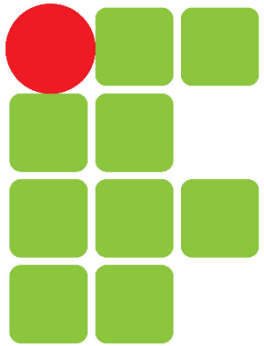
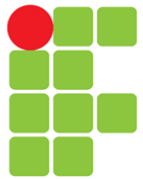


INSTITUTO FEDERAL
SÃO PAULO



Rotas e Navegação

Programação para Dispositivos Móveis (PDMI6)

Prof. Luiz Gustavo Diniz de Oliveira Véras

E-mail: gustavo_veras@ifsp.edu.br

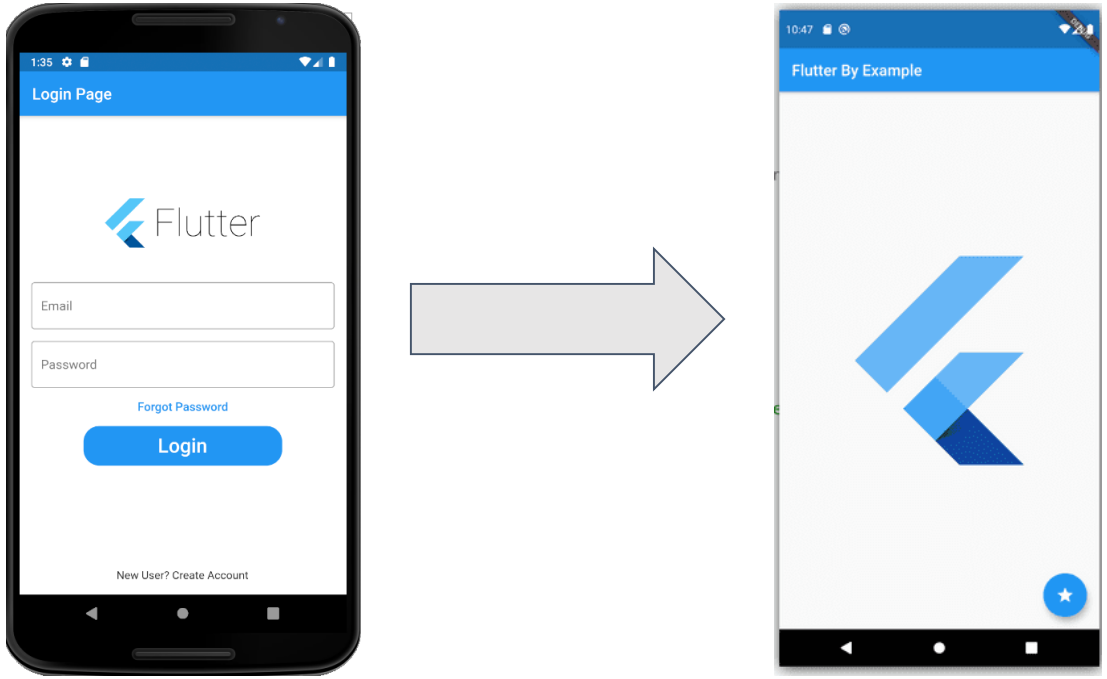
Conteúdo

- Introdução
- Criação de páginas
- Navegando entre páginas
- Passando dados para uma página
- Recebendo dados de uma página
- Exercícios

Introdução

Organizar informações em várias telas é um dos blocos de construção mais importantes de qualquer arquitetura de um aplicativo.

Um exemplo muito comum é aquele em que a primeira página do seu aplicativo é um **formulário de login** e, se o usuário fornecer uma combinação correta de nome de usuário e senha, uma **página de boas-vindas** parece.

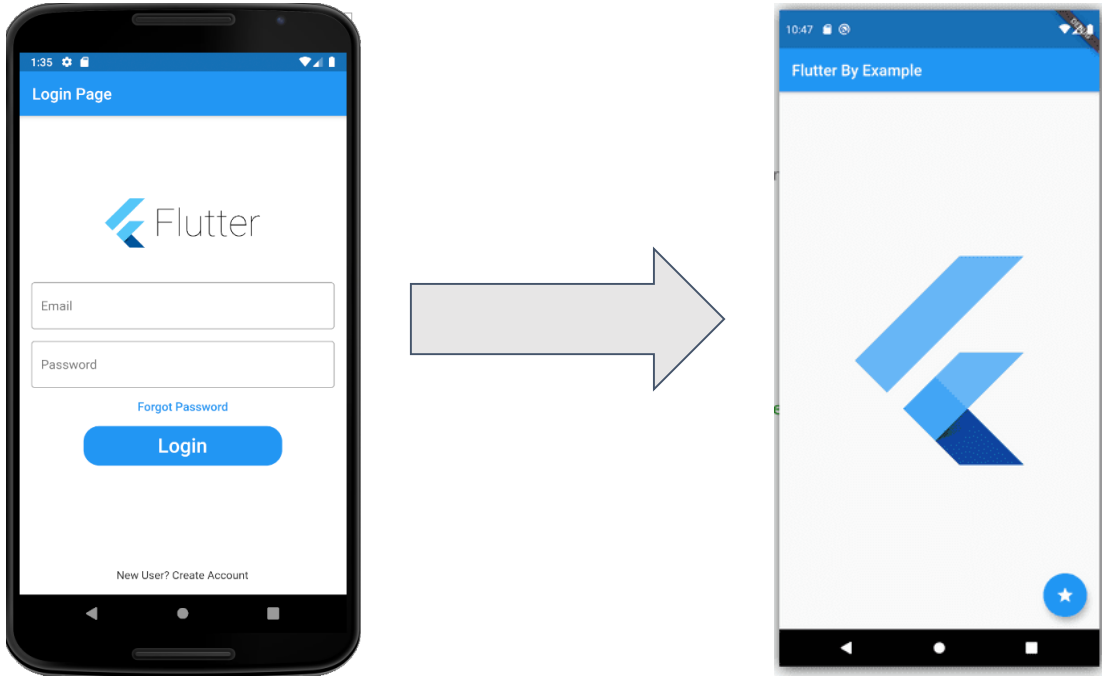


Introdução

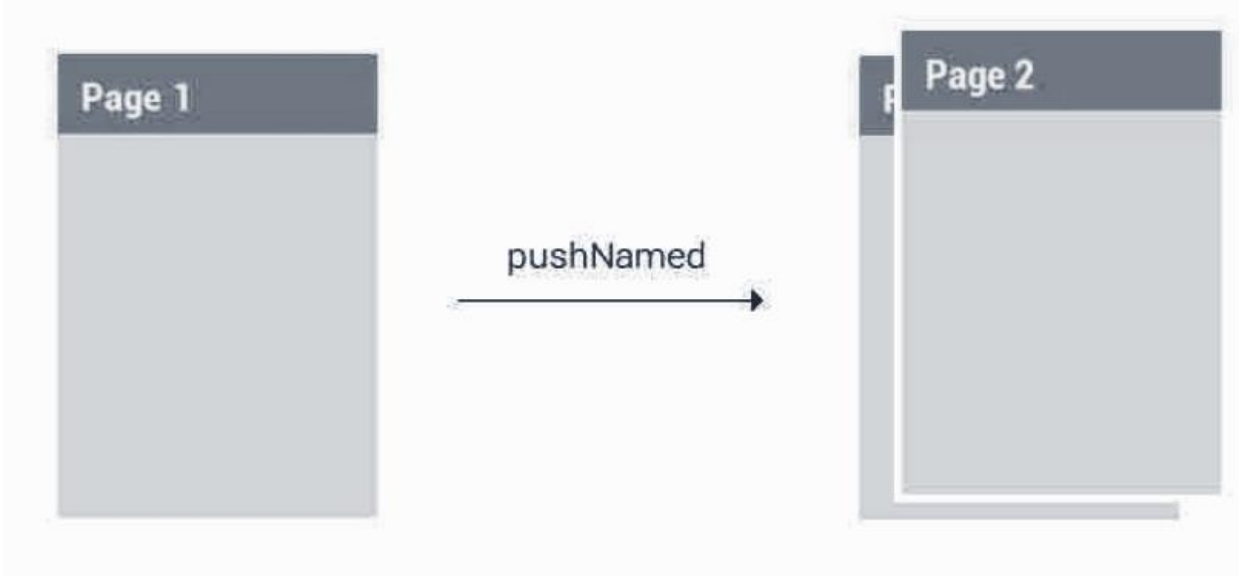
No mundo Flutter, as páginas do seu aplicativo são chamadas de **routes**, ou **screens**.

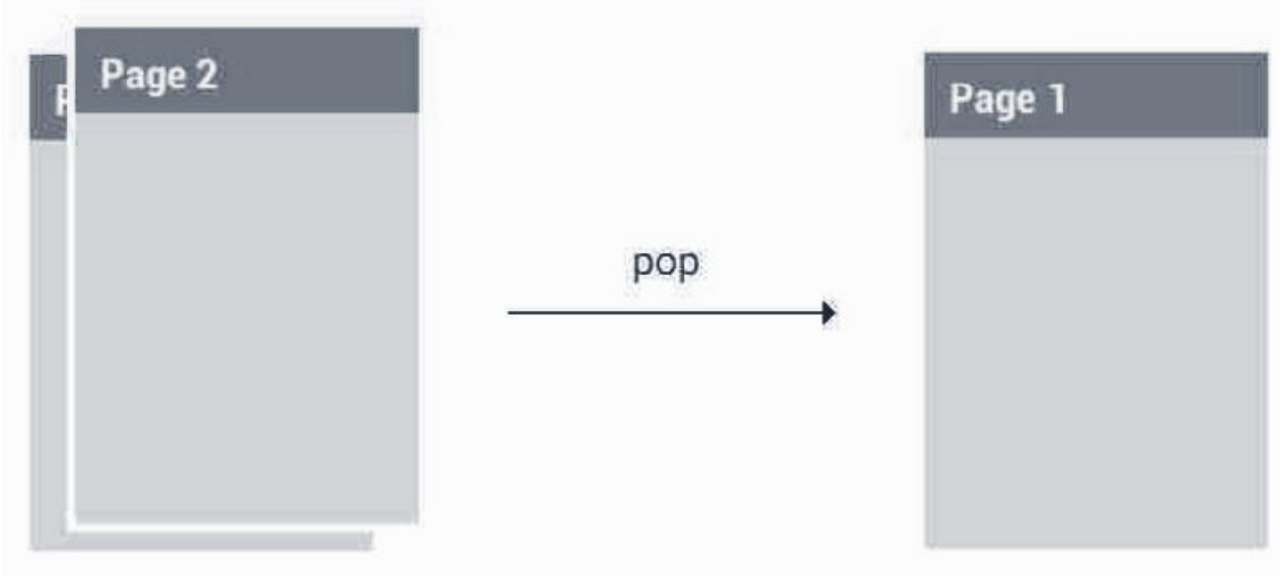
Elas são equivalentes a:

- a **Activities** no Android
- **ViewControllers** no iOS



As páginas para as quais você navega são "sobrepostas" e a estrutura as acompanha usando uma stack (pilha).



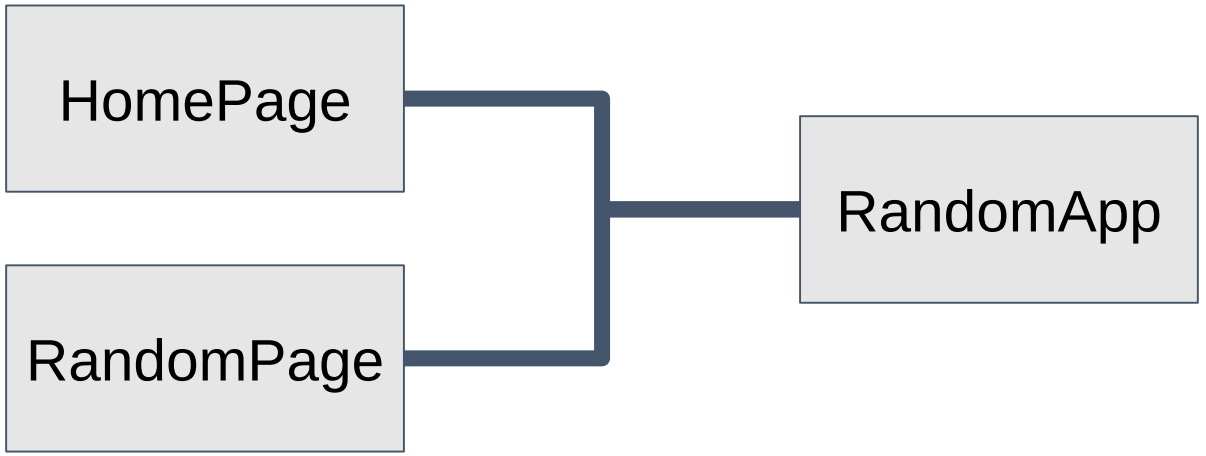


Criação de rotas

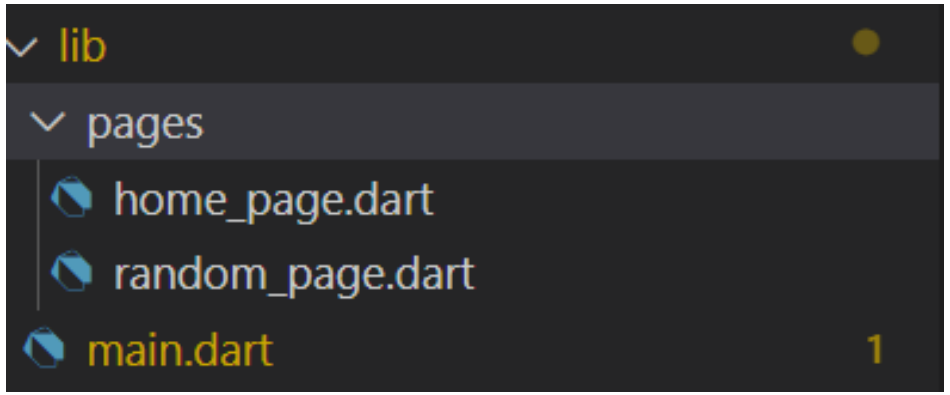
Vamos criar duas páginas

- HomePage
- RandomPage (Vai gerar um número aleatório no meio da página)

A arquitetura do exemplo será:



Criação de rotas



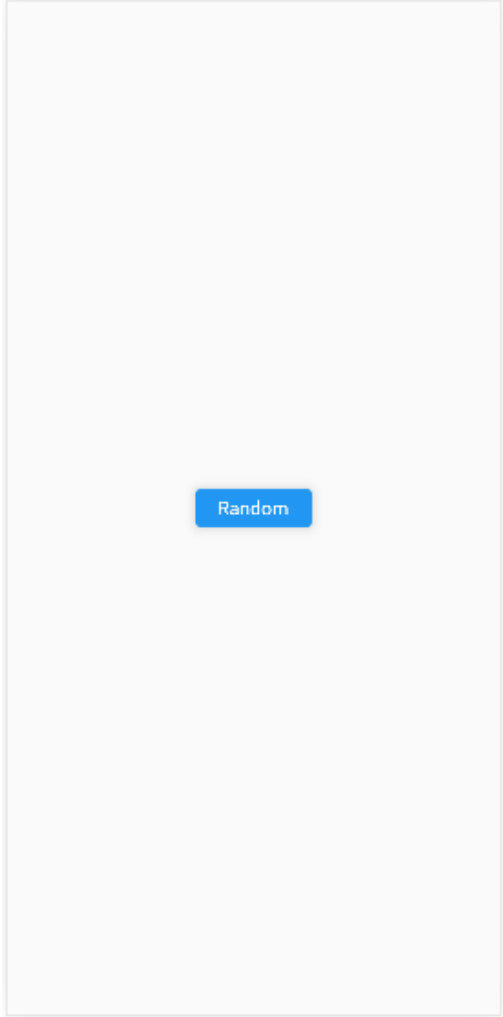
Quando há muitos arquivos, criar uma boa hierarquia de diretórios e subdiretórios é fundamental para não se perder em sua própria arquitetura.

Crie uma pasta especifica para as páginas.

Criação de rotas

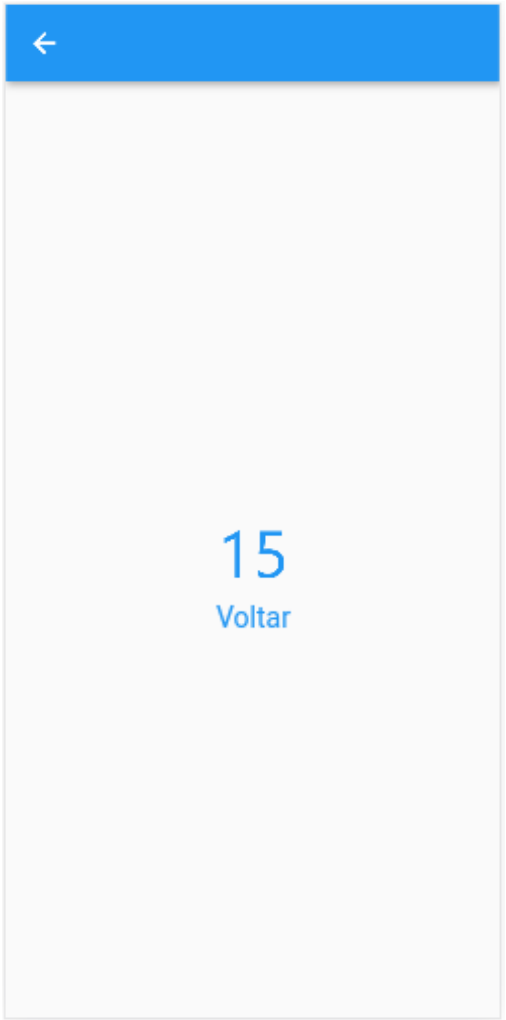
routes/home_page.dart

```dart
import 'package:flutter/material.dart';

class HomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            // Código 1: Navega para a segunda tela.
          },
          child: const Text("Random"),
        ),
      ),
    );
  }
}
```



routes/random_page.dart

```dart
import 'dart:math';
import 'package:flutter/material.dart';

class RandomPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
       appBar: AppBar(),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text("${Random().nextInt(20)}"),
            TextButton(
              onPressed: () => {
                  // Código 2: Retornar para a primeira tela.
               },
              child: Text("Voltar"),
            )
          ],
        ),
      ),
    );
  }
}
```

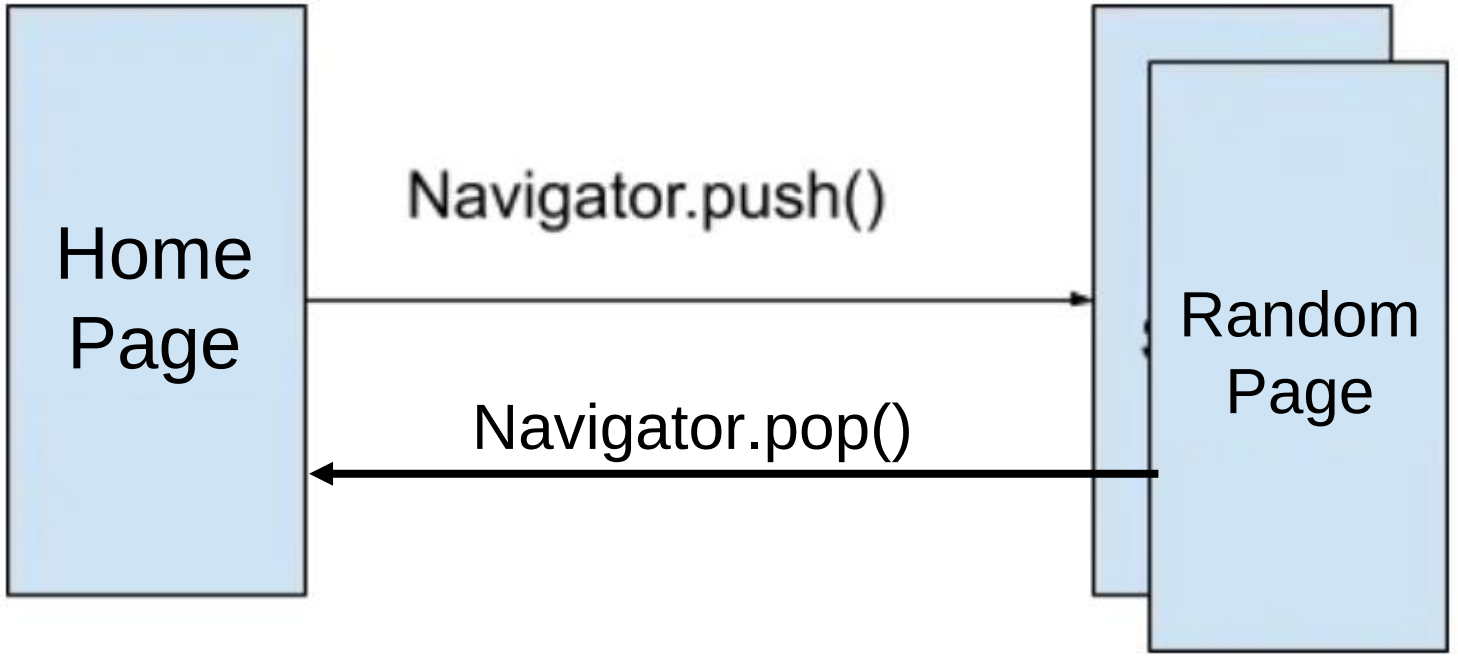


Criação de rotas

main.dart

```dart
import 'package:flutter/material.dart';

void main() {
  runApp(RandomApp());
}

class RandomApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: HomePage(),
    );
  }
}
```

Criação de rotas



Navegando entre as rotas

Dentro de um *Event Handler (*atributo onPressed*)* de um botão, por exemplo, podemos invocar os métodos:

Para ir para uma screen (Código 1)

```
Navigator.push(context, MaterialPageRoute(builder: (context) => RandomPage())
```

Para voltar de uma tela (Código 2)

```
Navigator.pop(context)
```

Navegando entre as rotas

A classe **MaterialPageRoute<T>** substitui uma tela por outra usando uma transição de slide do Android.

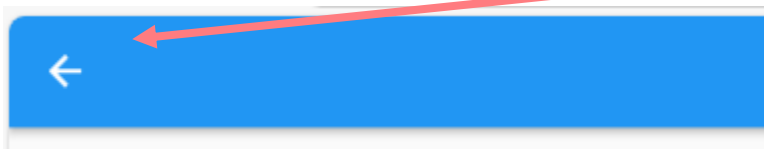
Dentro de um *Event Handler (*atributo onPressed*)* de um botão, por exemplo, podemos invocar os métodos:

Para ir para uma screen (Código 1)

```
Navigator.push(context, MaterialPageRoute(builder: (context) => RandomPage())
```

Para voltar de uma tela (Código 2)

```
Navigator.pop(context)
```



Se o **Scaffold** dentro do **MaterialApp** contiver um **AppBar**, um botão de retorno que funciona como o método **pop()** será criado.

Navegando entre as rotas

routes/home_page.dart

```
        child: ElevatedButton(
          onPressed: () {
            Navigator.push(
              context,
              MaterialPageRoute(
                builder: (context) => RandomPage(),
              ),
            );
          },
          child: Text("Random"),
        ),
```

Para ir para uma tela (Código 1)

routes/random_page.dart

```
  TextButton(
              onPressed: () => Navigator.pop(context),
              child: Text("Voltar"),
            )
```

Para voltar de uma tela (Código 2)

Navegando entre as rotas

O que fizemos até agora:

- Criamos duas rotas denominadas **HomePage** e **RandomPage**;
- Configuramos rotas no **MaterialApp(**) do main.dart;

Agora vamos utilizar a classe **Navigator** para enviar e receber dados entre as rotas/telas.

Passando dados entre páginas e widgets

O compartilhamento de dados entre duas ou mais páginas é uma necessidade muito comum:

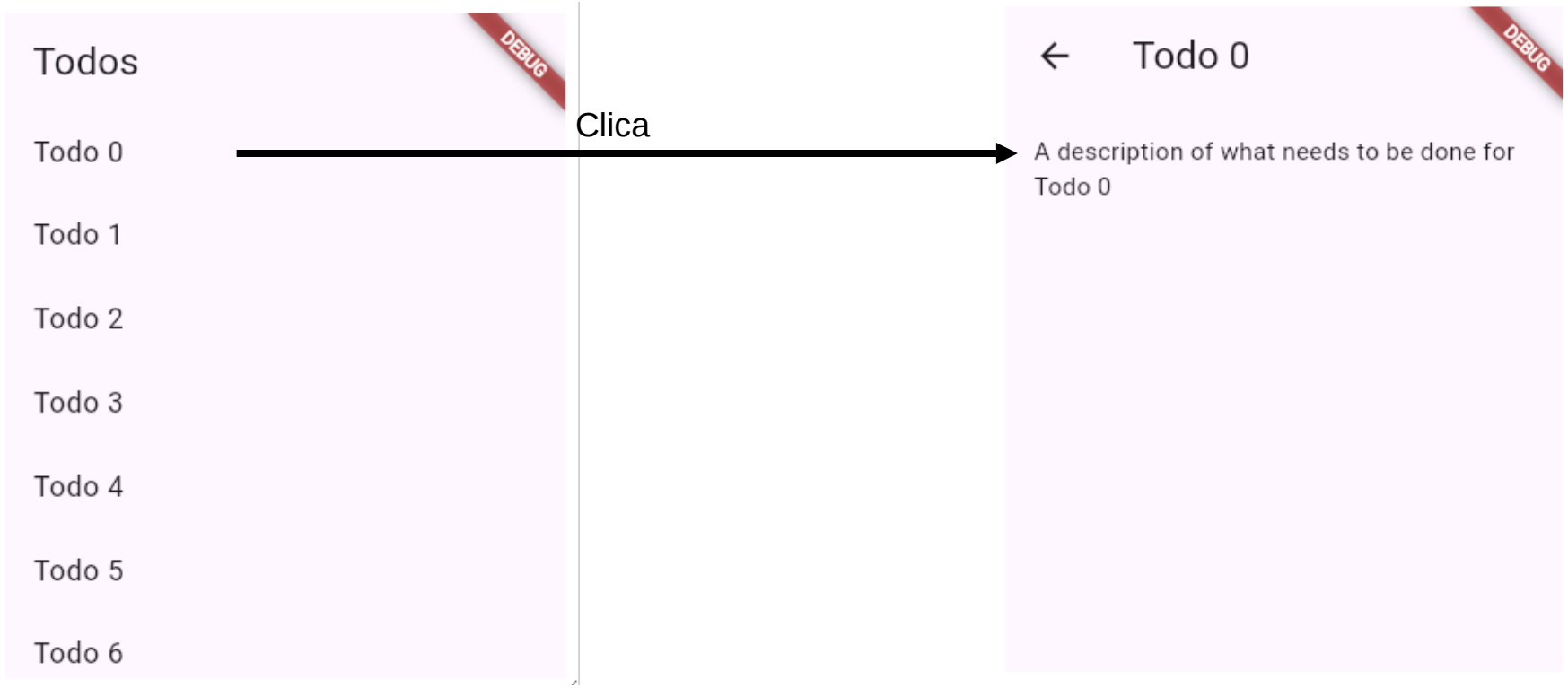
Widgets diferentes localizados em várias partes da árvore de *Widgets* podem exigir algum compartilhamento de dados.

Alguns casos comuns dessa necessidade:

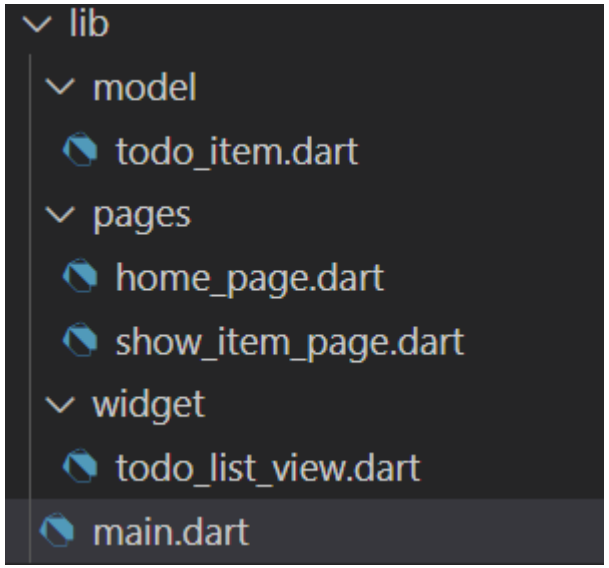
- Em um layout com muitas abas, você precisa passar os dados de uma aba para outra;
- Você tem uma rota (uma página) que deve enviar um tipo primitivo ou um objeto complexo para outro rota ao navegar para ela;
- Na mesma página, dois widgets precisam trocar dados.

Passando dados entre páginas e widgets

Exemplo: Lista de tarefas (TODO List)



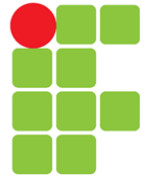
Passando dados entre páginas e widgets



Quando há muitos arquivos, criar uma boa hierarquia de diretórios e subdiretórios é fundamental para não se perder em sua própria arquitetura.

Crie uma pasta especifica para as páginas.

Passando dados entre páginas e widgets



#1 - A primeira coisa a fazer é a criação de uma classe de modelo, que é uma classe que representa um item de tarefa a ser feita.

model/todo_item.dart

```dart
class TodoItem {
  late final String _title; // atributo privado (_nomeField)
  late final String _description; // atributo privado (_nomeField)


  //construtor com named parameters
  TodoItem({required String title, required String description})
      : _title = title,
        _description = description;

  String get title => _title;
  String get description => _description;
}
```

Passando dados entre

#2 - Para armazenar os objetos de ***Todo***, vamos criar uma classe dedicada a isso em um *widget* que vai ser usado para listar os items.

O método List.generate automatiza a criação de itens baseado em seus índices, e cria uma instância de TodoItem para cada um deles.

pages/home_page.dart

```dart
import 'package:flutter/material.dart';
import '../model/todo_item.dart';

class HomePage extends StatelessWidget {
  late final List<TodoItem> todos;

  HomePage() {
    todos = List.generate(
        30,
        (index) => TodoItem(
            title:
                "Item ${index}",
            description:
                "Uma descrição para o Item ${index}"));
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(),
      body: TodoListView(todo), // Classe especificada por
nós!!
    );
  }
}
```

Passando dados entre

#3 – Crie um *Widget* que gera uma *ListView* de Todo.

Note o uso de **ListTile**. Este **Widget** é utilizado para criar **itens de lista** com um layout padronizado. Com ele podemos configurar a função *Event Handler* que será executada quando um item for selecionado.

ShowItemPage será o Widget utilizado para apresentar as informações de um TodoItem.

widget/todo_list_view.dart

```dart
import 'package:flutter/material.dart';
import 'package:todo_list/pages/show_item_page.dart';
import '../model/todo_item.dart';

class TodoListView extends StatelessWidget {
  const TodoListView({
    required this.todos,
  });

  final List<TodoItem> todos;

  @override
  Widget build(BuildContext context) {
    return ListView.builder(
      itemCount: todos.length,
      itemBuilder: (context, index) {
        return ListTile(
          title: Text(todos[index].title),
          onTap: () => Navigator.push(
            context,
            MaterialPageRoute(
              builder:
                  (context) =>
                      ShowItemPage(todo: todos[index]),
            ),
          ),
        );
      },
    );}
}
```

Passando dados entre

#3 – Crie um *Widget* que gera uma *ListView* de Todo.

Note o uso de **ListTile**. Est **Widget** é utilizado para criar **iten de lista** com um layou padronizado. Com ele podemo configurar a função *Event Handle* que será executada quando um item for selecionado.

ShowItemPage será o Widget utilizado para apresentar as informações de um TodoItem.

widget/todo_list_view.dart

```
import 'package:flutter/material.dart';
import 'package:todo_list/pages/show_item_page.dart';
import '../model/todo_item.dart';

class TodoListView extends StatelessWidget {
  const TodoListView({
    required this.todos,
  });

  st<TodoItem> todos;

  uild(BuildContext context) {
   ListView.builder(
  ount: todos.length,
  uilder: (context, index) {
  turn ListTile(
    title: Text(todos[index].title),
    onTap: () => Navigator.push(
      context,
      MaterialPageRoute(
        builder:
            (context) =>
                ShowItemPage(todo: todos[index]),
      ),
    ),
  );
  },
  );}
}
```

Passando dados entre páginas e widgets

#4 - Agora, crie a segunda tela. O título da tela contém o título do todo, e o corpo da tela mostra a descrição.

Como a tela de detalhes é um *StatelessWidget* normal, exija que o usuário insira um Todo na IU. Em seguida, crie a IU usando o objeto Todo fornecido.

pages/show_item_page.dart

```dart
import 'package:flutter/material.dart';
import '../model/todo_item.dart';

class ShowItemPage extends StatelessWidget {
  // Declara o campo que mantém o TodoItem
  final TodoItem todo;

  // Requere Todo no construtor.
  const ShowItemPage({required this.todo});

  @override
  Widget build(BuildContext context) {
    // Usa o Todo para criar o UI
    return Scaffold(
      appBar: AppBar(title: Text(todo.title)),
      body: Padding(
        padding: const EdgeInsets.all(16),
        child: Text(todo.description),
      ),
    );
  }
}
```

Passando dados entre páginas e widgets

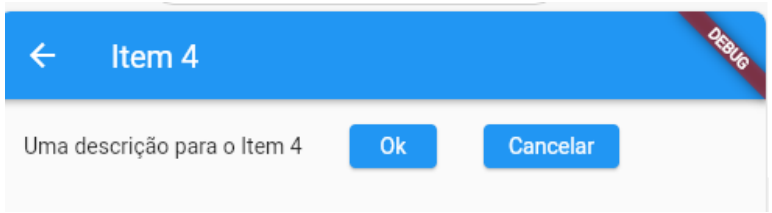
Em alguns casos, você pode querer retornar dados de uma nova tela. Por exemplo, digamos que você envie uma nova tela que apresente duas opções para um usuário.

Quando o usuário toca em uma opção, você quer informar a primeira tela da seleção do usuário para que ele possa agir com base nessas informações.

Você pode fazer isso com o método **`Navigator.pop().`**

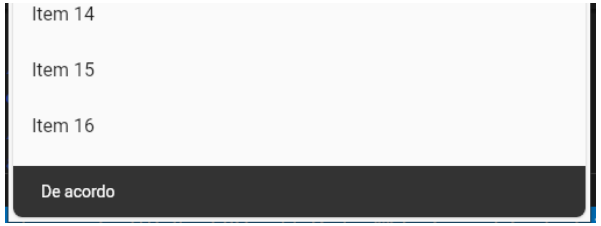
Passando dados entre páginas e widgets

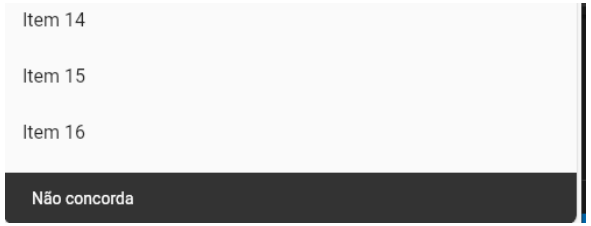
Aproveitando o mesmo exemplos anterior, vamos adicionar dois botões em **ShowItemPage**.




```
Navigator.pop(context, true)

Navigator.pop(context, false)
```

Em **TodoListView,** devemos esperar o retorno de um objeto Future e acionar um SnackBar.






```
Future _showResponseFromNextPage() async {

    ScaffoldMessenger.of(context)
      ..removeCurrentSnackBar()
      ..showSnackBar(
          SnackBar(content:
          Text('${result ? "De acordo" : "Não
                                        concorda"}')));
}
```

Passando dados entre páginas e widgets

#5 – Em *TodoListView*, define o *TileList* para invocar a função que retorna um objeto Future.

Quando o usuário selecionar uma das opções em *ShowItemPage*, este será o código responsável por lidar com a resposta.

pages/todo_list_view.dart

```
return ListTile(
         title: Text(todos[index].title),
         onTap: () => _showResponseFromNextPage(context, index),
       );
```

```
Future _showResponseFromNextPage(BuildContext context, int index) async {
  final result = await Navigator.push(
    context,
    MaterialPageRoute(
      builder: (context) => ShowItemPage(todo: todos[index]),
    ),
  );

  ScaffoldMessenger.of(context)
    ..removeCurrentSnackBar()
    ..showSnackBar(
        SnackBar(content: Text('${result ? "De acordo" : "Não concorda"}')));
}
```

Passando dados entre páginas e widgets

#6 – Em *ShowItemPage*, insira botões que chamando ***Navigator.pop*** com valores respectivos às suas ações.

No botão “Ok”, *true* deve ser retornado para a tela anterior. No botão “Cancelar”, deve ser retornado *false*.

pages/show_item_page.dart

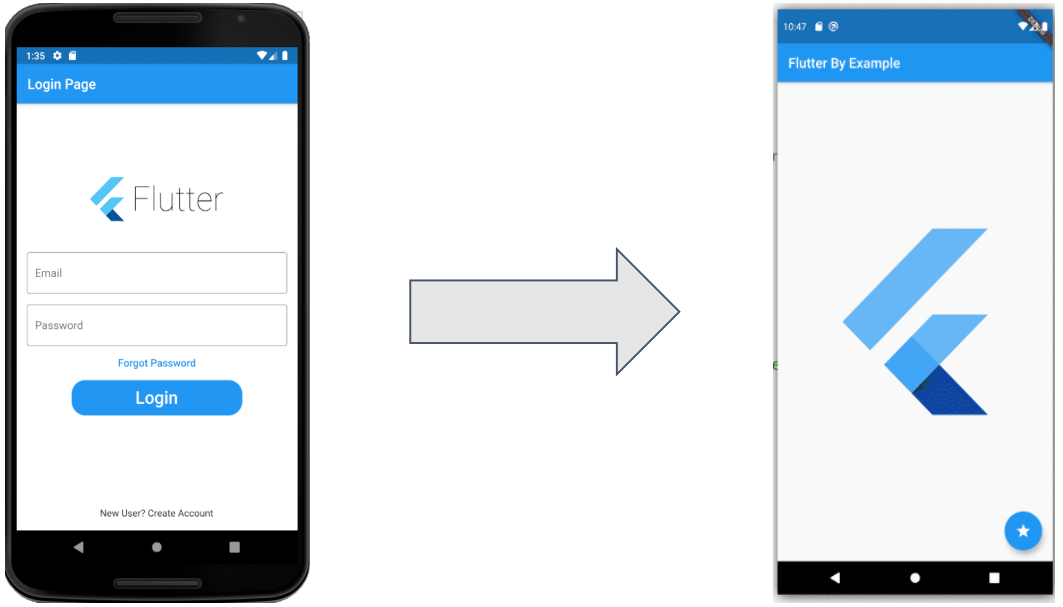
```
return Scaffold(
      appBar: AppBar(title: Text(todo.title)),
      body: Padding(
        padding: const EdgeInsets.all(16),
        child: Row(
          children: [
            Text(todo.description),
            Padding(padding: EdgeInsets.symmetric(horizontal: 15)),
            ElevatedButton(
              onPressed: () => Navigator.pop(context, true),
              child: Text("Ok"),
            ),
            Padding(padding: EdgeInsets.symmetric(horizontal: 15)),
            ElevatedButton(
              onPressed: () => Navigator.pop(context, false),
              child: Text("Cancelar"),
            ),
          ],
        ),
      ),
    );
```

Exercícios

1. Implemente o layout representado abaixo. O login só será bem-sucedido se os valores digitados forem "admin" e "1234". Se corretos, o usuário será redirecionado para a nova página. Caso contrário, uma mensagem de falha de login será exibida via **SnackBar**.

Veja alguns Widgets novos que podem ser utilizados:

Imagem (Image.asset ou Image.network):
https://api.flutter.dev/flutter/widgets/Image-class.html
Input (TextField):
https://api.flutter.dev/flutter/material/TextField-class.html
FloatingActionButton
https://api.flutter.dev/flutter/material/FloatingActionButton-class.html
Exibição de erro (SnackBar):
https://api.flutter.dev/flutter/material/SnackBar-class.html

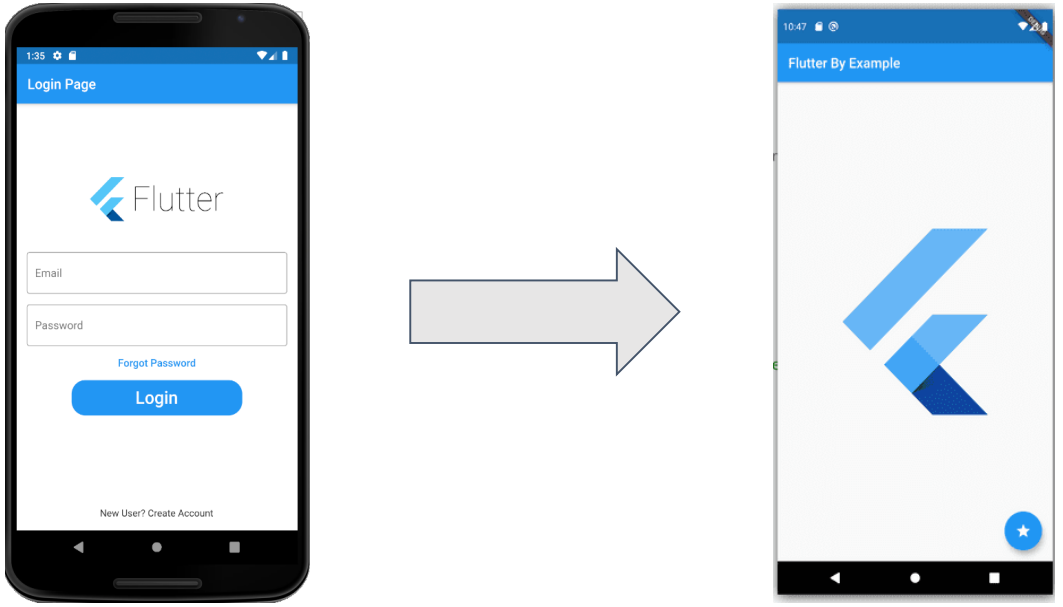


Exercícios

1. Implemente o layout representado abaixo. O login só será bem-sucedido se os valores digitados forem "admin" e "1234". Se corretos, o usuário será redirecionado para a nova página. Caso contrário, uma mensagem de falha de login será exibida via **SnackBar**.

Siga a seguinte sugestão de arquitetura!

- `models/login.dart` → Define a classe `Login` para armazenar usuário e senha.
- `screens/login_screen.dart` → Tela de login, onde o usuário digita as credenciais.
- `screens/home_screen.dart` → Tela que o usuário acessa após o login.
- `widgets/login_form.dart` → Um widget reutilizável contendo os inputs e botão.
- `assets/logo.png` → Uma imagem para exibir na tela de login.



Referência

MIOLA, Alberto. **Flutter Complete Reference: Create Beautiful, Fast and Native Apps for Any Device.** In:___. Routes and navigation. Independently published, 2020.

Simple app state management. <Disponível em https://docs.flutter.dev/data-and-backend/state-mgmt/simple>

Navigation and routing. <Disponível em https://docs.flutter.dev/ui/navigation>

Navigate to a new screen and Back. <Disponível em https://docs.flutter.dev/cookbook/navigation/navigation-basics >

Send data to a new screen. <Disponível em https://docs.flutter.dev/cookbook/navigation/passing-data >

Return data from a screen <Disponível em https://docs.flutter.dev/cookbook/navigation/returning-data >