

Workshop - Flutter

Sobre Flutter

- Framework para desenvolvimento de interface de usuário
- Baseado no Dart
- Criação de aplicativos Mobile, Desktop e Web
- Desenvolvido pelo Google
- Versão atual 3.32.*



Rápida



Produtiva

```
Scaffold( appBar: AppBar( title: Text('Hello World'), ), body: ...)
```

Flexível

Sobre Flutter

- Porque Flutter?
 - Mercado
 - Versatilidade
 - Desempenho



ByteDance



ebay



GROUPON



Conceitos iniciais

Flutter

Conceitos iniciais

- Estrutura inspirada no React
- Constrói a UI a partir de widgets
- Widgets descrevem a aparência e o comportamento
- Mudança de estado faz o widget reconstruir
- O Flutter usa árvore de widgets a partir de `runApp()`
- Widgets são subclasses de **Stateless** ou **Stateful**
 - Sobrescrevendo a função `build()`

Conceitos iniciais

- Widget → Componente da tela (texto, botão, ícone, imagem, ...)

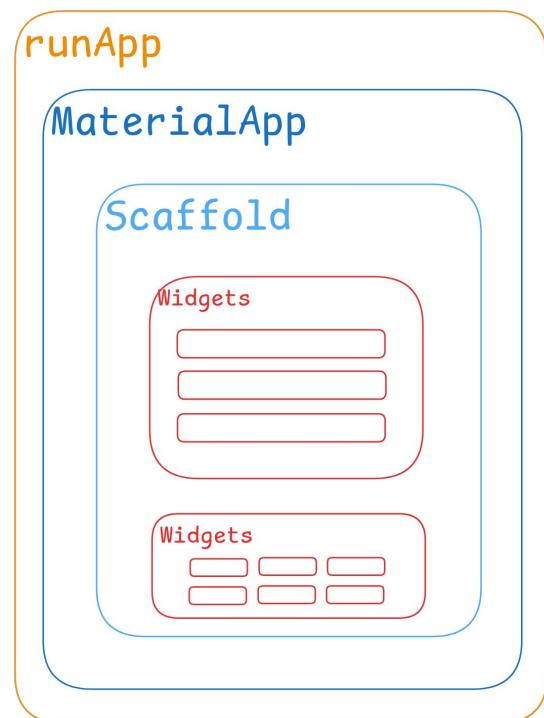
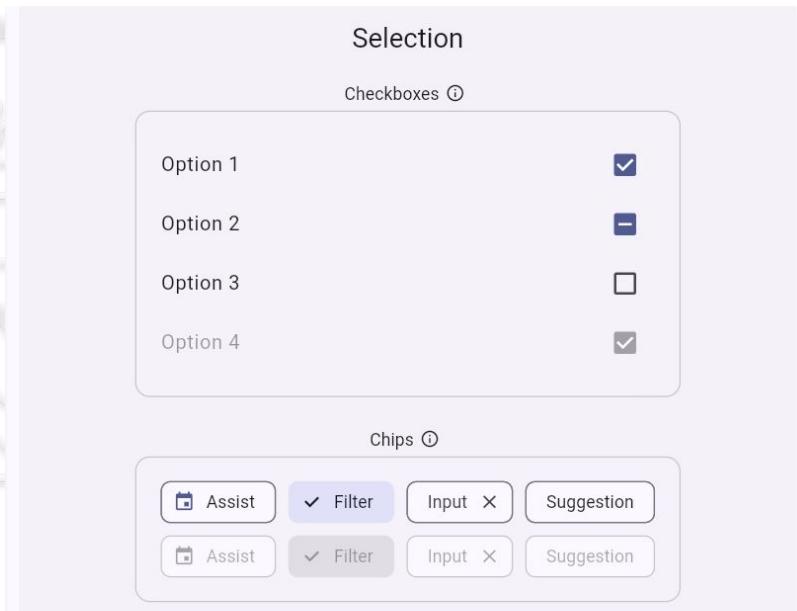
Material 3

The screenshot displays the Material 3 design system interface. On the left, a sidebar lists 'Components', 'Color', 'Typography', and 'Elevation'. Below these are sliders for 'Brightness' and 'Material 3'. A grid of color swatches shows rows for blue, green, yellow, orange, red, and purple. A grid of icon swatches shows icons for leaf, cloud, flower, tree, and animal. The main area is divided into sections:

- Actions**:
 - Common buttons**: Shows examples of Elevated, Filled, Filled tonal, and Outlined buttons with '+' icons.
 - Floating action buttons**: Shows four blue buttons with '+' icons, one of which is larger and highlighted.
 - Icon buttons**: Shows eight circular buttons with gear icons.
 - Segmented buttons**: Shows a horizontal bar with segments for Day, Week, Month, and Year.
- Selection**:
 - Checkboxes**: Shows four options (Option 1, Option 2, Option 3, Option 4) each with a checked checkbox.
 - Chips**: Shows four chips labeled Assist, Filter, Input, and Suggestion, each with a checkmark or X.
 - Date picker**: Shows a date input field with a calendar icon and the text "Show date picker".
 - Time picker**: Shows a time input field with a clock icon and the text "Show time picker".
 - Menus**: Shows a menu icon with a three-dot ellipsis.

Conceitos iniciais

- Árvore de widgets



Widgets usuais

Flutter

Widgets usuais

- **MaterialApp** → Widgets necessários para apps do Material Design
- **Scaffold** → Aplica o Material Design nos widgets
- **Text** → Sequência de texto estilizado em seu aplicativo
- **Row, Column** → Widgets nas direções horizontal (**Row**) e vertical (**Column**)
- **Container** → Retângulo personalizável com o **BoxDecoration**
- **Center** → Centraliza o widget em layouts flexíveis
- **ElevatedButton** → Um botão com aparência elevada em um plano
- **OutlinedButton** → Botão que contém somente a borda aparente
- **Expanded** → Faz o widget ocupar todo o espaço disponível

Widgets usuais

- Abra o VS Code
- Instale a extensão Flutter Widget Snippets
- Novo projeto: Ctrl + Shift + P → Flutter: New Project → Empty Application
- Selecione o local onde deseja salvar
- Nomeie seu projeto, semelhante ao exemplo, usando seu nome:
 - project_jefferson
- Execute seu projeto sem fazer nenhuma alteração



Widgets usuais

- Estrutura do projeto:
 - `.dart_tool/`
 - `.idea/`
 - **`assets/` → Suas imagens ficam aqui (criar pasta manualmente)**
 - `build/`
 - **`lib/` → Seus códigos ficam aqui**
 - `linux/`
 - `...`
 - **`pubspec.yaml` → Declaração de bibliotecas e arquivos personalizados aqui**
 - `...`

Como construir uma tela?

Flutter

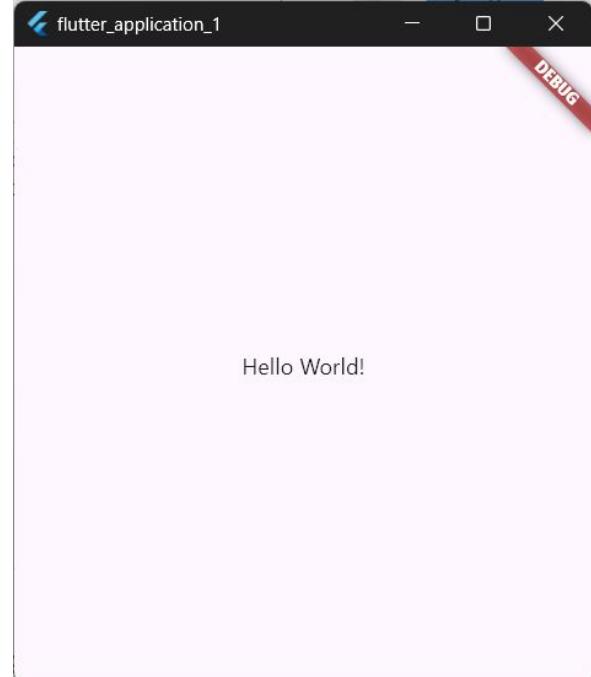
Nova tela

- Stateless:
 - Widget estático
 - Método build() → constrói o widget
 - Mais rápido para renderizar
- Stateful:
 - Widget dinâmico
 - Métodos:
 - initState() → 1x antes do build
 - build() → constrói o widget
 - dispose() → Quando a tela sai
 - setState() → Atualiza a tela
 - Lembrem de usar @override antes do método
 - Esses métodos não tem parâmetros e não tem retorno, ou seja, void.

Nova tela

- Stateless

```
class MainApp extends StatelessWidget {  
  const MainApp({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return const MaterialApp(  
      home: Scaffold(  
        body: Center(  
          child: Text('Hello World!'),  
        ),  
      ),  
    );  
  }  
}
```



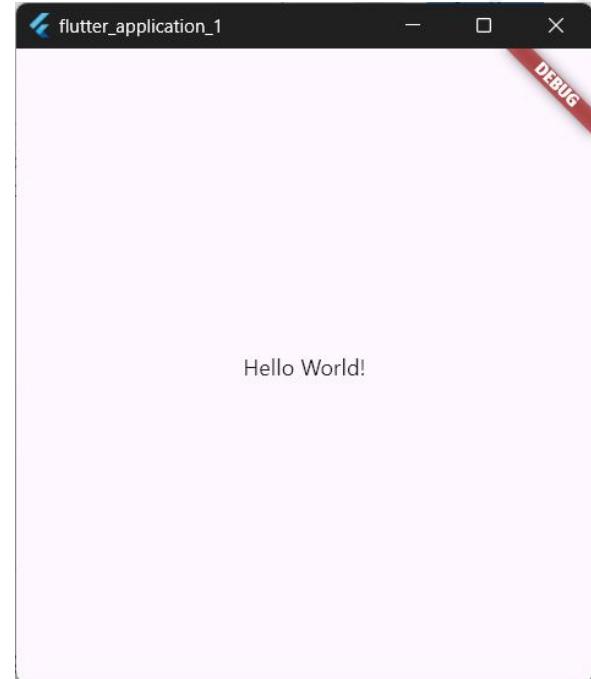
Nova tela

- Stateful

```
class MainApp extends StatefulWidget {
  const MainApp({super.key});
  @override
  State<MainApp> createState() => MainAppState();
}

class MainAppState extends State<MainApp> {
  @override
  void initState() {
    // Rotinas necessárias aqui
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      home: Scaffold(body: Center(child: Text('Hello World!'))));
  }
}
```



Nova tela

- Dentro da pasta lib:
 - Crie um arquivo nomeado second_page.dart
 - Escreva stl e selecione Flutter StatefulWidget
 - Nomeie a classe como SecondPage
 - Na função build():
 - Remova o Placeholder() e adicione um MaterialApp()
 - Dentro do MaterialApp pressione ctrl + espaço e selecione home e adicione um Scaffold()
 - Dentro do Scaffold pressione ctrl + espaço e selecione body e adicione um Center()
 - Dentro do Center pressione ctrl + espaço e selecione **child** e adicione um Text()
 - O Text recebe uma **String** para ser exibido na tela, por exemplo “Second Page”
 - No arquivo main, na função main remova MainApp e adicione SecondPage

Nova tela

- Adicionar um campo de texto e atualizar o texto exibido
 - Na classe `SecondPageState` para recuperar o texto digitado:
 - `final _controller = TextEditingController(text: ''');`
 - `var textScreen = 'Second page';`
 - Envolver com uma `Column` e alterar o `text` em `Center(child: Text(textScreen)),`
 - Adicionar dentro da `Column`
 - `TextField(controller: _controller, decoration: InputDecoration(border: OutlineInputBorder(), label: Text('My Label'))),`
 - Adicione um novo **ElevatedButton** com o texto “**Update**” e dentro da função `onPressed` nos {} adicione
 - `setState(() {textScreen = _controller.text;});`
- Podemos limitar o tamanho do `TextField` usando um `SizedBox`
- Dica: **ctrl + espaço** mostra os atributos disponíveis

Como navegar entre as telas?

Flutter

Navegar entre telas

- O MaterialApp disponibiliza o Navigator
- Navigator.push(context, Route)
 - Empilha a tela usando o MaterialPageRoute
 - `Navigator.push(context, MaterialPageRoute(builder: (context) => SecondPage()))`
- Navigator.pop(context)
 - Desempilha a tela com `Navigator.pop(context);`

Navegar entre telas

- Organização inicial:
 - No runApp retorne para usar o MainApp
 - Crie uma novo arquivo first_page.dart e crie a classe FirstPage como stateless
 - Adicione a FirstPage no home do MaterialApp do MainApp
 - Adicione um Scaffold e uma Column no body do Scaffold, dentro da FirstPage
 - Adicione um `Text('First Page'),`
 - Abaixo dele Adicione:
 - `ElevatedButton(onPressed: () {}, child: Text('Go to Second Page'))`
 - Remover o **const** antes do Scaffold, se houver
 - Organizar os códigos dentro de src/presenter/pages
 - Remove da SecondPage o MaterialApp

Navegar entre telas

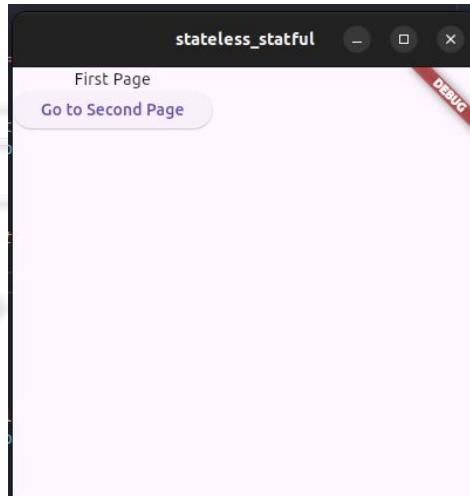
- Dentro do ElevatedButton():
 - Na função onPressed, dentro das {}
 - `Navigator.push(context, MaterialPageRoute(builder: (context) => SecondPage()))`
- No arquivo second_page.dart
 - Adicione outro ElevatedButton com o texto “Back to First Page”
 - No onPressed, dentro das {}
 - `Navigator.pop(context);`
- Volte para o main.dart e execute a aplicação

Organização dos widgets

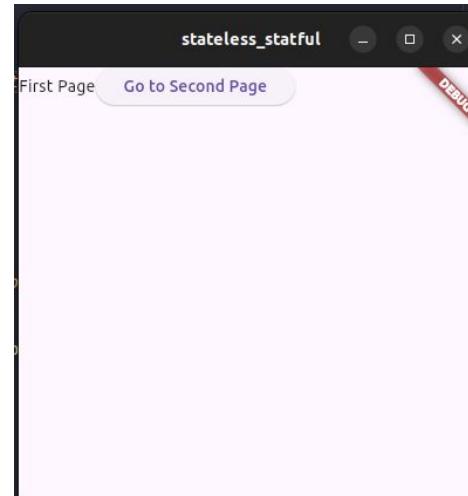
Flutter

Organização da tela

- Organização principal por colunas (Column) e linhas (Row)
- Child or childrens?



Column

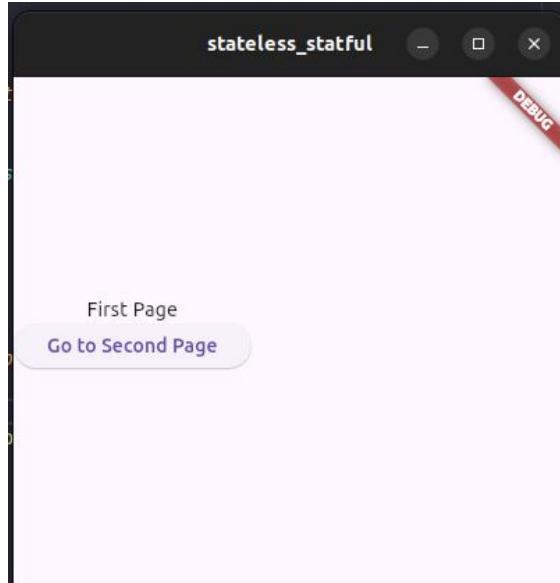
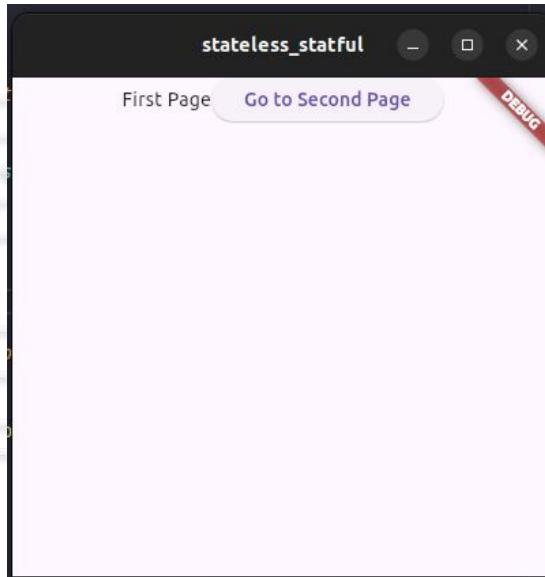


Row

Organização da tela

Alinhamentos na Row e na Column:

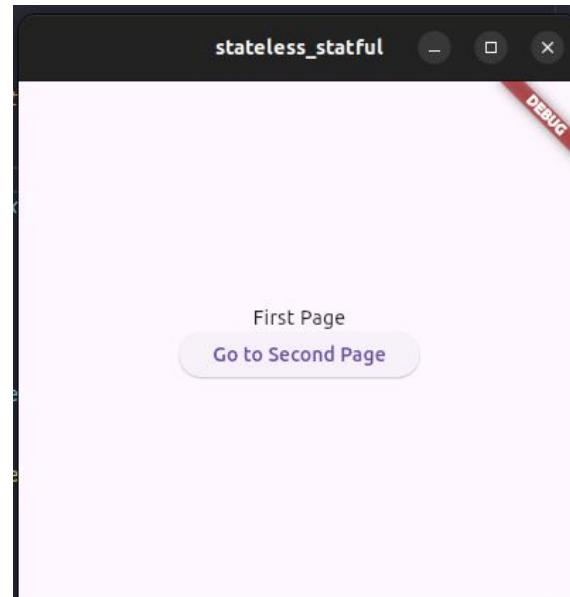
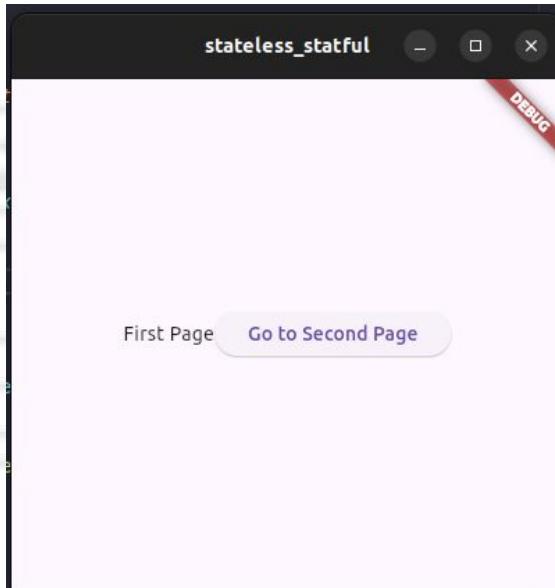
```
Row(mainAxisAlignment: MainAxisAlignment.center,)    Column(mainAxisAlignment: MainAxisAlignment.center,)
```



Organização da tela

E para ficar no meio da tela?

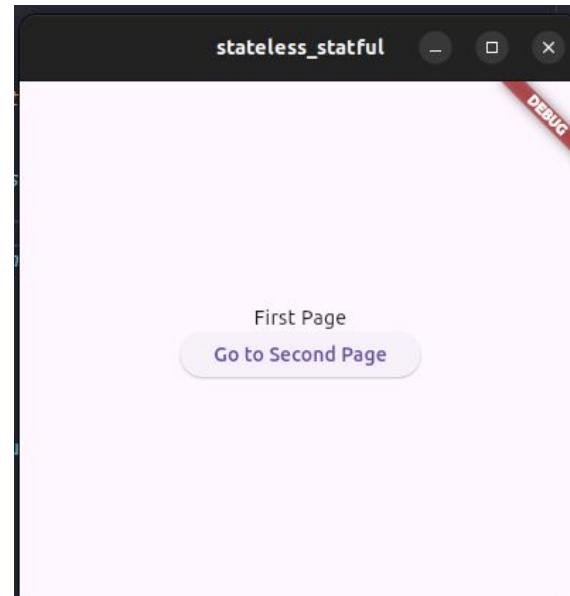
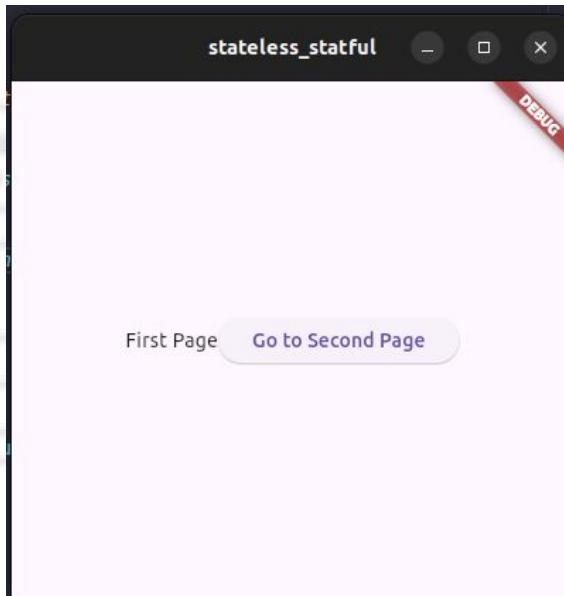
- Envolve a Row ou a Column com o Center



Organização da tela

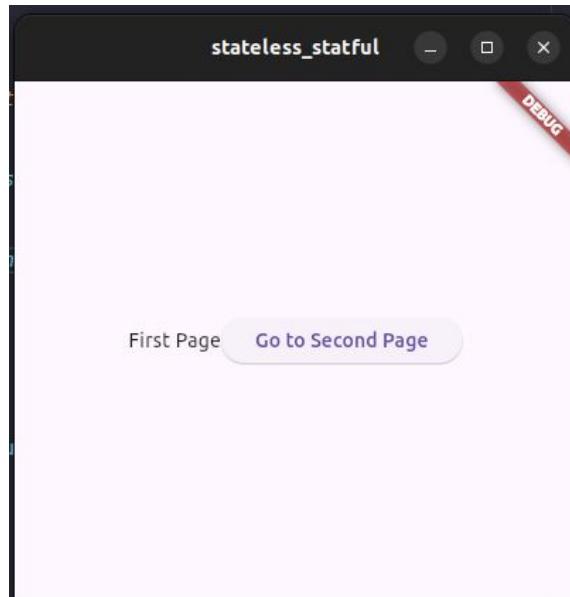
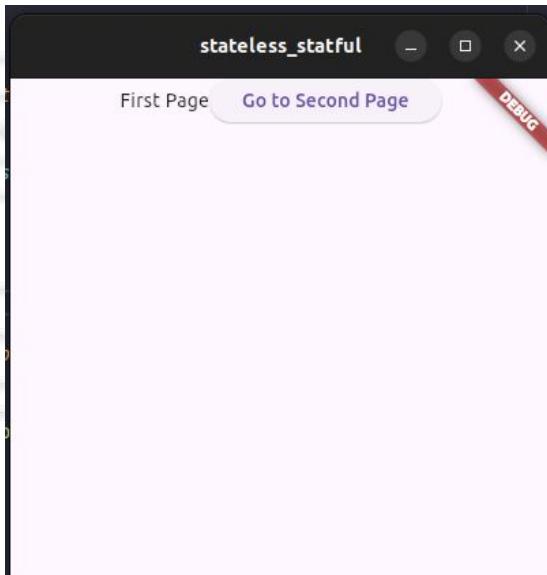
E para ficar no meio da tela?

- Posso usar uma Row dentro da Column?



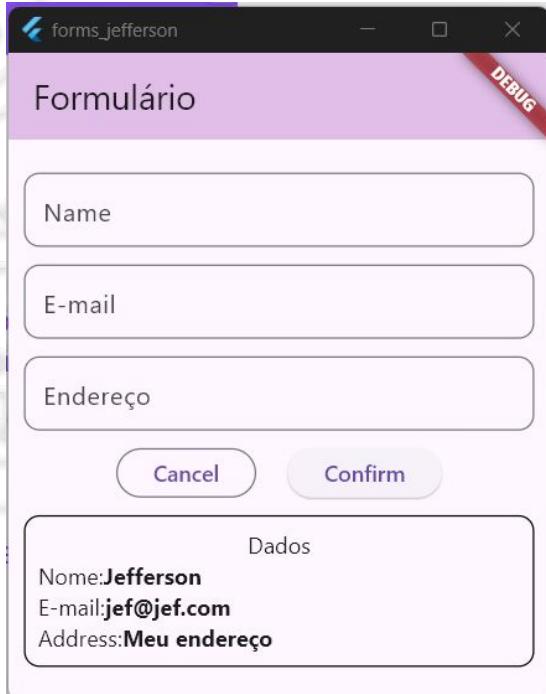
Organização da tela

- Existe outro meio para alinhar? Sim, o `crossAxisAlignment: CrossAxisAlignment.center`
- Só funciona se algum widget ocupar toda a tela em ambos os sentidos



Atividade

- Replique essa tela abaixo no arquivo `form_page.dart` dentro de `src/presenter/pages`



- Recuperar os dados digitados;
- Exibir esses dados nos campos de texto ao lado;
- Limpar os textos nos campos do formulário ao pressionar confirm;
- Caso pressione cancel somente limpar os campos;

Atividade

- Para usar uma AppBar e add espaço em branco nas bordas da exibição:

```
return Scaffold(  
    appBar: AppBar(  
        backgroundColor: Colors.amber, // Personalizar a cor  
        title: Text('data'),  
    ),  
    body: Padding( // Adiciona espaço em branco como uma borda  
        padding: const EdgeInsets.all(8.0),  
        child: Column(children: []));
```

- Para personalizar um container:

```
Container(  
    decoration: BoxDecoration(  
        border: Border.all(color: Colors.blue),  
        borderRadius: BorderRadius.circular(10)  
    ),  
    child: Center(child: Text(textScreen))),
```



Pages e Stores

Flutter

Pages e Stores

- Pages → Responsáveis por componentes na tela e suas navegações
- Stores → Responsáveis por lógicas que atualizam os estados da tela
- Na pasta lib:
 - dentro da pasta src/presenter/:
 - Crie uma nova pasta chamada stores
 - Crie um store para cada página criada dentro da pasta stores
- Exemplo: second_page.dart <> second_store.dart
- Adicionar dependência do Provider no pubspec.yaml
 - site: pub.dev

Pages e Store

- Provider
 - Gerenciamento de estado da tela
 - Usado nos stores para notificar mudanças
- O que muda no main?

```
void main() {  
  runApp(  
    MultiProvider(  
      providers: [ChangeNotifierProvider(create: (_) => SecondStore())],  
      child: MainApp(),  
    ),  
  );  
}
```

Pages e Store

- Dentro do store colocamos:

```
class SecondStore extends ChangeNotifier {  
  var _textScreen = 'Second page';  
  
  String get textScreen => _textScreen;  
  
  void updateText(String text) {  
    _textScreen = text;  
    notifyListeners();  
  }  
}
```

Pages e Store

- Como escutar as mudanças?

```
Text(context.watch<SecondStore>().textScreen),
```

- Como acessar as funções?

```
ElevatedButton(  
    onPressed: () {  
        context.read<SecondStore>().updateText(_controller.text);  
    },  
    child: Text('Update')  
);
```

Exercício

- Criem o store da FormPage
- Usem o provider para atualizar o texto na tela

Personalização

Flutter

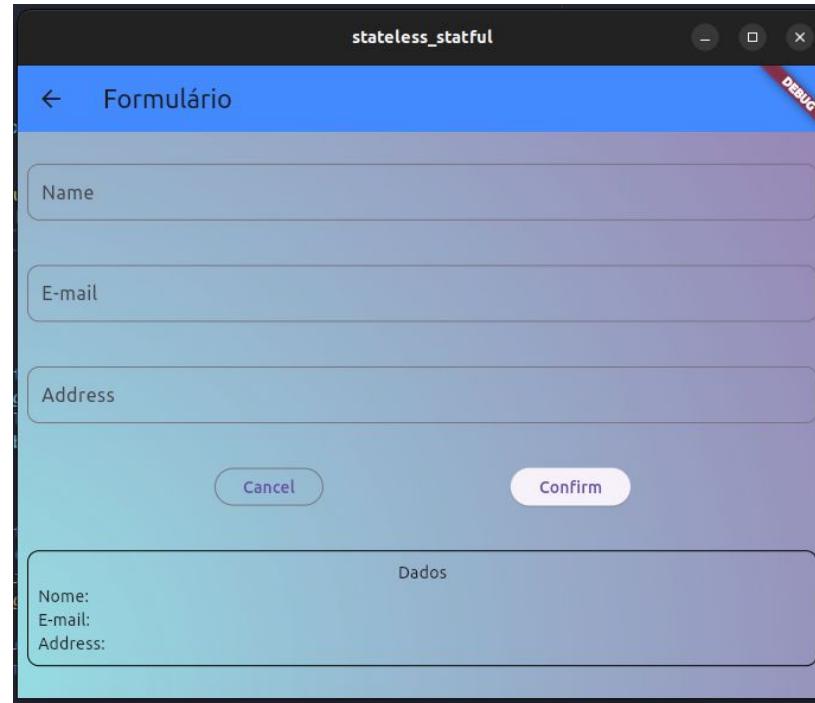
Personalização

- Todas as imagens, fontes, vídeos, entre outras mídias personalizadas podem ficar na pasta asset
- Vamos adicionar uma imagem em background?
 - Adicionar imagem ao assets
 - Adicionar assets no pubspec.yaml
 - Envolver o Padding na FormPage com o Widget Stack
 - Adicione antes do Padding:

```
Opacity(opacity: 0.5,  
        child: Image(  
            image: AssetImage('assets/image.png'),  
            repeat: ImageRepeat.repeat,  
            width: double.maxFinite,  
            height: double.maxFinite,,),,
```

Personalização

- Resultado



Comunicação Front e Back

Flutter

Comunicação Front-Back

Protobuf

Método para serializar dados estruturados.

Usado para armazenar e trocar dados de forma eficiente e compacta.

Como funciona:

- **Definição de mensagens:** Define os tipos de dados (arquivo .proto)
- **Compilação:** Gera o código para a linguagem;
- **Serialização:** Encapsula os dados para enviar;
- **Desserialização:** Desencapsula os dados recebidos.

Comunicação Front-Back

Protobuf

É preciso:

- Adicionar o **protobuf**, **protoc_plugin**, **fixnum** e **http** nas dependências
- Executar o comando: **dart pub global activate protoc_plugin**
- Adicionar o comando no terminal ao path
- Criar arquivos com extensão proto, exemplo: packages.proto
- Compilar os protos com: **protoc --dart_out=./ *.proto**
 - É preciso estar na mesma pasta onde os protos estão

Comunicação Front-Back

```
name: workshop_flutter
description: "A new Flutter project. "
publish_to: 'none'
version: 0.1.0

environment:
  sdk: ^3.8.1

dependencies:
  flutter:
    sdk: flutter
  provider: ^6.1.5+1
  http: ^1.5.0
  fixnum: ^1.1.1
  protobuf: ^4.2.0

dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^5.0.0
  protoc_plugin: ^22.5.0

flutter:
  uses-material-design : true
  assets:
    - assets/images/
```

pubspec.yaml

Comunicação Front-Back

Definição de mensagens

```
syntax = "proto2";

message Carro {
    required string modelo = 1;
    required string cor = 2;
}

message Motor {
    required int32 marcha = 1;
    required string velocidade = 2;
    optional bool ligado = 3;
}

message Pneu {
    required int32 rotacao = 1;
}
```

Criação do arquivo com extensão .proto

Comunicação Front-Back

Desserialização

```
import 'dart:typed_data';
import 'package:car_project/external/proto/packages.pb.dart';
// Classe responsável por adaptar e decodificar mensagens protobuf relacionadas ao motor
class EngineAdapter {
  // Método estático que decodifica uma mensagem protobuf do tipo Motor
  static Engine decodeProto(Uint8List encodedEngine) {
    try{
      // Tenta decodificar o buffer de bytes para uma instância de Motor
      final engine = Engine.fromBuffer(encodedEngine);
      return engine; // Retorna a instância de Motor decodificada
    }
    catch(e) {
      // Lança uma exceção personalizada em caso de erro na decodificação
      throw Exception('Erro ao decodificar o proto');
    }
  }
}
```

Comunicação Front-Back

Serialização

```
class EngineAdapter {  
    // Método de Dessorialização oculto  
    // Método estático que codifica uma instância de Motor em uma mensagem protobuf  
    static Uint8List encodeProto(Engine engine) {  
        final encodedUser = engine.writeToBuffer();  
        return encodedUser; // Retorna o buffer de bytes codificado  
    }  
}
```

Comunicação Front-Back

Vamos praticar o proto?

- Criar pasta em src/
 - external/
- Criar pastas em src/external/
 - adapters/
 - datasources/
 - protos/
- Em proto/ vamos criar um arquivo chamado package.proto
 - Dentro dele criem a mensagem User para a tela FormPage
 - Usar as variáveis string: name, email e address
- Compilem esse novo proto
 - No terminal: cd lib/src/external/protos/
 - No terminal: protoc --dart_out=./ *.proto

Comunicação Front-Back

Vamos praticar serialização e desserialização?

- Na pasta `src/external/adapters/`
 - criar arquivo `user_adapter.dart`
 - Colocar os métodos estáticos de serialização e desserialização.

Comunicação Front-Back

HTTP

Pacote utilizado para realizar requisições de forma assíncrona.

- **Funções de alto nível:**
 - `http.get()` → recuperar dados do servidor
 - `http.post()` → envia dados ao servidor
 - `http.put()` → atualiza um recurso
 - `http.delete()` → remove um recurso
- **Classes:** Oferecem mais controle sobre as requisições HTTP:
 - `http.Client` ← Usamos em nosso contexto
 - `http.Request`
 - `http.Response`

Comunicação Front-Back

No Flutter, podemos organizar essas chamadas do http usando a classe nomeada como datasource, ficando na pasta src/external/datasources

As rotas são combinadas com o backend:

- Neste caso foi:
 - Endereço do servidor:
<http://127.0.0.1>
 - Porta: 8000
 - Rota: /get-state

```
import 'dart:typed_data';
import 'package:http/http.dart' as http;
class EngineDatasource {
  final client = http.Client();
  // Método para recuperar o estado do motor na api
  Future<Engine> getState() async {
    try {
      // Faz uma requisição GET para o endpoint especificado
      final response = await client.get(
        Uri.parse('http://127.0.0.1:8000/get-state'),
      );
      // Retorna os bytes se a requisição foi bem-sucedida
      if (response.statusCode == 200) {
        return EngineAdapter.decodeProto(response.bodyBytes);
      } else {
        throw Exception('Server can\'t process request');
      }
    } catch (e) {
      throw Exception('Can\'t connect to server');
    }
  }
}
```

Comunicação Front-Back

No EngineStore, usamos uma função assíncrona para fazer a chamada ao backend.

- Instanciamos o datasource
- Criamos o método assíncrono para tentar se comunicar com o backend
- caso dê erro, capturamos isso na variável de mensagem de erro

```
class EngineStore extends ChangeNotifier {  
  final engineDatasource = EngineDatasource();  
  
  var _engine = Engine();  
  Engine get engine => _engine;  
  
  String errorMessage = '';  
  
  Future<void> getState() async {  
    try {  
      final engine = await engineDatasource.getState();  
      _engine = engine;  
    } catch (e) {  
      errorMessage = e.toString();  
    }  
  }  
}
```

Comunicação Front-Back

Vamos praticar o HTTP?

- Vamos criar o UserDatasource e fazer os métodos:
 - showInformations: Não envia nada na rota '/show-informations' e retorna um User, usando o método get do http
 - updateInformations: Envia um user encoded, não recebe nada na rota '/update-informations', retorna um bool caso o response.statusCode == 200, usando o método post do http
- Vamos atualizar o FormStore com os métodos:
 - showInformations: Não retorna nada, e tenta atualizar as informações usando o que foi recebido do datasource
 - updateInformations: Não retorna nada, recebe as informações de nome, email e endereço, caso o retorno seja true, atualiza as informações exibidas.
- Chamar método showInformations no initState da página (lembrar de fazer o override)

Observações:

- Criem uma variável para receber o erro, quando houver

Projeto final

Flutter

Projeto final

Antes de tudo vamos praticar:

- Uso da documentação oficial
 - Exemplo de Gridview → Visualização em grade
 - Exemplo de TabBar →Uso de abas

Projeto final

Esse projeto tenta simular um aluguel de filmes de forma bem simplificada

O objetivo é usar todos os conceitos aprendidos e pesquisar algumas soluções

É preciso que as cores e o design sejam próximas

É preciso fazer as chamadas para api

Estruturem o projeto de forma semelhante ao que foi passado

A api e os assets necessários serão passados

As rotas e protos estão na api

Projeto final

O que tem no UserStore?

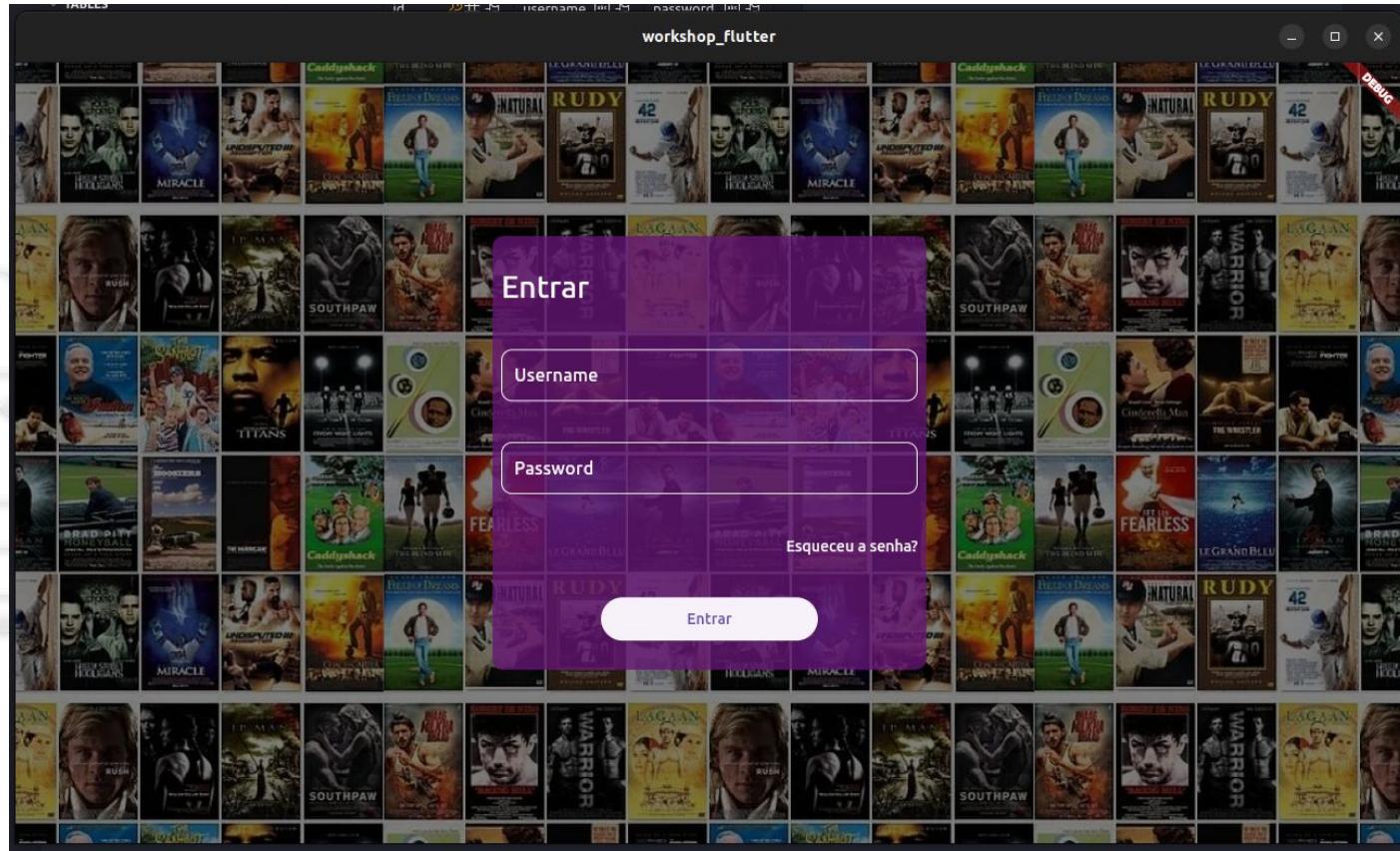
```
void initUser(User user) {this.user = user;}  
Future<void> getAvailableMovies() async {}  
Future<void> getRentalMovies() async {}  
Future<bool> rentalMovie(Movie movie) async {}  
Future<bool> watchMovie(Movie movie) async {}
```

Lembrem que é possível usar () async {} no botão da interface

Quando utilizar await no botão, é boa prática verificar se o contexto da tela ainda existe com o comando context.mounted ← isso retorna um booleano

Na api já existe o usuário joao, com senha 1234, cadastrado

Telas



Telas

workshop_flutter

joao

Available movies

Movies rental

DEMO

R\$ 9.90	R\$ 9.80	R\$ 8.80	R\$ 10.80
R\$ 12.80	R\$ 15.80	R\$ 8.80	R\$ 8.80

Telas

workshop_flutter

DEBUG

A movie poster for the film "Adão Negro". It features a close-up of a bald man's face with a determined expression. In the background, there are other characters including a woman in a green outfit and a man in a blue suit. A large eagle is perched on a hand in the foreground. The title "PODER NASCIDO DA FURIA" and "ADÃO NEGRO" are at the bottom, along with the release date "20 DE OUTUBRO" and "SOMENTE NOS CINEMAS".

Adão Negro

Há cerca de 5.000 anos, o escravo Teth-Adam recebe poderes divinos, mas é aprisionado por usá-los de forma vingativa. Ele é libertado no presente e sua moral ambígua o coloca em conflito com a Sociedade da Justiça da América. Adão Negro deve decidir se será o destruidor ou o herói que o mundo precisa.

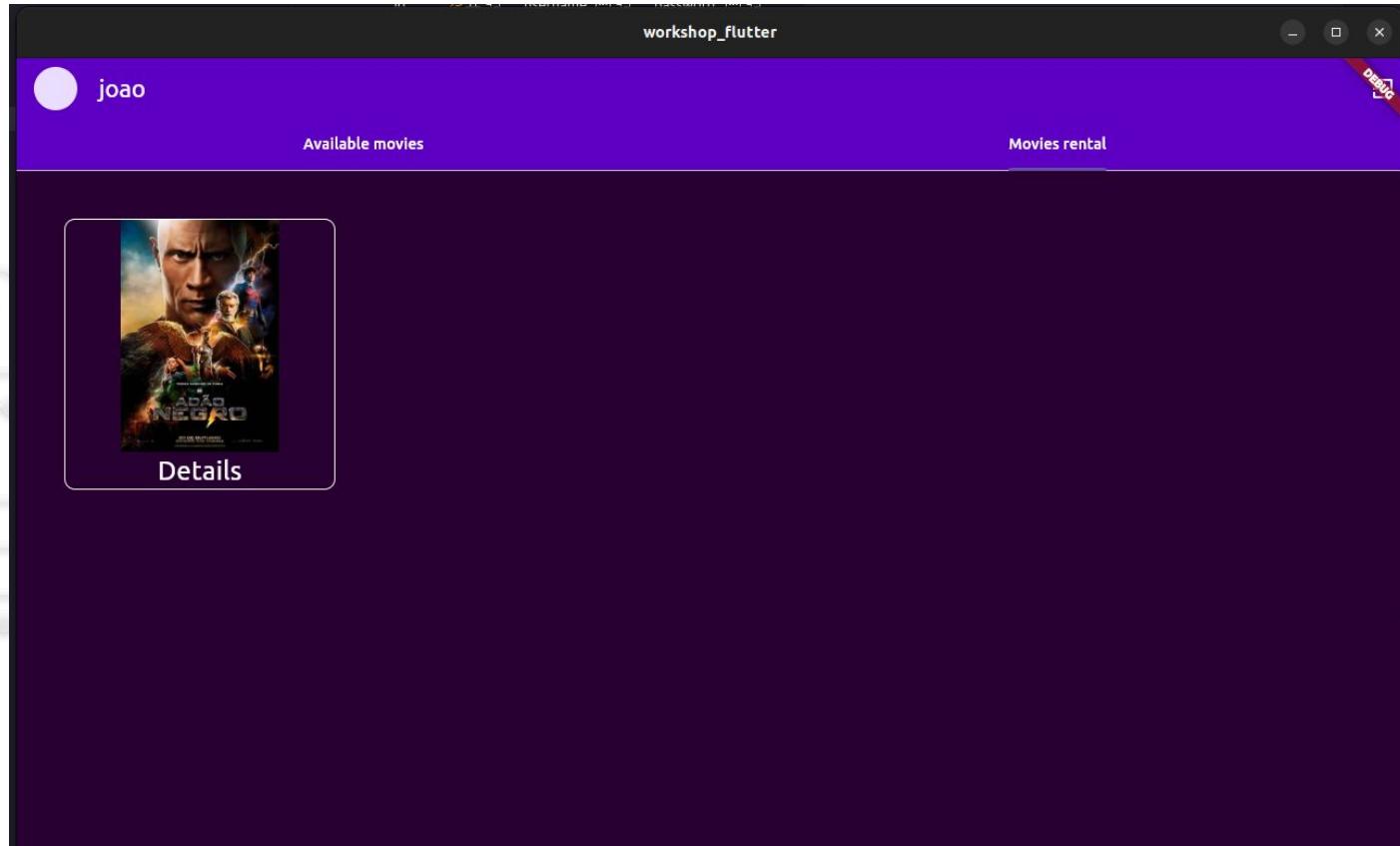
Year
2022

Director
Jaume Collet-Serra

R\$ 9.90

Cancel Rental

Telas



Telas

workshop_flutter

DEBUG

A movie poster for the film "Adão Negro". It features a close-up of a bald man's face with a determined expression. In the background, there are other characters including a woman in a green outfit and a man in a blue suit. A large eagle is perched on a hand in the foreground. The title "PODER NASCIDO DA FURIA" and "ADÃO NEGRO" are at the bottom, along with the release date "20 DE OUTUBRO" and "SOMENTE NOS CINEMAS".

Adão Negro

Há cerca de 5.000 anos, o escravo Teth-Adam recebe poderes divinos, mas é aprisionado por usá-los de forma vingativa. Ele é libertado no presente e sua moral ambígua o coloca em conflito com a Sociedade da Justiça da América. Adão Negro deve decidir se será o destruidor ou o herói que o mundo precisa.

Year
2022

Director
Jaume Collet-Serra

R\$ 9.90

Cancel Watch

Telas

