Terceiro passo:*** como o Método responder()  também foi movido para dentro do ***estado***State  pois também precisava do ***estado***.

***Quarto passo:*** como a *árvore de componentes* também precisam do ***estado,*** ou seja, quando a variável mudar, a *árvore de componentes* precisa se atualizar de forma reativa, o Método build()  também foi movido para dentro do ***estado***State***.***

**7:05**

**2. Noções básicas de Flutter**

42. Usando StatefulWidget #02

***Quinto passo:*** para conectar o Componente ***StatefulWidget*** PerduntaApp com o ***estado***PerguntaAppState  foram feitos em dois pontos:

1. A partir da herança extends foi passado no *Generics* qual foi o *Tipo*de Componete ***StatefulWidget*** estará ligado a esse State específicamente <PerguntaApp> .
2. E no *Componente StatefulWidget* ***PerguntaApp***foi criado um *Método*chamado createState() que espera um *Componente do Tipo* PerguntaAppState e irá retornar return o estado ***State***PerguntaAppState().

**7:29**

**2. Noções básicas de Flutter**

42. Usando StatefulWidget #02

***PerguntaApp***, é o Componente que estará presente na *Árvore de Componentes*, tanto, que ele está sendo passado para o ***runApp***para ser chamado assim que o aplicativo for aberto runApp(PerguntaApp());

Como o ***Flutter***identifica que ***PergutaApp***é um *Componente StatefulWidget*, ele irá chamar o ***estado***e a dentro de PerguntaAppState que gerencia o ***estado***e a ***Árvore de Componentes***.

**8:32**

**2. Noções básicas de Flutter**

42. Usando StatefulWidget #02

***Último passo:*** para que a***Interface Gráfica*** seja ***notificada***quando a ***variável***for alterada, é colocar dentro de um setState((){}) o trecho de comando ***pergutaSelecionada++;***.

***Exemplo:***

1. setState((){
2. pergutaSelecionada++;
3. });

Dessa forma, ele irá ficar monitorando se houve alguma mudança e essa mudança será atualizada na página da aplicação.

**5:52**

**2. Noções básicas de Flutter**

44. Tornando Membros Privados

***TRANSFORMANDO Variáveis, Métodos, Classes e Componentes em PRIVADOS***

**6:01**

**2. Noções básicas de Flutter**

44. Tornando Membros Privados

Colocando um *underline* (***\_***) na frente de uma *Classe, Variável, Método ou Componente*, o tornará ***Privado***ou visível somente dentro do arquivo onde foi criado.

E para ser acessado externamente, deverá ser criado um *Método****get***e um ***set***.

**7:05**

**2. Noções básicas de Flutter**

44. Tornando Membros Privados

***CÓDIGO ATUALIZADO COM CLASSE, METODOS E VARIÁVEIS PRIVADOS***

1. // ignore\_for\_file: use\_key\_in\_widget\_constructors
3. import 'package:flutter/material.dart';
5. main() => runApp(PerguntaApp());
7. class \_PerguntaAppState extends State<PerguntaApp> {
8. var \_perguntaSelecionada = 0;
10. void \_responder() {
11. setState(() {
12. \_perguntaSelecionada++;
13. });
15. // ignore: avoid\_print
16. print(\_perguntaSelecionada);
17. }
19. @override
20. Widget build(BuildContext context) {
21. final perguntas = [
22. 'Qual é a sua cor favorita?',
23. 'Qual é o seu animal favorito?',
24. ];
26. return MaterialApp(
27. home: Scaffold(
28. appBar: AppBar(
29. title: const Text('Perguntas'),
30. ),
31. ...

**7:08**

**2. Noções básicas de Flutter**

44. Tornando Membros Privados

1. ...
2. body: Column(children: [
3. Text(perguntas[\_perguntaSelecionada]),
4. ElevatedButton(
5. onPressed: \_responder,
6. child: const Text('Resposta 1'),
7. ),
8. ElevatedButton(
9. onPressed: () {
10. // ignore: avoid\_print
11. print('Resposta 2 foi selecionada');
12. },
13. child: const Text('Resposta 2'),
14. ),
15. ElevatedButton(
16. // ignore: avoid\_print
17. onPressed: () => print('Resposta 3!!!'),
18. child: const Text('Resposta 3'),
19. ),
20. ...

**7:12**

**2. Noções básicas de Flutter**

44. Tornando Membros Privados

1. ElevatedButton(
2. onPressed: () {},
3. child: const Text('Resposta 4'),
4. ),
5. ]),
6. ),
7. );
8. }
9. }
11. class PerguntaApp extends StatefulWidget {
12. @override
13. \_PerguntaAppState createState() {
14. return \_PerguntaAppState();
15. }
16. }

O *Componente****PerguntaApp***continuará público, caso seja enviado para outro arquivo, continuará tendo acesso ao mesmo.

**0:12**

**2. Noções básicas de Flutter**

45. Criando Widget Questão

***CRIANDO UM NOVO WIDGET - QUESTÃO***

*Neste Widget ficará todas as formatações relacionadas a este Componente, evitando de poluir a Árvore de Componentes.*

**0:32**

**2. Noções básicas de Flutter**

45. Criando Widget Questão

Dentro da pasta ***lib*** será criado o arquivo ***questao.dart*** onde estarão todas as configurações relacionadas a ***Classe Questao.***

**2:26**

**2. Noções básicas de Flutter**

45. Criando Widget Questão

A ***Classe***criada terá o nome de ***Questao***: class Questao extends StatelessWidget{}

Dentro da ***Classe***será criado uma *variável****final***do *Tipo****String***de nome ***texto***: final String texto;

Irá criar um **Construtor**para a ***Classe***: Questao(this.texto);

E em build, irá criar um retorno return para o Text() usando a *variável*texto como *parâmetro*:

1. Widget build(BuildContext context){
2. return Text(texto);
3. }

**2:32**

**2. Noções básicas de Flutter**

45. Criando Widget Questão

***CÓDIGO CLASSE QUESTAO CRIADA***

1. import 'package:flutter/material.dart';
3. class Questao extends StatelessWidget {
4. final String texto;
6. Questao(this.texto);
8. @override
9. Widget build(BuildContext context) {
10. return Text(texto);
11. }
12. }

**2:39**

**2. Noções básicas de Flutter**

45. Criando Widget Questão

Dentro do ***main.dart***, deverá ser importada a ***Classe***criada anteriormente ***questao.dart***: import './questao.dart';

E o Componente ***Text()*** será ***substituído***pelo *Componente****Questao()*** recém criado juntamente com o parâmetro ***perguntas[\_perguntaSelecionada]:***

1. ...
2. body: Column(children: [
3. Questao(perguntas[\_perguntaSelecionada]),
4. ...

**3:42**

**2. Noções básicas de Flutter**

45. Criando Widget Questão

Com isso, se cria diversos componentes com configurações distintas que depois são agrupados na Árvore de componentes, criando assim aplicativos maiores.

**0:03**

**2. Noções básicas de Flutter**

47. Aplicando Estilo ao Widget Questão

***APLICANDO ESTILO AO COMPONENTE QUESTAO***

*Agora ficará claro o motivo de se dividir os componentes.*

**1:40**

**2. Noções básicas de Flutter**

47. Aplicando Estilo ao Widget Questão

1. Widget build(BuildContext context) {
2. return Text(
3. texto,
4. style: const TextStyle(fontSize: 28),
5. textAlign: TextAlign.center,
6. );
7. }

***TextAlign.center*** não irá funcionar, pois, o ***texto***ocupa todo o espaço do que está visível na tela.

Para conseguir centralizar o texto corretamente na tela, será preciso ***envolve-lo*** em um ***Container***para poder configurar outros ***atributos***relacionados ao Container

**3:30**

**2. Noções básicas de Flutter**

47. Aplicando Estilo ao Widget Questão

1. return Container(
2. width: double.infinity,
3. margin: const EdgeInsets.all(30),

width: double.infinity,  ajusta a largura do *Container*a largura da tela.

margin: const EdgeInsets.all(30),  ajusta a margem do *Container*em todos os lados.

margin: EdgeInsets.fromLTRB(left, top, right, bottom),  ajusta a margem do *Container*individualmente na ***esquerda***left, no ***topo***top, na ***direita***right e na ***base***bottom.

**4:26**

**2. Noções básicas de Flutter**

51. Passando Função Callback

1. import 'package:flutter/material.dart';
3. class Resposta extends StatelessWidget {
4. final String texto;
5. final void Function() quandoSelecionado;
7. Resposta(this.texto, this.quandoSelecionado);
9. @override
10. Widget build(BuildContext context) {
11. return Container(
12. width: double.infinity,
13. margin: const EdgeInsets.fromLTRB(50, 5, 50, 5),
14. child: ElevatedButton(
15. onPressed: quandoSelecionado,
16. child: Text(texto),
17. ),
18. );
19. }
20. }

A *Função****quandoSelecionado***será chamada quando o botão for clicado.

Já em ***main.dart*** o segundo parâmetro do *Componente* ***Resposta***, será usado o *Método*\_responder

**0:01**

**2. Noções básicas de Flutter**

53. Usando Tipo Map nas Perguntas

***OBS.: Correção dos erros de tipo Object e String***

Fala, pessoal!

Nessa aula vamos fazer uma pequena correção relacionada às próximas duas aulas. Devido a atualizações tanto no Dart quanto no Flutter, algumas questões em relação a tipagem mudaram um pouco.

Na próximas aulas, o código exato delas irá gerar um pequeno erro de tipagem na hora de transformar um Objeto em Lista e os elementos em String.

Para corrigir é bem simples. Basta adicionar duas funções em dois pontos do código, a função "cast()" para transformar um Objeto em Lista e a função "toString()" para transformar os elementos em Texto.

O primeiro caso vai acontecer lá na criação do Widget Questao:

1. Questao(perguntas[\_perguntaSelecionada]['texto'].toString()),

**0:15**

**2. Noções básicas de Flutter**

53. Usando Tipo Map nas Perguntas

O segundo caso, na aula seguinte, acontece lá no **for**:

1. for (String textoResp in perguntas[\_perguntaSelecionada].cast()['respostas']) { respostas.add(Resposta(textoResp, \_responder)); }

Basta adicionar as duas funções, cast() e toString() e tudo funcionará certinho!

Bons estudos!

**0:18**

**2. Noções básicas de Flutter**

53. Usando Tipo Map nas Perguntas

***USANDO TIPO MAP NAS PERGUNTAS***

**0:23**

**2. Noções básicas de Flutter**

53. Usando Tipo Map nas Perguntas

Atualizando o código atual, ao invés de retornar uma*lista de Strings*, irá retornar uma *lista de Maps* e o ***Map***será uma ***String***e um ***Objeto***.

***Exemplo:***List<Map<String, Object>>

O ***Map***normal retornando uma ***String***: {'texto': 'Qual é a sua cor favorita?'} chave *'texto'* e valor *'Qual é a sua cor favorita?'*.

***Map***retornando ***String***e ***Object***: {'respostas': ['Qual é a sua cor favorita?']}  chave *'respostas'* e valor *['Preto', 'vermelho', 'verde', 'branco']*, pois, o *Tipo****Object***retorna tanto ***String***quanto uma ***Lista***.

**3:48**

**2. Noções básicas de Flutter**

53. Usando Tipo Map nas Perguntas

No ***Componente Questao***, as perguntas eram acessadas através de uma ***Lista de Strings,*** agora elas serão acessadas por uma ***Lista de Maps***, contendo a pergunta e uma lista de respostas. Para acessar o valor de um Map, se acrescenta mais um para de colchetes e dentro "com aspas simples" se coloca o nome da chave que se quer acessar que no caso é: ['texto'].

***Ex.:***Questao(perguntas[perguntaSelecionada]['texto'].toString),

***Obs.:***o toString foi usado no final para converter o ***Map***em uma ***String*** e não dar erro no **Dart 2.14**.

**2:35**

**2. Noções básicas de Flutter**

55. Lista para Widgets #01

***DICA: ESTUDAR DIFERENÇA ENTRE CÓDIGO IMPERATIVO E CÓDIGO DECLARATIVO.***

***EXEMPLO DE CÓDIGO IMPERATIVO:***

**2:56**

**2. Noções básicas de Flutter**

55. Lista para Widgets #01

***Objetivo Principal:***

* Pegar a lista de respostas e colocar diretamente dentro da árvore de componentes;
* A partir de todas as perguntas, já se tem \_pergntaSelecionada;
* Então será criado um for que irá guardar em textoResp a pergunta selecionada perguntas[\_pergntaSelecionada]  a chave ['resposta']. Neste ***for***, irá dar todas as ***respostas*** relacionadas à ***pergunta***.
* ***Ex.:***for(String textoResp in perguntas[\_pergntaSelecionada]['respostas']){};
* Agora devemos criar um ***Componente do Tipo Resposta*** com todas as respostas, para isso, será criada uma *Lista*List de *Widget*<Widget> com nome de respostas que recebe uma Lista vazia ['']. ***Ex.:***List<Widget> respostas = [''];

**4:53**

**2. Noções básicas de Flutter**

55. Lista para Widgets #01

* Dentro do for será adicionado na ***Lista de Respostas*** respostas.add(), será criado uma resposta passando o textoResp e o *Método*\_responder que precisam ser passados por *parâmetro*para criar uma ***resposta***. ***Ex.:***respostas.add(Resposta(textoResp, \_responder)); Isso de forma imperativa dizendo exatamente o que deve ser feito, passo a passo, foi criado uma Lista, foi criado um for, foi pego exatamente o que se queria, foi adicionado os elementos na lista e agora se tem um ***Lista de Respostas***.

**5:43**

**2. Noções básicas de Flutter**

55. Lista para Widgets #01

Agora será atualizado na Árvore de Componentes o Componente Resposta.  
Será usado o operador três pontinhos "***spread***" ...  que irá fazer com que todos os ***Elementos***da *Lista*sejam adicionados nesta outra *Lista*que pertence ao *Column*

**0:05**

**2. Noções básicas de Flutter**

56. Lista para Widgets #02

***REFATORANDO O CÓDIGO***

*DE IMPERATIVO PARA DECLARATIVO USANDO O MAP*

**1:00**

**2. Noções básicas de Flutter**

56. Lista para Widgets #02

Refatorando o ***laço for:*** criar uma ***Lista de String*** com nome ***Respostas***List<String> respostas = perguntas[\_perguntaSelecionada]['respostas'];   
Com esse comando gera uma lista contendo a lista de respostas.

Agora com o código respostas.map  (que é o nome da Lista) vai receber cada uma das respostas (que é a Lista de resposta).

Agora vai receber os ***valores textuais***((texto) e converter esse texto em um ***Componente do Tipo Resposta*** => Resposta(  passar o ***texto***e o ***Método \_responder***  como ***parâmetro*** para o *Componente Resposta* texto, \_responder));  e como o map()  não está retornando uma ***Lista***mas sim um ***Widget***, então, devemos usar a *notação ponto* .toList()  para transformar tudo em uma ***Lista.***

**2:55**

**2. Noções básicas de Flutter**

56. Lista para Widgets #02

***Código Completo:***respostas.map((texto) => Respostas(texto, \_responder)).toList();

**3:12**

**2. Noções básicas de Flutter**

56. Lista para Widgets #02

Com essa alteração tende se uma abordagem mais funcional e consequentemente mais declarativa.  
Não foi preciso criar um*laço for*, foram chamados as ***funções de transformação*** e foi gerado uma ***Lista de Widgets.***

Foi trocado uma abordagem Imperativa (usando o laço for) por uma abordagem mais funcional "Declarativa"

**4:22**

**2. Noções básicas de Flutter**

56. Lista para Widgets #02

A partir da LIsta de Strings das respostas textuais *"List<String> Respostas"*foram gerados os Componentes os *"Widgets",*onde, o conteúdo será colocado diretamente na ***Árvore de Componentes.***

1. ...
2. body: Column(children: [
3. Questao(perguntas[\_perguntaSelecionada]['texto'].toString()),
4. ...respostas.map((texto) => Respostas(texto, \_responder)).toList(),
5. ...

Resumo do que essa lina de comando faz:

* Pegou as respostas, que é uma Lista de Strings: respostas
* Usou o map, para converter a Lista de Strings em uma Lista de Widgets: .map((texto) => Respostas(texto, \_responder)) já que está dentro de uma Árvore de Componentes, então, o que será passado para o Column serão Widgets.
* E uma vez convertido para ***Widgets***, foi transformado o resultado do ***map()***em uma ***Lista***: .toList().

**5:12**

**2. Noções básicas de Flutter**

56. Lista para Widgets #02

* Por último,  foi usado o ***Spread ...***para pegar cada um dos elementos da *Lista* e coloca dentro da Lista dos ***Filhos "children"*** do ***Componete Column.***

**0:33**

**2. Noções básicas de Flutter**

58. Evitando Acessar Pergunta Inválida

Extraindo a Lista de ***Perguntas***e ***Respostas***de dentro do *Método build*, transformando-a em privado \_perguntas e em constante const

1. import 'package:flutter/material.dart';
2. import './questao.dart';
3. import './resposta.dart';
5. main() => runApp(PerguntaApp());
7. class \_PerguntaAppState extends State<PerguntaApp> {
8. var \_perguntaSelecionada = 0;
9. //   Cria a Lista de Perguntas e Respostas
10. final \_perguntas = const [
11. {
12. 'texto': 'Qual é a sua cor favorita?',
13. 'resposta': ['Preto', 'vermelho', 'verde', 'branco']
14. },
15. {
16. 'texto': 'Qual é o seu animal favorito?',
17. 'resposta': ['Coelho', 'Cobra', 'Elefante', 'Leão']
18. },
19. {
20. 'texto': 'Qual é o seu instrutor favorito?',
21. 'resposta': ['Maria', 'João', 'Leo', 'Pedro']
22. },
23. ];
24. ...

**1:33**

**2. Noções básicas de Flutter**

58. Evitando Acessar Pergunta Inválida

***TRATANDO O ERRO INDICE INVÁLIDO***

Criando um ***Método Getter*** bool get temPerguntaSelecionada{}  se tenha pergunta selecionada será mostrada a pergunta, caso contrário, a pergunta não será mostrada.

1. bool get temPerguntaSelecionada{
2. // Se o índice for menor que o tamanho da lista
3. // Significa que tem uma Pergunta Selecionada
4. return \_perguntaSelecionada < \_perguntas.length
5. }

**2:32**

**2. Noções básicas de Flutter**

58. Evitando Acessar Pergunta Inválida

***CRIANDO UMA ATRIBUIÇÃO CONDICIONAL***

Se tiver ***Pergunta Selecionada*** temPergutaSelecionada  ***então***?  mostra a ***Lista de Respostas*** \_perguntas[\_perguntaSelecionada]['respostas']  ***caso contrário*** :  não mostra nada [] .

1. List<String> respostas = temPergutaSelecionada
2. ? \_perguntas[\_perguntaSelecionada]['respostas']
3. : [];

**3:01**

**2. Noções básicas de Flutter**

58. Evitando Acessar Pergunta Inválida

***CRIANDO UMA ATRIBUIÇÃO CONDICIONAL AO BODY***

O ***body*** recebe a *Coluna*Column e para corrigir isso será atribuido body uma condição.

Se o ***body***tiver uma ***Pergunta Selecionada*** temPerguntaSelecionada , ***então***, ?  mostra a Coluna Column  ***caso contrário*** :  não mostra nada [].

**4:22**

**2. Noções básicas de Flutter**

58. Evitando Acessar Pergunta Inválida

Usar o ***Getter***temPergutaSelecionada  para evitar ***mudanças de estado*** desnecessárias:

1. void \_responder() {
2. if(temPergutaSelecionada){
3. setState(() {
4. \_perguntaSelecionada++;
5. });
6. }
7. }

**0:05**

**2. Noções básicas de Flutter**

64. Widget Questionário

***CRIANDO O COMPONENTE QUESTIONÁRIO***

Criar um novo arquivo chamado ***questionario.dart***

Criar a ***Classe StatelessWidget Questionario***e dentro do build copiar o Componente Column sendo assim refatorando o codigo.

E por ser uma Classe ***StatelessWidget***as variáveis são do *Tipo*final.

1. class Questionario extends StatelessWidget {
2. final List<Map<String, Object>> perguntas;
3. final int perguntaSelecionada;
4. final void Function() quandoResponder;
6. @override
7. Widget build(BuildContext context) {
8. return Column(children: [
9. Questao(perguntas[perguntaSelecionada]['texto'].toString()),
10. ...respostas.map((t) => Resposta(t, quandoResponder)).toList(),
11. ]);
12. }
13. }

**1:52**

**2. Noções básicas de Flutter**

64. Widget Questionário

Definir o Construtor da Classe usando parâmetros nomeados.

1. const Questionario({
2. @required this.perguntas,
3. @required this.perguntaSelecionada,
4. @required this.quandoResponder,
5. });

**3:00**

**2. Noções básicas de Flutter**

64. Widget Questionário

As respostas também estarão dentro do build do Questionario.

1. ...
2. @override
3. Widget build(BuildContext context) {
4. // Cria uma Lista contendo as respostas da Pergunta
5. // Se respostas tiver Pergunta Selecionada,
6. // mostra a Lista de Respostas
7. // caso contrário Não mostra nada.
8. List<String> respostas = temPerguntaSelecionada
9. ? perguntas[perguntaSelecionada].cast()['respostas']
10. : [];
11. ...

**3:08**

**2. Noções básicas de Flutter**

64. Widget Questionário

O ***Getter***temPerguntaSelecionada também precisa estar dentro da ***Classe Questionario***

1. // Método Getter
2. bool get temPerguntaSelecionada {
3. // Se o índice for menor que o tamanho da lista
4. // Significa que tem uma Pergunta Selecionada
5. return perguntaSelecionada < perguntas.length;
6. }

**3:25**

**2. Noções básicas de Flutter**

64. Widget Questionário

Também será preciso importar Questao e Resposta além do Material.

1. import 'package:flutter/material.dart';
2. import './questao.dart';
3. import './resposta.dart';

**4:41**

**2. Noções básicas de Flutter**

64. Widget Questionário

No arquivo ***main.dart*** será preciso importar o questionário.

1. import './questionario.dart';

**5:01**

**2. Noções básicas de Flutter**

64. Widget Questionário

E agora no ***body***será colocado o Questionario() juntamente com seus parâmetros.

1. ...
2. body: temPerguntaSelecionada
3. ? Questionario(
4. perguntas: \_perguntas,
5. perguntaSelecionada: \_perguntaSelecionada,
6. quandoResponder: \_responder,
7. )
8. : Resultado('Parabéns!!!'),
9. ...

**0:05**

**2. Noções básicas de Flutter**

66. Adicionando Pontuação #01

***ADICIONANDO PONTUAÇÃO***

Dentro do ***main.dart*** estão as ***Respostas*** e a cada uma das respostas, estará uma ***nota***.

1. class \_PerguntaAppState extends State<PerguntaApp> {
2. var \_perguntaSelecionada = 0;
3. //   Cria a Lista de Perguntas e Respostas
4. final \_perguntas = const [
5. {
6. 'texto': 'Qual é a sua cor favorita?',
7. 'respostas': [
8. {'texto': 'Preto', 'nota': 10},
9. {'texto': 'vermelho', 'nota': 5},
10. {'texto': 'verde', 'nota': 3},
11. {'texto': 'branco', 'nota': 1},
12. ]
13. },
14. ---

**0:22**

**2. Noções básicas de Flutter**

66. Adicionando Pontuação #01

1. ---
2. {
3. 'texto': 'Qual é o seu animal favorito?',
4. 'respostas': [
5. {'texto': 'Coelho', 'nota': 10},
6. {'texto': 'Cobra', 'nota': 5},
7. {'texto': 'Elefante', 'nota': 3},
8. {'texto': 'Leão', 'nota': 1},
9. ]
10. },
11. {
12. 'texto': 'Qual é o seu instrutor favorito?',
13. 'respostas': [
14. {'texto': 'Leo', 'nota': 10},
15. {'texto': 'Maria', 'nota': 5},
16. {'texto': 'João', 'nota': 3},
17. {'texto': 'Pedro', 'nota': 1},
18. ]
19. },
20. ];

**2:46**

**2. Noções básicas de Flutter**

66. Adicionando Pontuação #01

E dentro de ***questionario.dart*** quando for mapear as respostas, não será mais uma ***String*** e sim List<Map<String, Object>>  é um Object porque em main.dart, vão ter situações em que terá uma String como ***valor***e outras situações que terá um ***inteiro***int como ***valor***.

Uma vez recebendo as respostas vai estar percorrendo as ***respostas***respostas.map((resp) e estará pegando a ***resposta*** texto => Respostas(resp['texto'] e depois convertendo em ***String***.toString(),

Revisando:

No build do Questionario, está pegando uma Lista de Maps List<Map<String, Object>> está pegando cada uma das Listas de resposta:

1. 'respostas': [
2. {'texto': 'Preto', 'nota': 10},
3. {'texto': 'vermelho', 'nota': 5},
4. {'texto': 'verde', 'nota': 3},
5. {'texto': 'branco', 'nota': 1},
6. ]
7. },

**4:02**

**2. Noções básicas de Flutter**

66. Adicionando Pontuação #01

Dentro do  .map((resp) ***chave : valor***, usando a ***função Map*** para transformar isso em ***Componente do Tipo*** Resposta.

Como as ***respostas***(resp é cada um dos elementos que tem ***texto***e ***nota***neste caso queremos o ***valor textual*** (resp['texto'].toString .

***Código Completo:***

1. ...respostas
2. .map((resp) => Resposta(resp['texto'].toString(), quandoResponder))
3. .toList(),

**0:01**

**2. Noções básicas de Flutter**

68. Adicionando Pontuação #02

***ADICIONANDO PONTUAÇÃO***

Modificar a *chave*'nota' por 'pontuacao' para facilitar o entendimento.

Quando o usuário for responder, nesse ponto, se sabe quanto vale *resposta do usuário*, no momento que *houver o clique*, precisa passar o valor da resposta para o ***Método main***.

O ***Método***\_responder() vai receber por parâmetro a *pontuação do usuário*, \_responder(int pontuacao);.  
Também será criado de forma global, a variável pontuação total: var \_pontuacaoTotal = 0;

**2:29**

**2. Noções básicas de Flutter**

68. Adicionando Pontuação #02

Será passado um ***Método Anônimo*** ()  que irá ***chamar***=> o ***Método***quandoResponder passando a ***resposta***resp acessando o ***Valor***['pontuacao'] . ***Ex.:***() => qandoResponder(resp['pontuacao'])

**3:09**

**2. Noções básicas de Flutter**

68. Adicionando Pontuação #02

Convertendo ***Arrow Function*** em ***Block Body*** para o código ter uma leitura melhor.

1. ...
2. ...respostas.map((resp) {
3. return Resposta(
4. resp['texto'].toString(),
5. () => quandoResponder(resp['pontuacao']),
6. );
7. }).toList(),
8. ]);
9. ...

Desta forma, tense a Resposta( passando o valor textual e convertendo para ***String***resp['texto'].toString(),  e passando a ***Função Anonima*** () => quandoResponder(resp['pontuacao']),  e a Resposta está sendo retornada através do ***Método Map*** respostas.map((resp)  e que depois é convertido para uma ***Lista***.toList() .

**0:04**

**2. Noções básicas de Flutter**

69. Adicionando Pontuação #03

***Modificando a tela de Resultado, com resultados condicionais.***

Em Resultado será criada a variável pontuação final int pontuacao; que guardará o valor do Resultado .

Também será criado o Construtor da Classe que receberá por parâmetro o valor da variável: Resultado(this.pontuacao);

No ***Componente Main*** o *Componente* Resultado espera um parâmetro que é a pontuação, então, ele receberá a ***Pontação Total***: Resuñtado(\_pontuacaoTotal);

Voltando ao Componente Resultado, será criado um Getter que mostrará a frase de forma condicional.

1. String get fraseResposta {
2. if (pontuacao < 8) {
3. return 'Parabéns!';
4. } else if (pontuacao < 12) {
5. return 'Você é bom!';
6. } else if (pontuacao < 16) {
7. return 'Impressionante!!!';
8. } else {
9. return 'Nível jedi!!!!!';
10. }

**0:02**

**2. Noções básicas de Flutter**

70. Reiniciando Questionário

***Reiniciando o Questionário***

Será criado um botão para que possa voltar para o inicio do questionário com o valor das variáveis zeradas.

Uma das questões importantes com relação a comunicação entre os ***Componentes***é a *Comunicação Direta* e *Comunicação Indireta*.

***Comunicação Direta:*** é quando o *Componente Pai* passa ***parâmetros***para o *Componente Filho*.

***Comunicação Indireta:*** è quando o *Componente Pai* passa uma ***Função***para o *Componente Filho* para que o *Componente Filho* chame essa ***Função***e então o ***Componente Pai*** seja notificado*.*

**0:50**

**2. Noções básicas de Flutter**

70. Reiniciando Questionário

***Exemplo Comunicação Direta:***

O ***Componente Pai***, PerguntaApp passou um parâmetro \_pontuacaoTotal para o ***Componente Filho*** Resultado mostrou um determinado **Valor**.

No ***Componente Pai***, Questionario foram passados como parâmetro: \_perguntas, \_perguntaSelecionada e também foi passado o *Método*, que será chamado em algum momento e dessa forma, o ***Componente filho*** pode chamar um *Método*passando a informação para o *Componente Pai*.

**2:03**

**2. Noções básicas de Flutter**

70. Reiniciando Questionário

Exatamente através desse ***Método***\_responder, que está recebendo do *Componente Filho* o ***Atributo***int pontuacao e a ***Comunicação Indireta*** diz:  que será mandado uma *Função*para o ***Componente Filho***, o *Componente Filho* irá chamar essa *Função*em algum momento e mandar ***Dados***ao *Componente Pai* a partir dos parâmetro dessa ***Função***\_responder.

1. void \_responder(int pontuacao) {
2. //  Evita mudanças de estado desnecessárias.
3. if (temPerguntaSelecionada) {
4. setState(() {
5. \_perguntaSelecionada++;
6. // Soma o valor do parâmetro pontuacao com o valor da
7. //Variável Global \_pontuacaoTotal .
8. \_pontuacaoTotal += pontuacao;
9. });
10. }

**2:17**

**2. Noções básicas de Flutter**

70. Reiniciando Questionário

**este de mensagem**

***Exemplo Comunicação Indireta:***

Será criado um *Método*\_reiniciarQuestionario(){},  dentro, será chamado o setState((){}) e dentro irá reiniciar as variáveis: \_pergunaSelecionada = 0; e \_pontuacaoTotal = 0;

Com isso esses dois valores foram reinicializados para seus valores inicial.

1. main.dart
2. ...
3. // Os dois valores foram reinicializados para seus valores inicial.
4. void \_reiniciarQuestionario() {
5. setState(() {
6. \_perguntaSelecionada = 0;
7. \_pontuacaoTotal = 0;
8. });
9. }
10. ...

**2:45**

**2. Noções básicas de Flutter**

70. Reiniciando Questionário

Agora a Função \_reinicarQuestionario será passada como parâmetro para o Componente Resultado.

No Componente Resultado será adicionado mais um parâmetro que será um final Função Function() não recebe parametro e que ñ retorna nada void : ***Ex.:*** final void Function() quandoReinicarQuestionario; .

No Construtor será acrescentado mais um parâmetro this.quandoReinicarQuestionario . ***Ex:*** Resultado(this.pontacao, this.quandoReinicarQuestionario); .

1. resultado.dart
3. ...
4. class Resultado extends StatelessWidget {
5. final int pontuacao;
6. final void Function() quandoReiniciarQuestionario;
8. Resultado(this.pontuacao, this.quandoReiniciarQuestionario);
9. ...

**2:48**

**2. Noções básicas de Flutter**

70. Reiniciando Questionário

A Função \_reinicarQuestionario será chamada quando o botão for acionado, que será criado à seguir.

Essa Função \_reinicarQuestionario será adicionada como parâmetro ao ***Componente Resultado*** em ***Main***, pois é obrigatório.

1. main.dart
3. ...
5. body: temPerguntaSelecionada
6. ? Questionario(
7. perguntas: \_perguntas,
8. perguntaSelecionada: \_perguntaSelecionada,
9. quandoResponder: \_responder,
10. )
11. : Resultado(\_pontuacaoTotal, \_reiniciarQuestionario),
13. ...

**3:53**

**2. Noções básicas de Flutter**

70. Reiniciando Questionário

Agora, dentro de ***Resultado***dentro de build, o ***Widget Center*** será envolvido por uma coluna wrap with Column  e a coluna será centralizada com mainAxisAligment: mainAxisAligment.center, .

Agora que o Componente tem mais de um Elemento, mais de um filho, depois de Center pode se colocar mais um Elemento.

Será adicionado um FlatButton:

1. resultado.dart
3. ...
5. @override
6. Widget build(BuildContext context) {
7. return Column(
8. mainAxisAlignment: MainAxisAlignment.center,
9. children: [
10. Center(
11. child: Text(
12. fraseResposta,
13. style: const TextStyle(fontSize: 28),
14. )),
15. TextButton(
16. onPressed: quandoReiniciarQuestionario,
17. child: const Text(
18. 'Reiniciar?',
19. style: TextStyle(
20. fontSize: 20,
21. color: Colors.deepOrange,
22. ),
23. ),
24. ),
25. ],
26. );
27. }
29. ...

**6:09**

**2. Noções básicas de Flutter**

70. Reiniciando Questionário

No onPressed, ***Ex.:***onPressed: quandoReiniciarQuestionario, a ***Função***quandoReinicarQuestionario é chamada diretamente por não possuir parâmetro, por esse motivo, ela não precisa ser chamada através de uma ***Função Arrow*** () =>

**7:19**

**2. Noções básicas de Flutter**

70. Reiniciando Questionário

***Conclusão:***

Agora é possível voltar e reiniciar o formulário quantas vezes forem necessários, nesse caso foi feita uma ***Comunicação Indireta***, foi passado uma Função de Comunicação Direta para o Componente Filho (o componente pai passando par o componente filho) e no Componente Filho a Função foi invocada de tal forma que conseguiu chamar o comportamento no Pai.

Nesse caso, não foi passado nenhum parâmetro mas poderia ter sido passado.

**7:20**

**2. Noções básicas de Flutter**

70. Reiniciando Questionário

***Código do arquivo Main:***

1. // ignore\_for\_file: use\_key\_in\_widget\_constructors
3. import 'package:flutter/material.dart';
4. import './questionario.dart';
5. import './resultado.dart';
7. main() => runApp(PerguntaApp());
9. class \_PerguntaAppState extends State<PerguntaApp> {
10. var \_perguntaSelecionada = 0;
11. var \_pontuacaoTotal = 0;
13. //   Cria a Lista de Perguntas e Respostas
14. final \_perguntas = const [
15. {
16. 'texto': 'Qual é a sua cor favorita?',
17. 'respostas': [
18. {'texto': 'Preto', 'pontuacao': 10},
19. {'texto': 'vermelho', 'pontuacao': 5},
20. {'texto': 'verde', 'pontuacao': 3},
21. {'texto': 'branco', 'pontuacao': 1},
22. ]
23. },
24. ...

**7:23**

**2. Noções básicas de Flutter**

70. Reiniciando Questionário

1. {
2. 'texto': 'Qual é o seu animal favorito?',
3. 'respostas': [
4. {'texto': 'Coelho', 'pontuacao': 10},
5. {'texto': 'Cobra', 'pontuacao': 5},
6. {'texto': 'Elefante', 'pontuacao': 3},
7. {'texto': 'Leão', 'pontuacao': 1},
8. ]
9. },
10. {
11. 'texto': 'Qual é o seu instrutor favorito?',
12. 'respostas': [
13. {'texto': 'Leo', 'pontuacao': 10},
14. {'texto': 'Maria', 'pontuacao': 5},
15. {'texto': 'João', 'pontuacao': 3},
16. {'texto': 'Pedro', 'pontuacao': 1},
17. ]
18. },
19. ];
21. ...

**7:26**

**2. Noções básicas de Flutter**

70. Reiniciando Questionário

1. ...
2. // O Método \_responder() vai receber por parâmetro a pontuação do usuário.
3. void \_responder(int pontuacao) {
4. //  Evita mudanças de estado desnecessárias.
5. if (temPerguntaSelecionada) {
6. setState(() {
7. \_perguntaSelecionada++;
8. // Soma o valor do parâmetro pontuacao com o valor da
9. //Variável Global \_pontuacaoTotal .
10. \_pontuacaoTotal += pontuacao;
11. });
12. }
14. // ignore: avoid\_print
15. print(\_perguntaSelecionada);
16. print(\_pontuacaoTotal);
17. }
19. // Os dois valores foram reinicializados para seus valores inicial.
20. void \_reiniciarQuestionario() {
21. setState(() {
22. \_perguntaSelecionada = 0;
23. \_pontuacaoTotal = 0;
24. });
25. }
27. ...

**7:28**

**2. Noções básicas de Flutter**

70. Reiniciando Questionário

1. ...
2. // Método Getter
3. bool get temPerguntaSelecionada {
4. // Se o índice for menor que o tamanho da lista
5. // Significa que tem uma Pergunta Selecionada
6. return \_perguntaSelecionada < \_perguntas.length;
7. }
9. @override
10. Widget build(BuildContext context) {
11. return MaterialApp(
12. home: Scaffold(
13. appBar: AppBar(
14. title: const Text('Perguntas'),
15. ),
16. // Se body tiver uma Pergunta Selecionada
17. // Mostra a Column, caso contrário Não mostra nada.
18. ...

**7:30**

**2. Noções básicas de Flutter**

70. Reiniciando Questionário

1. ...
2. body: temPerguntaSelecionada
3. ? Questionario(
4. perguntas: \_perguntas,
5. perguntaSelecionada: \_perguntaSelecionada,
6. quandoResponder: \_responder,
7. )
8. : Resultado(\_pontuacaoTotal, \_reiniciarQuestionario),
9. ),
10. );
11. }
12. }
14. class PerguntaApp extends StatefulWidget {
15. @override
16. \_PerguntaAppState createState() {
17. return \_PerguntaAppState();
18. }
19. }

**7:32**

**2. Noções básicas de Flutter**

70. Reiniciando Questionário

***Código do arquivo Resultado:***

1. import 'dart:ui';
3. import 'package:flutter/material.dart';
5. class Resultado extends StatelessWidget {
6. final int pontuacao;
7. final void Function() quandoReiniciarQuestionario;
9. Resultado(this.pontuacao, this.quandoReiniciarQuestionario);
11. String get fraseResposta {
12. if (pontuacao < 8) {
13. return 'Parabéns!';
14. } else if (pontuacao < 12) {
15. return 'Você é bom!';
16. } else if (pontuacao < 16) {
17. return 'Impressionante!!!';
18. } else {
19. return 'Nível Jedi!!!!!';
20. }
21. }
23. ...

**7:34**

**2. Noções básicas de Flutter**

70. Reiniciando Questionário

1. ...
2. @override
3. Widget build(BuildContext context) {
4. return Column(
5. mainAxisAlignment: MainAxisAlignment.center,
6. children: [
7. Center(
8. child: Text(
9. fraseResposta,
10. style: const TextStyle(fontSize: 28),
11. )),
12. TextButton(
13. onPressed: quandoReiniciarQuestionario,
14. child: const Text('Reiniciar?'),
15. ),
16. ],
17. );
18. }
19. }