

Assíncrono e Armazenamento Offline

Usabilidade, Desenvolvimento Web, Mobile e Jogos

Prof. Gustavo Custodio
gustavo.custodio@anhembi.br



Assíncrono e Armazenamento Off-
line

Programação Assíncrona

Programação Assíncrona

- Quando você escreve um código, você espera que as instruções rodem de forma sequencial.

```
1  int x = 5;  
2  int y = x * 2;
```

- O esperado é que o valor de y será 10 porque a segunda linha **espera** a primeira linha terminar.

Programação Assíncrona

- Se uma linha demora para ser executada, o aplicativo “trava”, não permitindo input do usuário.
 - Por isso, na maioria das linguagens de programação modernas, incluindo Dart, você pode realizar **operações assíncronas**.
 - Dessa forma, o usuário pode continuar interagindo com o app enquanto ele realiza uma operação complicada.
- **Operações assíncronas não afetam a linha de execução principal.**

Futures

- Em Dart e Flutter, é possível escrever código assíncrono utilizando **Futures**.
- Um Future representa o resultado de uma operação assíncrona e possui dois estados: completo e incompleto.
- Um Future possui a sintaxe: `Future<Tipo>`
 - onde o Tipo corresponde ao retorno esperado da operação assíncrona.

Futures

- Vamos utilizar um exemplo onde adquirimos informação da API do *Google Books*.
- Para fazer isso, utilizamos a biblioteca `http` do Dart.
 - Mas antes, ela precisa ser adicionada em nosso projeto.
- Primeiro, crie um projeto no Flutter.

Futures

- Abra o arquivo `pubspec.yaml`.
- Procure a linha `dependencies`.
- E adicione o `http` versão 0.13.0 conforme mostrado abaixo:

```
1 dependencies:  
2   flutter:  
3     sdk: flutter  
4   http: ^0.13.0
```

- Em seguida, salve as alterações.

Futures

- Vamos criar nossa classe principal no arquivo main:

```
1  import 'package:flutter/material.dart';
2  import 'tela_google_books.dart';
3
4  void main() {
5    runApp(const MyApp());
6  }
7
8  class MyApp extends StatelessWidget {
9    const MyApp({Key? key}) : super(key: key);
10
11    // This widget is the root of your application.
12    @override
13    Widget build(BuildContext context) {
14      return MaterialApp(
15        title: 'Flutter Demo',
16        theme: ThemeData(
17          primarySwatch: Colors.green,
18        ),
19        home: PaginaFutura(),
20      );
21    }
22  }
```


Futures

- Em seguida crie um arquivo chamado `tela_google_books.dart`.
 - Realize os imports necessários:

```
1  import 'package:flutter/material.dart';  
2  import 'dart:async';  
3  import 'package:http/http.dart';  
4  import 'package:http/http.dart' as http;
```

Futures

- Crie um StatefulWidget chamado PaginaFutura:

```
1 class PaginaFutura extends StatefulWidget {  
2   const PaginaFutura({Key? key}) : super(key: key);  
3  
4   @override  
5   State<PaginaFutura> createState() => _PaginaFuturaState();  
6 }  
7  
8 class _PaginaFuturaState extends State<PaginaFutura> {  
9   // Retorna o resultado do conteúdo procurado na web  
10  String resultado = "";  
11  
12  @override  
13  Widget build(BuildContext context) {  
14    return Scaffold(  
15      appBar: AppBar(  
16        title: Text("Testando Futures"),  
17      ),  
18      body: Center(  
19        child: Column(children: [  
20          Spacer(), // adiciona um espaço entre elementos na coluna  
21          //_criarBotaoDeBusca(context),  
22          Spacer(),  
23          //Text(resultado.toString()),  
24        ]),  
25      ),  
26    );  
27  }
```

Futures

- Agora criaremos uma função chamada `getDadosAPI()`.
 - Ela será responsável por adquirir os dados em json da API do Google Books.

```
1 Future<Response> getDadosAPI() async {  
2     final String dominio = 'www.googleapis.com';  
3     final String caminho = '/books/v1/volumes/junbDwAAQBAJ';  
4     Uri url = Uri.https(dominio, caminho);  
5     return http.get(url);  
6 }
```

- A função retorna uma `Future` que quando completa, “se transforma” em uma `Response`.

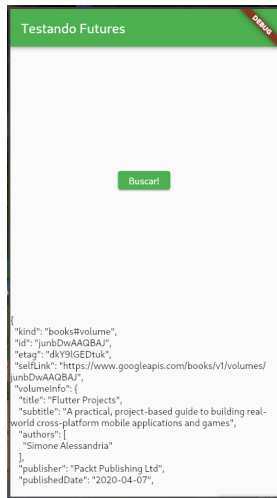
Futures

- Por último, criamos um botão que, quando pressionado, mostrará o texto recuperado da API em um widget de texto.

```
1 Widget _criarBotaoDeBusca(BuildContext context) {
2   return ElevatedButton(
3     child: Text('Buscar!'),
4     onPressed: () {
5       resultado = '';
6       setState(() {
7         resultado = resultado;
8       });
9       /* Chama getDadosAPI e quando a future for completada,
10        rode o código dentro do then */
11       getDadosAPI().then((valorRetorno) {
12         resultado = valorRetorno.body.toString().substring(0, 450);
13         setState(() {
14           resultado = resultado;
15         });
16       });
17     },
18   );
19 }
```

Futures

- Produzimos o seguinte resultado:
- No futuro aprenderemos como extrair os dados formatados de um json.



Futures

- O que aconteceu?
 - Quando clicamos no botão, chamamos a **função assíncrona** chamada `getDadosAPI`.
 - Quando o Future é completada, a função no `then` é invocada.

```
1  getDadosAPI().then((valorRetorno) {  
2      resultado = valorRetorno.body.toString().substring(0, 450);  
3      setState(() {  
4          resultado = resultado;  
5      });  
6  });
```

Futures

- O `then` não facilita a visibilidade do código.
 - Cria muitas estruturas aninhadas.
 - Uma alternativa ao `then` são as palavras chave `async` e `await`.

Async e Await

- Ao invés de utilizar, o then, vamos reescrever o trecho de código onde o Future é executado utilizando o await.

```
1   onPressed: ()async {
2     resultado = '';
3     setState(() {
4       resultado = resultado;
5     });
6     // Métodos assíncronas devem ter um await antes de serem chamados.
7     var valorRetorno = await getDadosAPI();
8     resultado = valorRetorno.body.toString().substring(0, 450);
9     setState(() {
10      resultado = resultado;
11    });
12  },
```


Async e Await

- Observe que a função anônima chamada pelo `onPressed` agora possui um `async` em sua assinatura.
 - Todos os métodos que utilizam a palavra chave `await` devem ser assíncronos (`async`).

Async e Await

- Lado a lado a diferença entre o then e o async / await:

```
1 Future<Response> getData() {  
2     String url = 'https://url.com';  
3     return http.get(url);  
4 }  
5  
6 void algumMetodo() {  
7     getData()  
8         .then((valor) {  
9         // Faça algo com o valor retornado.  
10    })  
11    }  
12 }
```

```
1 Future<Response> getData() {  
2     String url = 'https://url.com';  
3     return http.get(url);  
4 }  
5  
6 Future algumMetodo() async {  
7     var valor = await.getData();  
8     // Faça algo com o valor retornado.  
9 }
```

Programação Assíncrona

- O armazenamento offline utiliza programação assíncrona para armazenar conteúdo.
- Isso será feito utilizando o recurso SharedPreferences.



Assíncrono e Armazenamento Of-
fline

Armazenamento Offline

Shared Preferences

- O primeiro passo é realizar a importação do SharedPreferences.
 - Abra o arquivo `pubspec.yaml` e adicione o seguinte conteúdo em `dependencies`:

```
1      dependencies:  
2        flutter:  
3          sdk: flutter  
4        shared_preferences: ^2.0.15
```

Shared Preferences

- Adicione o import em seu projeto:

```
1  import 'package:shared_preferences/shared_preferences.dart';
```

- Crie uma classe que herda de StatefulWidget chamada MinhaPaginaPrincipal.
 - Adicione o atributo em _MinhaPaginaPrincipalState:
 - `int` contadorApp;

Armazenamento Offline

- Crie um método chamado `lerEscreverPreferencias`:
 - `Future lerEscreverPreferencias()` `async`.
- Dentro desse método, crie uma instância de `SharedPreferences`:
 - `SharedPreferences prefs = await SharedPreferences.getInstance();`

Armazenamento

- Ainda no mesmo método, tente ler o valor armazenado de contadorApp.
 - Se o valor é nulo, atribua 1.
 - Caso contrário, acrescente 1 no contador.

```
1  int? valorSalvo = prefs.getInt('contadorApp');
2  // Verifica se o contador salvo é nulo
3  if (valorSalvo == null) {
4      contadorApp = 1;
5  }else {
6      contadorApp = valorSalvo;
7      contadorApp++;
8  }
```


Armazenamento

- Atualize o valor de contadorApp armazenado:
 - `await prefs.setInt('contadorApp', contadorApp);`
- E atualize o valor na tela:

```
1   setState(() {  
2       contadorApp = contadorApp;  
3   })
```

Armazenamento

- Ainda na classe `_MinhaPaginaPrincipalState`, adicione o método `initState`:

```
1  @override
2  void initState() {
3      lerEscreverPreferencias();
4      super.initState();
5  }
```

Armazenamento

- No método build, adicione o seguinte código

```
1      body: Center(  
2        child: Column(  
3          mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
4          children: [  
5            Text('Você abriu o aplicativo ' +  
6              contadorApp.toString() + ' vezes.'),  
7            ElevatedButton(  
8              onPressed: (){},  
9              child: Text('Reiniciar Contador'),  
10           )  
11          ],  
12        ),  
13      ),
```

Armazenamento

- Por último, adicione um método para apagar os dados armazenados:

- Future deletarPreferencias()async

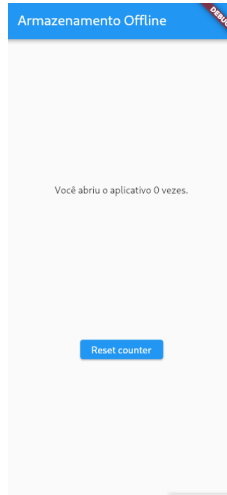
```
1 Future deletarPreferencias() async {  
2   SharedPreferences prefs = await SharedPreferences.getInstance();  
3   await prefs.clear();  
4   setState(() {  
5     contadorApp = 0;  
6   });  
7 }
```

- Adicione o método ao botão:

- onPressed: deletarPreferencias,

Armazenamento

- O resultado é um aplicativo que conta quantas vezes ele foi aberto.



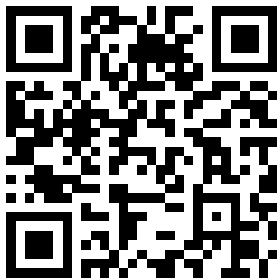
Referências

 Simone Alessandria and Brian Kayfitz.

Flutter Cookbook: Over 100 proven techniques and solutions for app development with Flutter 2.2 and Dart.

Packt Publishing Ltd, 2021.

Conteúdo



<https://gustavotcustodio.github.io/usabilidade.html>

Obrigado

gustavo.custodio@anhembi.br