



Componentes de Interface

Parte 4

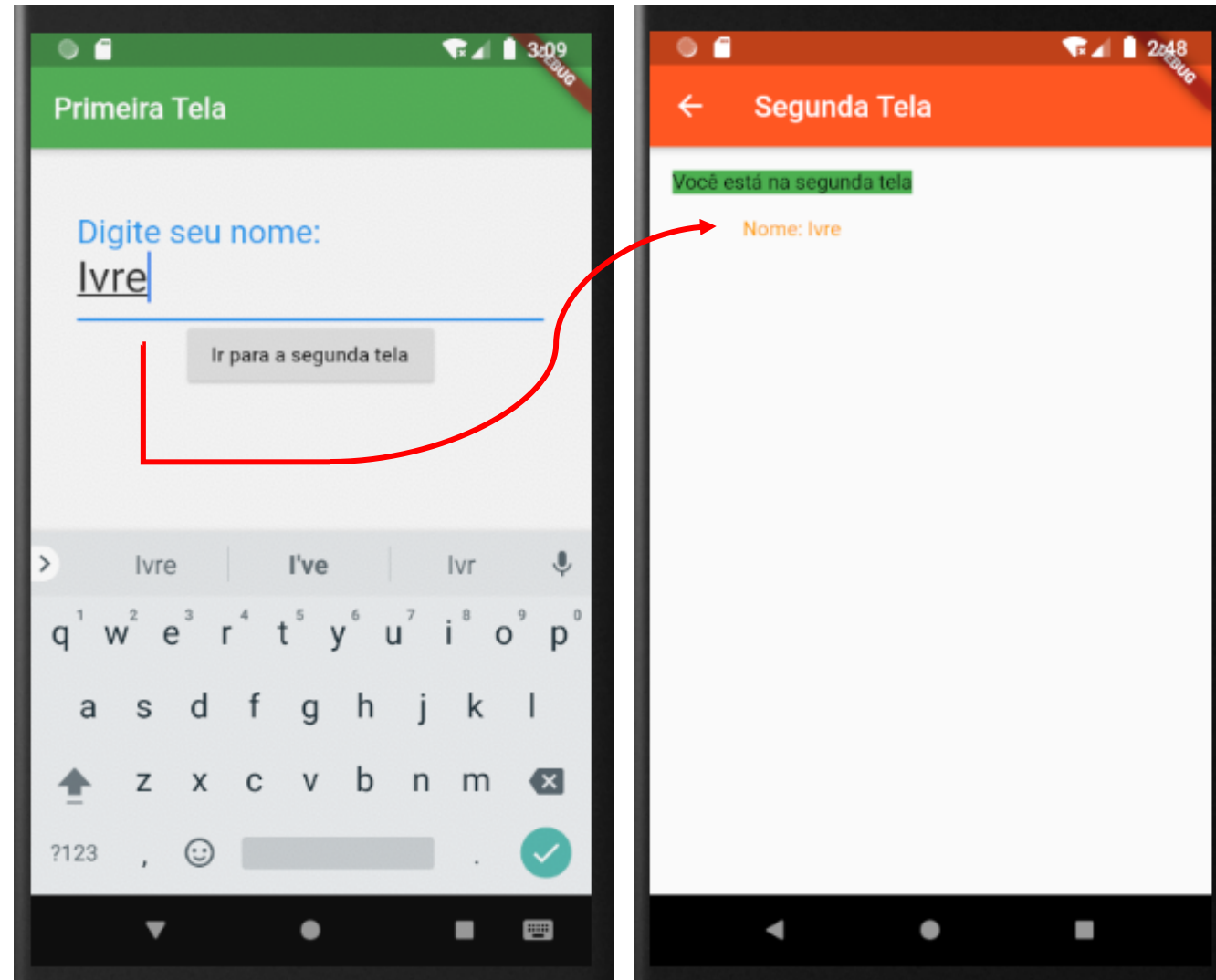
Profa. Ilo Rivero
(ilo@pucminas.br)

O que vamos aprender nessa aula?

- Dados entre telas
- Nomear rotas
- Layout
- List view
- AlertDialog

Passar dados entre telas

- Dando continuidade no exemplo anterior de navegação entre telas, vamos agora passar um valor de uma tela para a outra
- Observe que o nome digitado na **Primeira Tela** é mostrado na Segunda Tela



Passar dados entre telas

Segunda Tela:

```
class SegundaTela extends StatefulWidget {  
  String valor;  
  SegundaTela({this.valor});  
  @override  
  _SegundaTelaState createState() => _SegundaTelaState();  
}
```

Construtor na classe Segunda Tela que recebe um parâmetro opcional {}

Primeira Tela:

```
onPressed: (){  
  Navigator.push(  
    context,  
    MaterialPageRoute(  
      builder: (context) => SegundaTela(valor: _textEditingController.text)  
    ), // MaterialPageRoute  
  );  
}
```

Ao chamar a classe da Segunda Tela, é possível passar como parâmetro o valor digitado no campo do Text Field

Passar dados entre telas

- Para acessar o valor recebido no construtor da classe Segunda Tela na classe `_SegundaTelaState()`, vamos usar o objeto `widget` para recuperar o valor

```
body: Container(  
  padding: EdgeInsets.all(16),  
  child: Column(  
    mainAxisAlignment: MainAxisAlignment.start,  
    children: <Widget>[  
      Text("Você está na segunda tela",  
        style: TextStyle(  
          backgroundColor: Colors.green,  
        )), // TextStyle, Text  
      Text("\nNome: "+ widget.valor,  
        style: TextStyle(  
          color: Colors.orange,  
        )), // TextStyle, Text  
    ], // <Widget>[]  
  ), // Column
```

Acessando o atributo criado na classe SegundaTela

Nomear rotas


- É possível nomear as rotas e usa-las para realizar a navegação entre as telas, vamos usar os seguintes argumentos: **initialRoute** (rota de início) e **routes** (outras rotas)

```
initialRoute: "/",  
routes: {  
    "/segunda": (context) =>  
    SegundaTela(),  
},
```

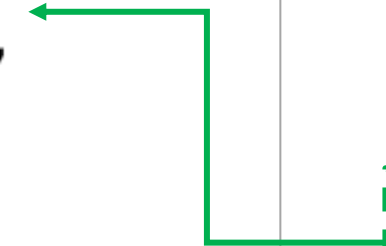
Nomear rotas

```
import 'package:flutter/material.dart';
import 'package:nomeando_rotas_app/SegundaTela.dart';
void main(){
  runApp(MaterialApp(
    initialRoute: "/",
    routes: {
      "/segunda": (context) => SegundaTela(),
    },
    home: PrimeiraTela(),
  )); // MaterialApp
}
```

"/" indica qual é a rota de início



"/segunda" nome dado para a rota, podemos indicar quantas forem necessárias



Nomear rotas

Lembrando que o botão deve ser alterado, pois esse foi depreciado

RaisedButton na Primeira Tela()

```
RaisedButton(  
  child: Text("Ir para a segunda tela"),  
  onPressed: () {  
    Navigator.pushNamed(  
      context,  
      "/segunda",  
    );  
  },  
) , // RaisedButton
```

E para navegar entre as telas, agora vamos usar o método **pushNamed()** do Navigator

E para a rota, agora usamos apenas o nome, que foi criado na função `void main() { }`

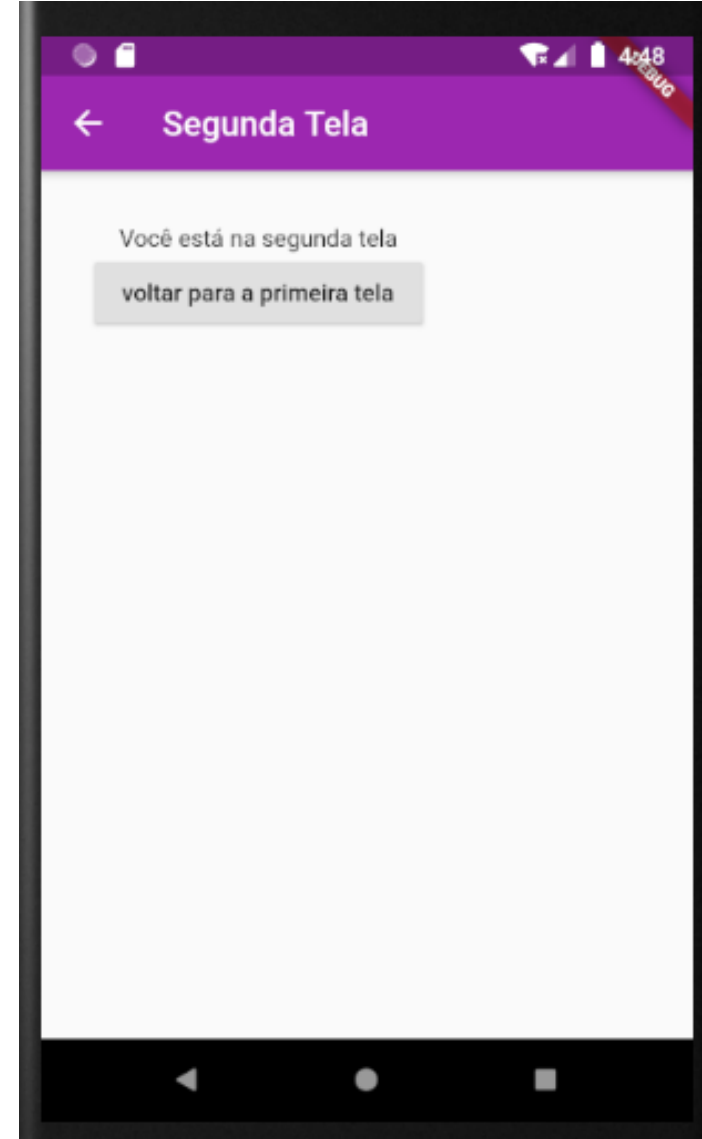
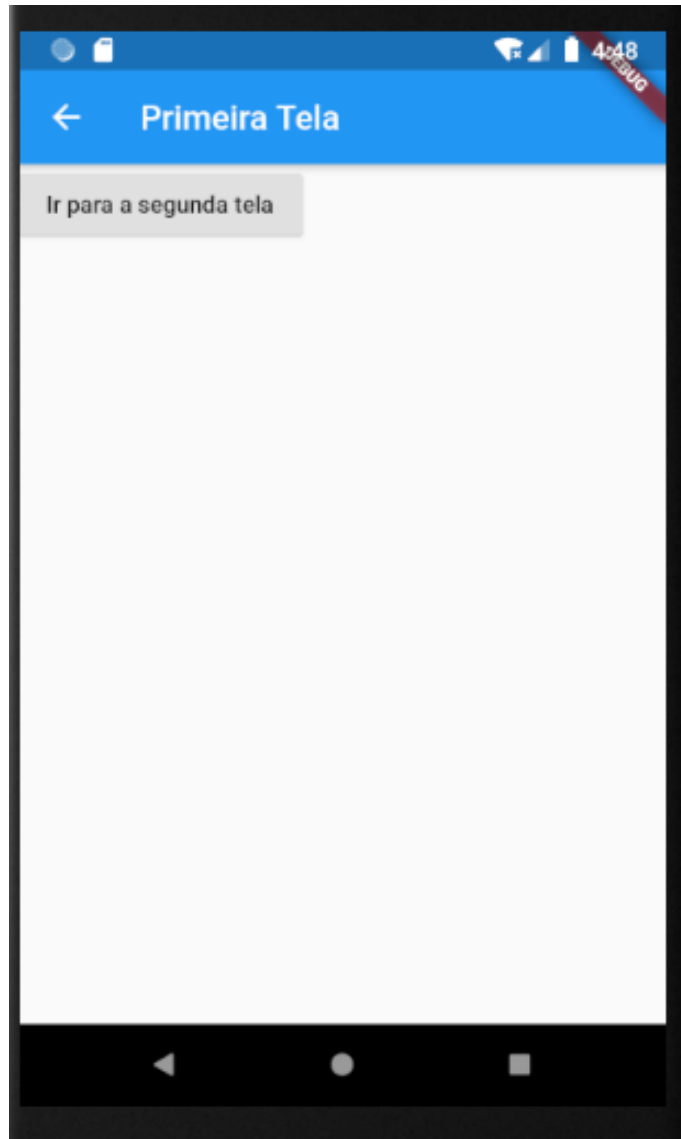
Nomear rotas

ElevatedButton na Segunda Tela()

```
body: Container(  
  padding: EdgeInsets.all(32),  
  child: Column(  
    children: <Widget>[  
      Text("Você está na segunda tela"),  
      ElevatedButton(  
        child: Text("voltar para a primeira tela"),  
        onPressed: (){  
          Navigator.pushNamed(context, "/");  
        },  
      ), // ElevatedButton  
    ], // <Widget>[]  
  ), // Column  
, // Container
```

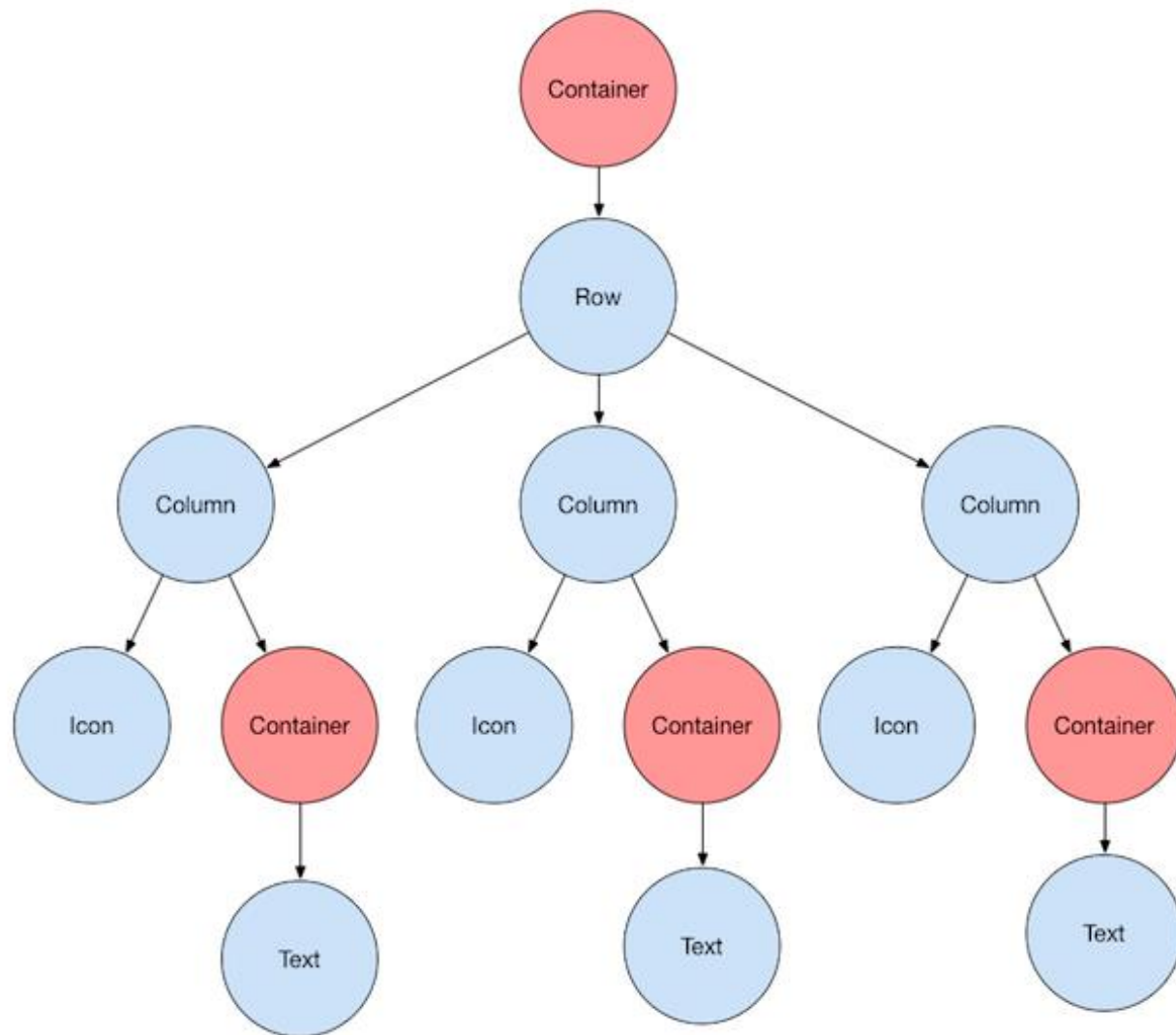
Para voltar para a tela anterior usamos a rota "/" que foi indicada na main()

Nomear rotas

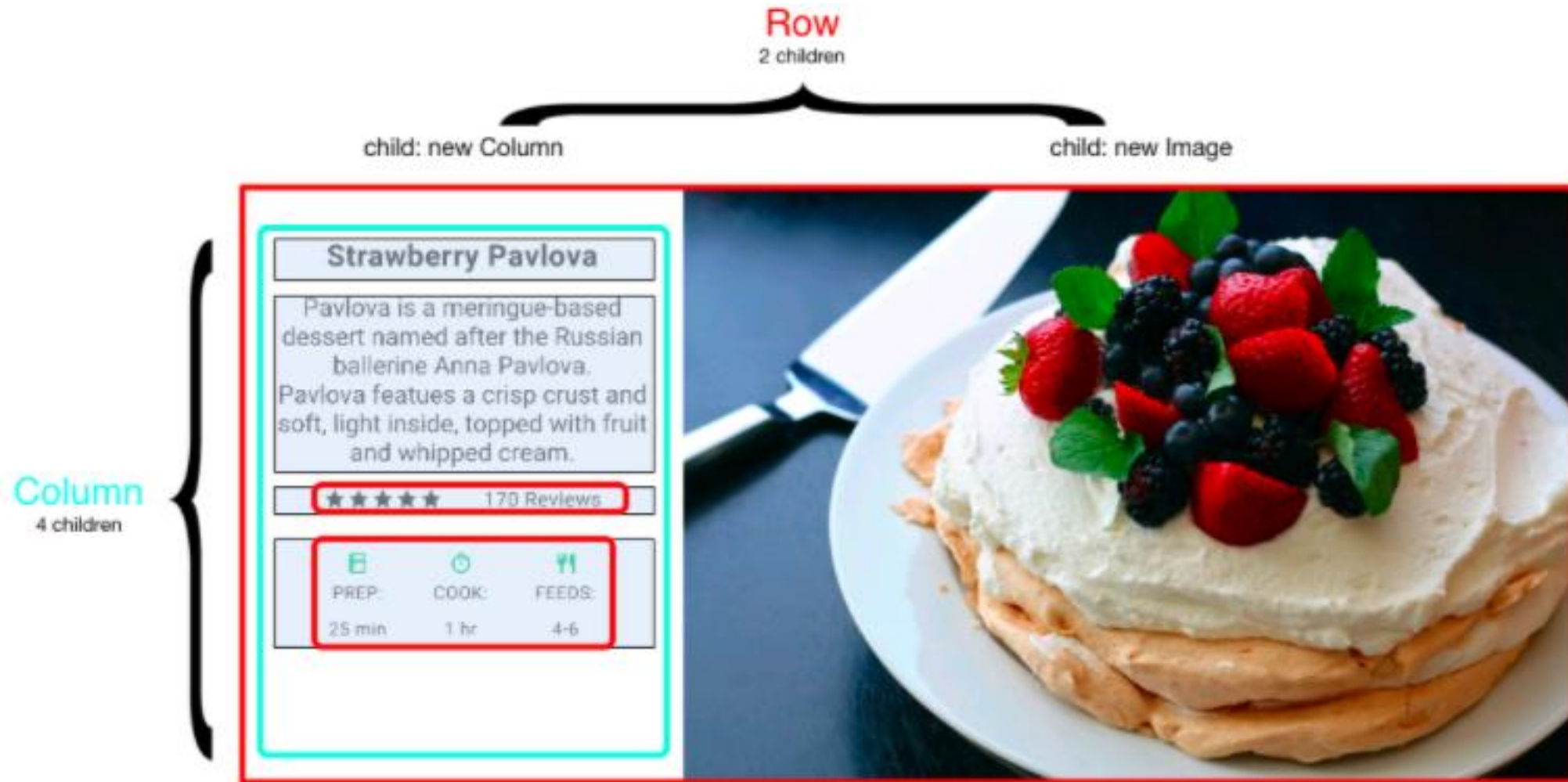


Layout

- O diagrama ao lado representa uma árvore de widgets para montar uma interface de um APP
- É possível construir interfaces com linhas e colunas, umas dentro das outras

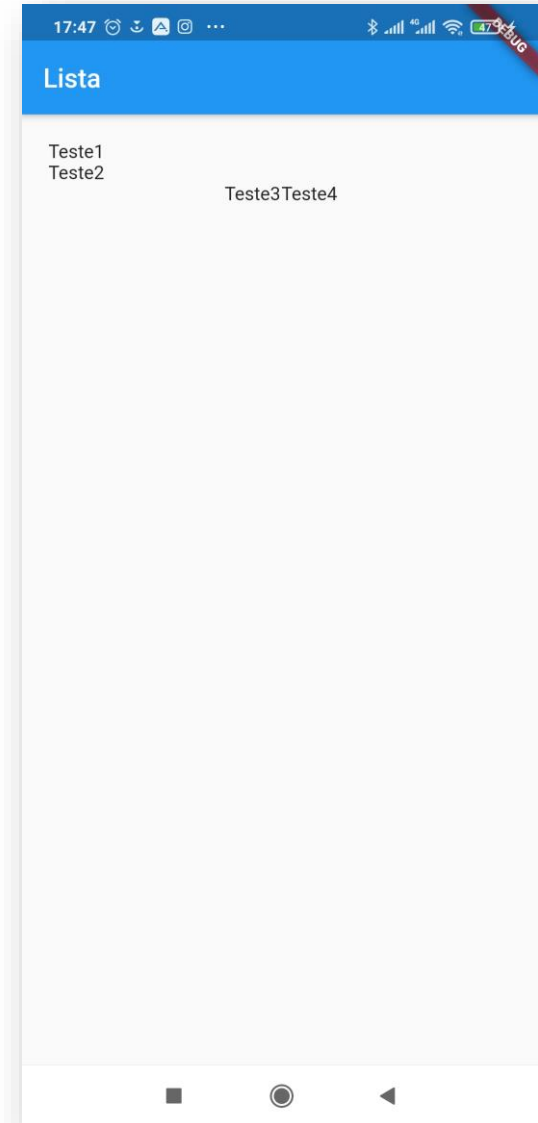


Layout



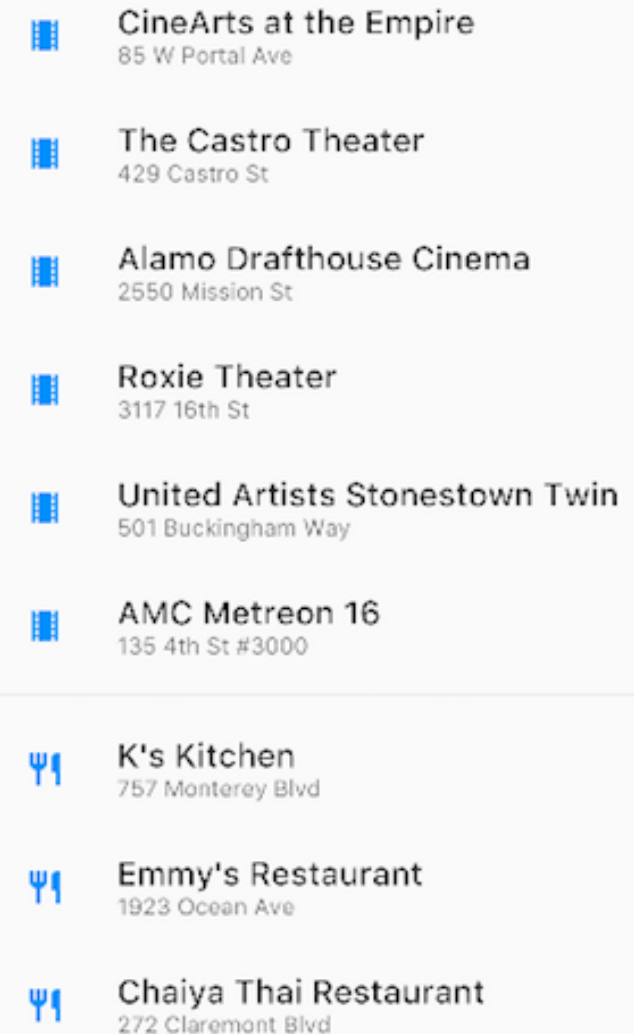
Layout

```
body: Container(  
  width: double.infinity,  
  padding: EdgeInsets.all(20),  
  child: Column(  
    mainAxisAlignment: MainAxisAlignment.start,  
    crossAxisAlignment: CrossAxisAlignment.start,  
    children: <Widget>[  
      Text("Teste1"),  
      Text("Teste2"),  
      Row(  
        mainAxisAlignment: MainAxisAlignment.center,  
        crossAxisAlignment: CrossAxisAlignment.center,  
        children: <Widget>[  
          Text("Teste3"),  
          Text("Teste4"),  
        ], // <Widget>[]  
      ), // Row  
    ], // <Widget>[]  
  ) // Column  
) // Container
```



ListView

- ListView é um widget em forma de coluna, que fornece **rolagem automática** quando seu conteúdo é muito longo para sua caixa de renderização
- Pode ser colocado horizontalmente ou verticalmente



ListView

```
ListView.builder(  
  // Let the ListView know how many items it needs to build.  
  itemCount: items.length,  
  // Provide a builder function. This is where the magic happens.  
  // Convert each item into a widget based on the type of item it is.  
  itemBuilder: (context, index) {  
    final item = items[index];  
  
    return ListTile(  
      title: item.buildTitle(context),  
      subtitle: item.buildSubtitle(context),  
    );  
  },  
);
```

Para converter cada item em um widget, use o construtor `ListView.builder()`. Em geral, forneça uma função de construtor que verifica com qual tipo de item você está lidando e retorne o widget apropriado para esse tipo de item.

ListView

```
body: Container(  
  padding: EdgeInsets.all(20),  
  child: ListView.builder(  
    itemCount: 5,  
    itemBuilder: (context, indice) {  
      print("item ${indice}");  
      return ListTile(  
        title: Text(indice.toString()),  
        subtitle: Text("subtitulo"),  
      ); // ListTile  
    }, // ListView.builder  
  ), // Container
```

O método builder() usado para converter cada item em um widget

O itemCount será usado para indicar quantas vezes será o builder será executado

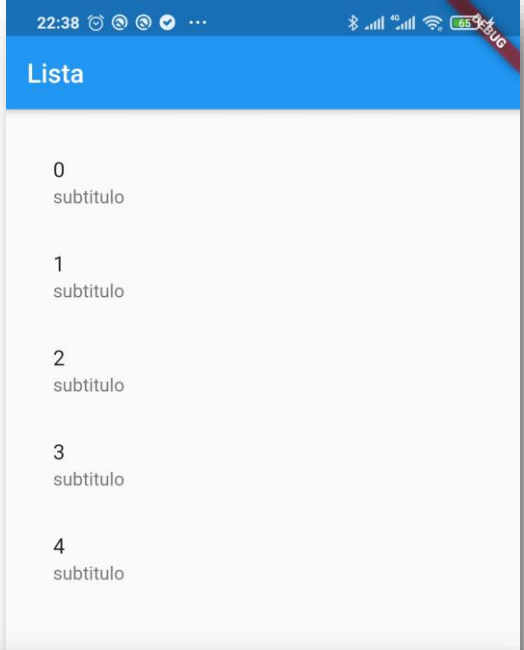
Vamos usar um return com o ListTile() para escrever na tela um title e um subtitle

ListView

Tela:

```
body: Container(  
  padding: EdgeInsets.all(20),  
  child: ListView.builder(  
    itemCount: 5,  
    itemBuilder: (context, indice) {  
      print("item ${indice}");  
      return ListTile(  
        title: Text(indice.toString()),  
        subtitle: Text("subtitulo"),  
      ); // ListTile  
    },  
  ), // ListView.builder  
), // Container
```

Console:



```
oppidx_max 27, oppidx_min 0  
I/flutter (16333): item 0  
I/flutter (16333): item 1  
I/flutter (16333): item 2  
I/flutter (16333): item 3  
I/flutter (16333): item 4
```

ListView

```
class _HomeState extends State<Home> {  
  
  List _items = [];  
  
  void _carregarItens(){  
    _items = [];  
    for(int i=0; i<=10; i++){  
      Map<String, dynamic> item = Map();  
      item["titulo"] = "Titulo ${i} da lista";  
      item["descricao"] = "Descrição ${i} da lista";  
      _items.add(item);  
    }  
  }  
}
```

Método `_carregarItens()`,
será usado para carregar
itens, poderá ser usado
para carregar itens da web
ou de um banco de dados

```
@override  
Widget build(BuildContext context) {  
  _carregarItens();  
  return Scaffold(  
    appBar: AppBar(  
      title: Text("Lista"),  
    ), // AppBar  
    body: Container(...), // Container  
  ); // Scaffold  
}
```

Chamada do método

ListView

```
child: ListView.builder(  
  itemCount: _itens.length,  
  itemBuilder: (context, indice) {  
    Map<String, dynamic> item = _itens[indice];  
    print("item ${item["titulo"]}");  
    return ListTile(  
      title: Text(indice.toString()),  
      subtitle: Text("subtitulo"),  
    ); // ListTile  
  }  
), // ListView.builder
```

_itens[] é uma lista de
itens criada no método
_carregarItens()

```
child: ListView.builder(  
  itemCount: _itens.length,  
  itemBuilder: (context, indice) {  
    //Map<String, dynamic> item = _itens[indice]  
    print("item ${_itens[indice]["titulo"]}");  
    return ListTile(  
      title: Text(indice.toString()),  
      subtitle: Text("subtitulo"),  
    ); // ListTile  
  }  
), // ListView.builder
```

```
Console  
I/flutter (24132): item Titulo 0 da lista  
I/flutter (24132): item Titulo 1 da lista  
I/flutter (24132): item Titulo 2 da lista  
I/flutter (24132): item Titulo 3 da lista  
I/flutter (24132): item Titulo 4 da lista  
I/flutter (24132): item Titulo 5 da lista
```

ListView

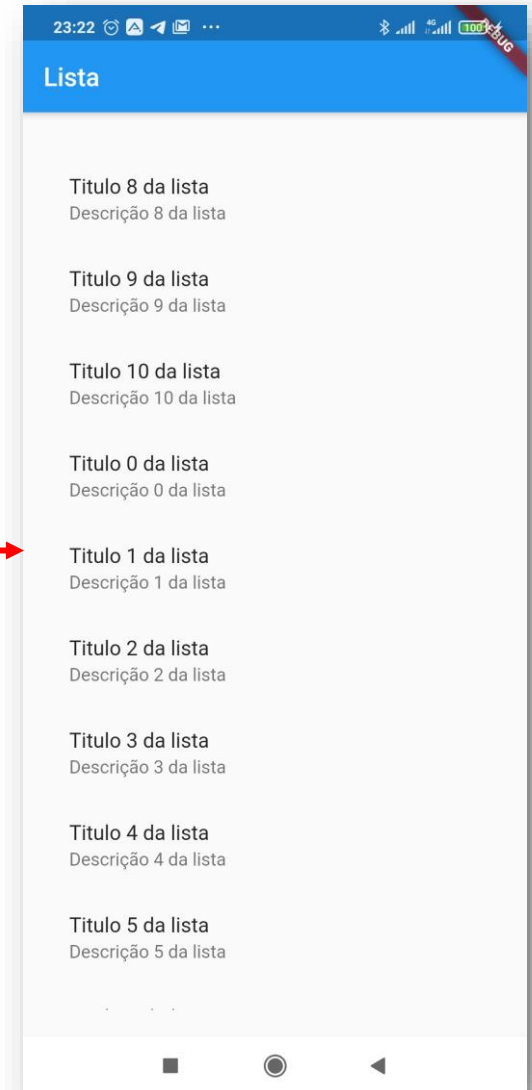
```
body: Container(  
  padding: EdgeInsets.all(20),  
  child: ListView.builder(  
    itemCount: _itens.length,  
    itemBuilder: (context, indice) {  
      return ListTile(  
        title: Text(_itens[indice]["titulo"]),  
        subtitle: Text(_itens[indice]["descricao"]),  
      ); // ListTile  
    },  
  ), // ListView.builder  
, // Container
```



ListView

```
void _carregarItens(){  
    _itens = [];  
    for(int i=0; i<=10; i++){  
        Map<String, dynamic> item = Map();  
        item["titulo"] = "Titulo ${i} da lista";  
        item["descricao"] = "Descrição ${i} da lista";  
        _itens.add(item);  
    }  
}
```

Para não ficar recarregando sempre a lista novamente ao final como mostrado na tela, vamos “zerar” a lista ao chamar o método `_carregarItens()`



ListView - onTap e onLongPress

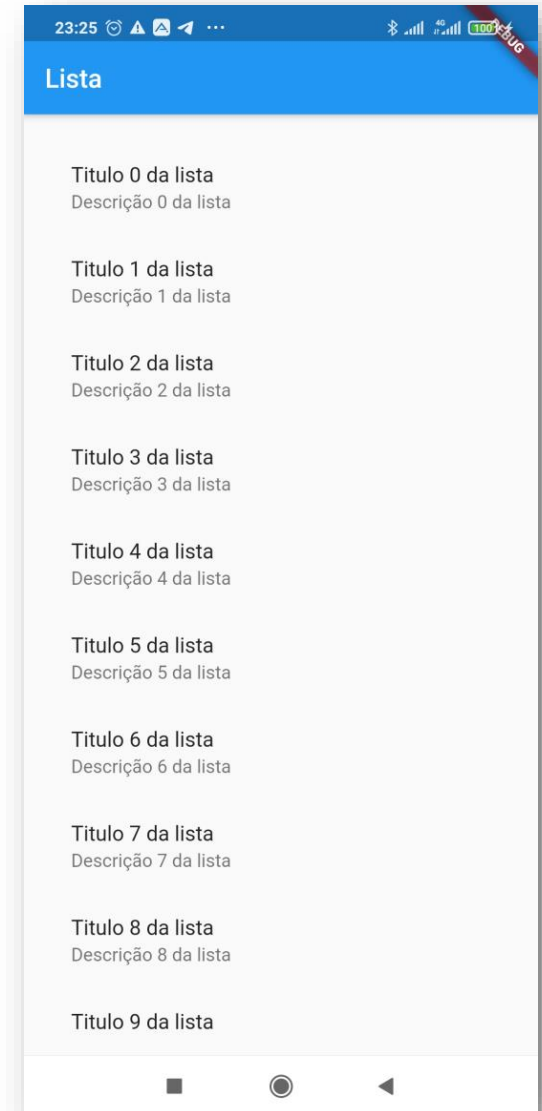
```
return ListTile(  
  onTap: (){  
    print("Clique com onTap");  
  },  
  onLongPress: (){  
    print("Clique com onLongPress");  
  },  
  title: Text(_itens[indice]["titulo"]),  
  subtitle: Text(_itens[indice]["descricao"]),  
); // ListTile
```

onTap: Clique simples no item

onLongPress: Clique e segura o item

ListView - onTap e onLongPress

```
Console
↑
↓
↺
↻
I/flutter (26072): Clique com onTap
I/flutter (26072): Clique com onLongPress
I/flutter (26072): Clique com onTap
I/flutter (26072): Clique com onLongPress
```



ListView - onTap e onLongPress

```
return ListTile(  
  onTap: (){  
    print("Clique com onTap {indice}");  
  },  
  onLongPress: (){  
    print("Clique com onLongPress {indice}");  
  },  
  title: Text(_itens[indice]),  
  subtitle: Text(_itens[indice]),  
); // ListTile
```

Console

I/GED (26072): ged_boost_gpu_freq, level
oppidx_max 27, oppidx_min 0
I/flutter (26072): Clique com onTap 1
I/flutter (26072): Clique com onLongPress 4
I/flutter (26072): Clique com onTap 6

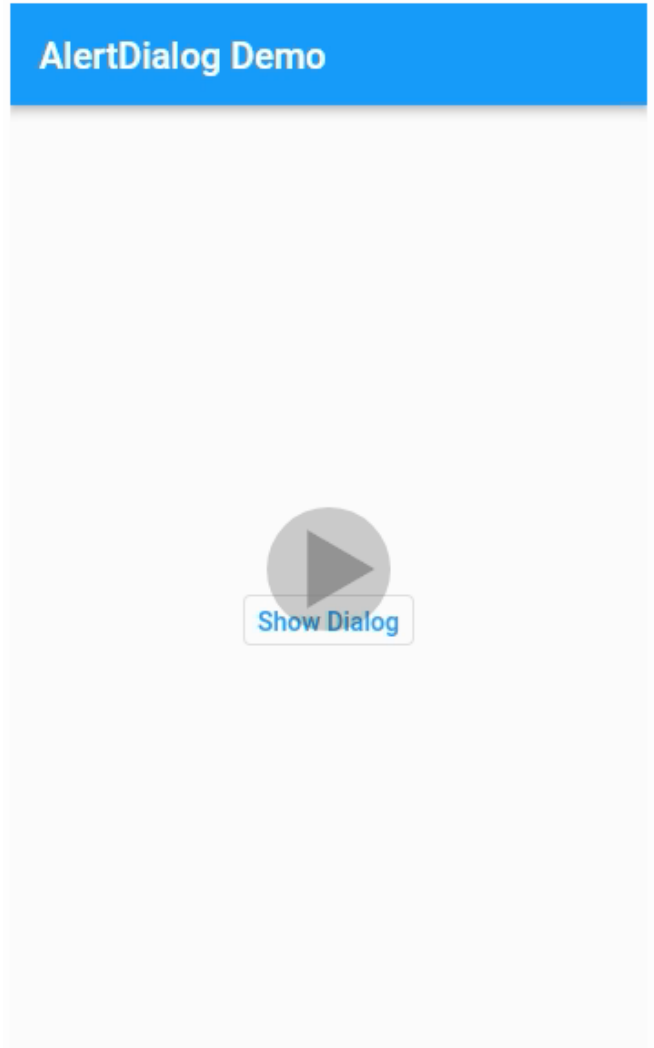
ShowDialog e AlertDialog

- Uma caixa de diálogo de alerta (**AlertDialog**) informa o usuário sobre as situações que requerem reconhecimento
- Um AlertDialog tem um título opcional e uma lista opcional de ações. O título (title) é exibido acima do conteúdo (content) e as ações são exibidas abaixo do conteúdo

```
const AlertDialog({  
  Key key,  
  this.title,  
  this.titlePadding,  
  this.titleTextStyle,  
  this.content,  
  this.contentPadding = const EdgeInsets.fromLTRB(16, 16, 16, 16),  
  this.contentTextStyle,  
  this.actions,  
  this.actionsPadding = EdgeInsets.zero,  
  this.actionsOverflowDirection,  
  this.actionsOverflowButtonSpacing,  
  this.buttonPadding,  
  this.backgroundColor,  
  this.elevation,  
  this.semanticLabel,  
  this.insetPadding = _defaultInsetPadding,  
  this.clipBehavior = Clip.none,  
  this.shape,
```

ShowDialog e AlertDialog

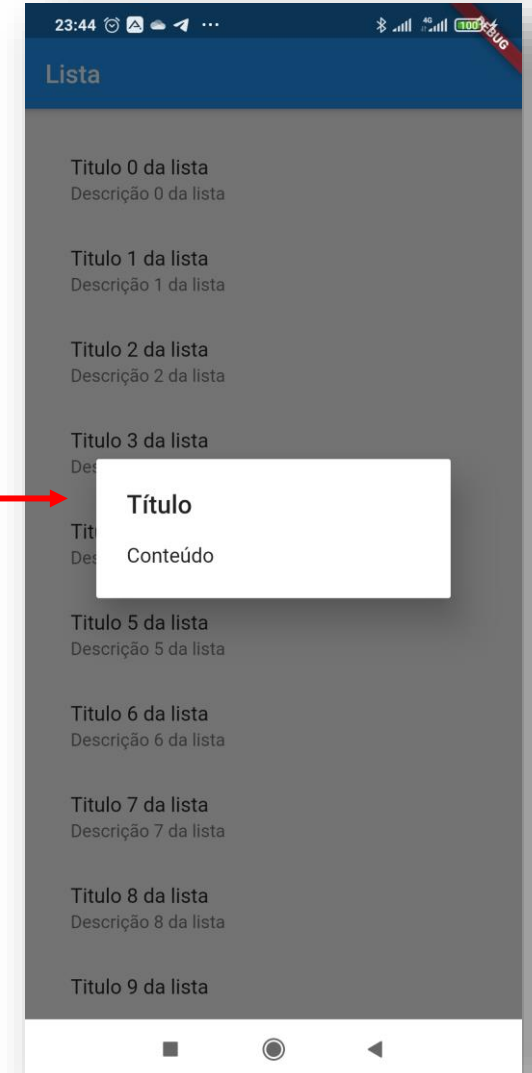
```
Future<void> _showMyDialog() async {  
  return showDialog<void>(  
    context: context,  
    barrierDismissible: false, // user must tap button!  
    builder: (BuildContext context) {  
      return AlertDialog(  
        title: Text('AlertDialog Title'),  
        content: SingleChildScrollView(  
          child: ListBody(  
            children: <Widget>[  
              Text('This is a demo alert dialog.'),  
              Text('Would you like to approve of this message?'),  
            ],  
          ),  
        ),  
        actions: <Widget>[  
          TextButton(  
            child: Text('Approve'),  
            onPressed: () {  
              Navigator.of(context).pop();  
            },  
          ),  
        ],  
      );  
    },  
  );  
}
```



ShowDialog e AlertDialog

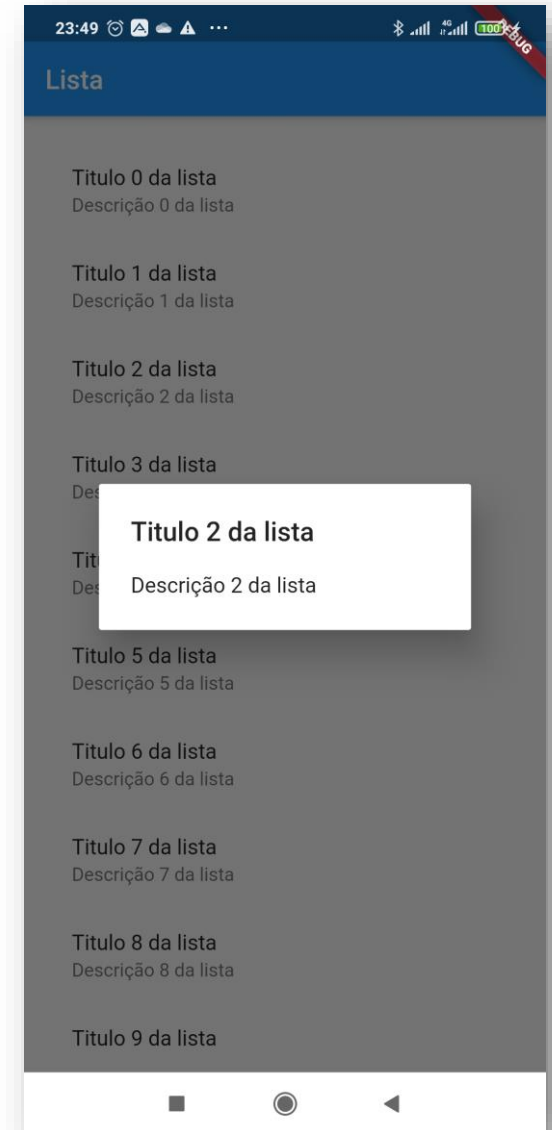
```
return ListTile(  
  onTap: () {  
    //print("Clique com onTap ${indice}");  
    showDialog(  
      context: context,  
      builder: (context) {  
        return AlertDialog(  
          title: Text("Título"),  
          content: Text("Conteúdo"),  
        ); // AlertDialog  
      },  
    );  
  },  
);
```

Observe que o
ShowDialog está
dentro do onTap,
ou seja, ao clicar
em um item da
lista



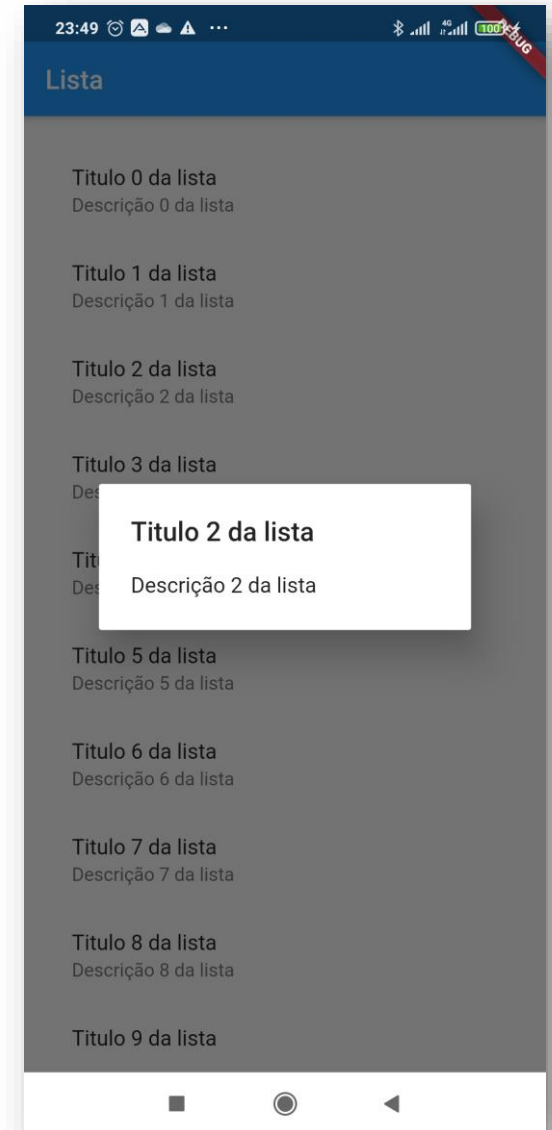
ShowDialog e AlertDialog

```
- return ListTile(  
  onTap: (){  
    //print("Clique com onTap ${indice}");  
    showDialog(  
      context: context,  
      builder: (context){  
        return AlertDialog(  
          title: Text(_itens[indice]["titulo"]),  
          content: Text(_itens[indice]["descricao"]),  
        ); // AlertDialog  
      }  
    );  
  },  
);
```



ShowDialog e AlertDialog

```
- return ListTile(  
  onTap: (){  
    //print("Clique com onTap ${indice}");  
    showDialog(  
      context: context,  
      builder: (context){  
        return AlertDialog(  
          title: Text(_itens[indice]["titulo"]),  
          content: Text(_itens[indice]["descricao"]),  
        ); // AlertDialog  
      }  
    );  
  },  
);
```



AlertDialog

```
return AlertDialog(  
  title: Text(_itens[indice]["titulo"]),  
  titleTextStyle: TextStyle(  
    fontSize: 20,  
    color: Colors.red,  
    backgroundColor: Colors.yellow,  
  ), // TextStyle  
  content: Text(_itens[indice]["descricao"]),  
); // AlertDialog
```



AlertDialog

```
return AlertDialog(  
  title: Text(_itens[indice]["titulo"]),  
  titleTextStyle: TextStyle(...), // TextStyle  
  content: Text(_itens[indice]["descricao"]),  
  actions: <Widget>[ //definir widgets  
    FlatButton(  
      onPressed: (){  
        print("Selecionado sim");  
      },  
      child: Text("Sim")  
    ), // FlatButton  
    FlatButton(  
      onPressed: (){  
        print("Selecionado não");  
      },  
      child: Text("Não")  
    ), // FlatButton  
  ], // <Widget>[]  
); // AlertDialog
```

FlatButton
foi
depreciado



AlertDialog

```
return AlertDialog(  
  title: Text(_itens[indice]["titulo"]),  
  titleTextStyle: TextStyle(  
    fontSize: 20,  
    color: Colors.red,  
    backgroundColor: Colors.yellow,  
  ), // TextStyle  
  content: Text(_itens[indice]["descricao"]),  
  actions: <Widget>[ //definir widgets  
    TextButton(  
      onPressed: (){  
        print("Selecionado sim");  
        Navigator.pop(context);  
      },  
      child: Text("Sim")  
    ), // TextButton  
    TextButton(  
      onPressed: (){  
        print("Selecionado não");  
        Navigator.pop(context);  
      },  
      child: Text("Não")  
    ), // TextButton  
  ], // <Widget>[]  
); // AlertDialog
```

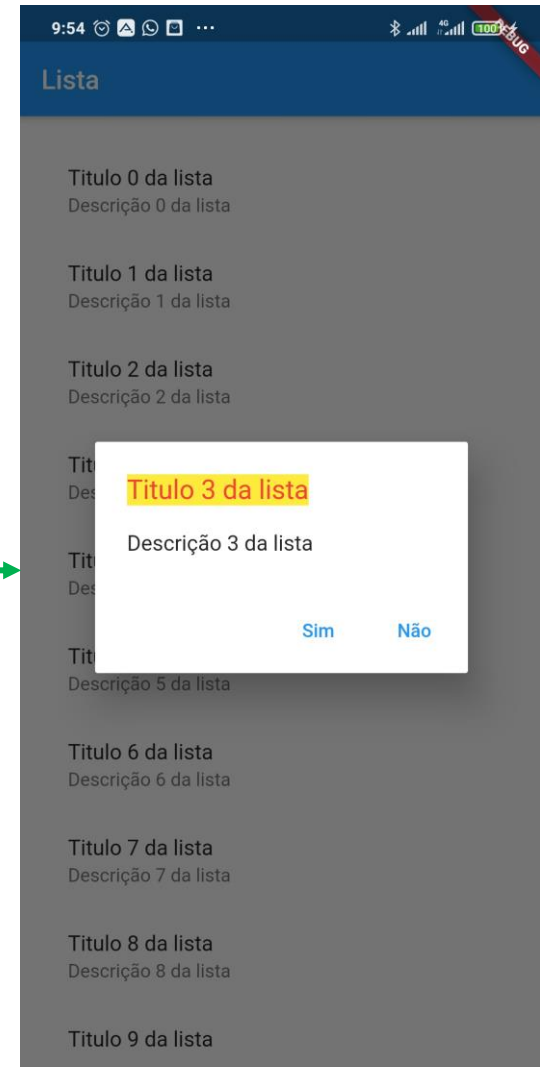
Vamos usar em
TextButton, vez
de, FlatButton



AlertDialog

```
Console  
I/flutter (28849): Selecionado não  
I/flutter (28849): Selecionado não  
I/flutter (28849): Selecionado não  
I/flutter (28849): Selecionado sim  
I/flutter (28849): Selecionado não
```

Ao clicar no Sim ou Não é impresso no console, Selecionado sim ou Selecionado não, no entanto, não fecha a tela de alerta



AlertDialog

```
actions: <Widget>[ //definir widgets
  FlatButton(
    onPressed: (){
      print("Selecionado sim");
      Navigator.pop(context);
    },
    child: Text("Sim")
  ), // FlatButton
  FlatButton(
    onPressed: (){
      print("Selecionado não");
      Navigator.pop(context);
    },
    child: Text("Não")
  ), // FlatButton
]. // <Widget>[]
```

Navigator.pop() para fechar a tela do AlertDialog ao clicar no Sim ou no Não

Referências Bibliográficas

- Curso da Udemy – Flutter Essencial do professor Ricardo Lecheta.
- Curso da Udemy - Desenvolvimento Android e IOS com Flutter 2020 – Crie 15 Apps do professor Jamilton Damasceno.
- Site Flutter – flutter.dev
 - <https://flutter.dev/docs/development/ui/layout>
 - <https://flutter.dev/docs/cookbook/lists/mixed-list>
 - <https://flutter.dev/docs/development/ui/layout#listview>
 - <https://api.flutter.dev/flutter/material/AlertDialog-class.html>