Flutter Básico - Aula 3

Exercícios 9 a 12

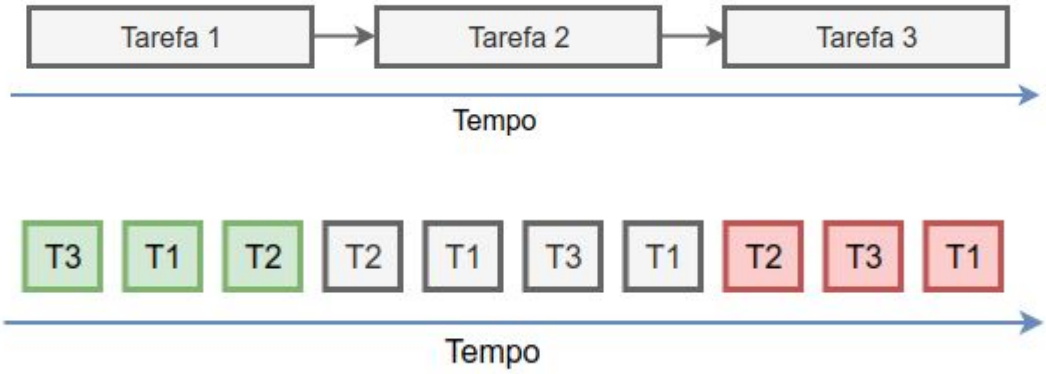
Fernando Duwe

v 1.0

Dicas de hoje

- iampawan/FlutterExampleApps - GitHub; Ver mais

- Funções assíncronas são executadas fora do fluxo padrão de execução de uma aplicação
- Enquanto aguarda o retorno, libera a aplicação para executar outras funções
- Utilizada para aguardar resultado de fontes externas



Exemplo de uma função assíncrona

```
Future<List> getAllPets() async {

    final dataList = await DbUtil.getData('pets');

    _petList = dataList.map((pets) => Pet.fromMap(pets)).toList();

    return _petList;
  }
```

Ver mais

async

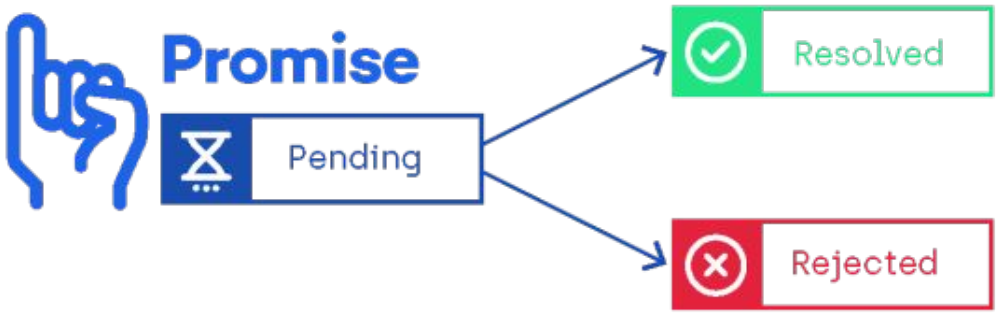
O async determina que um método será assíncrono, ou seja, não irá retornar algo imediatamente, então o aplicativo pode continuar a execução de outras tarefas enquanto o processamento não é finalizado.

await

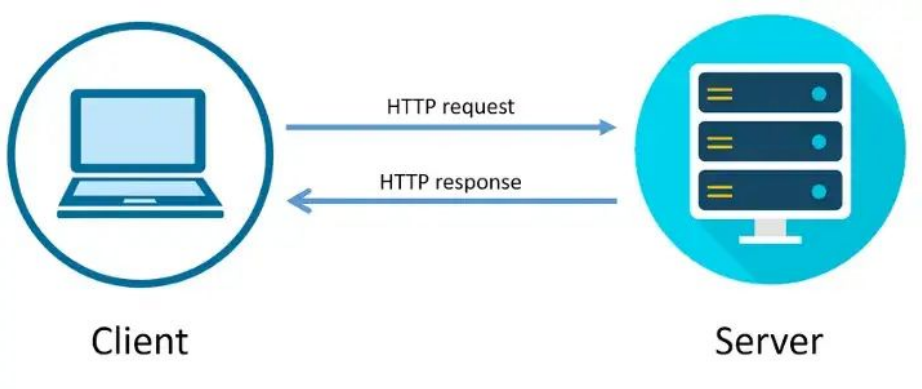
O await serve para determinar que o aplicativo deve esperar uma resposta de uma função antes de continuar a execução. Isso é muito importante pois há casos em que uma função depende do retorno de outra.

Future

O Future determina que uma função irá retornar algo no “futuro”, ou seja, é uma função que levará um tempo até ser finalizada.



- HTTP: Protocolo de comunicação
- Requisição: Solicitação realizada para o servidor através do navegador, app, etc
- Enviada para o servidor, o servidor processa a requisição e devolve um resultado



- Primeiro, vamos importar o pacote em nosso projeto, através do arquivo pubspec.yaml
- Depois, vamos ao código!

```
import 'package:http/http.dart' as http;

Future<Map> Internal_GetWineList() async {
    Uri vrURL = Uri.https("api.sampleapis.com", "/wines/reds");

    http.Response result = await http.get(vrURL);

    return json.decode(result.body);
}
```

- Widget utilizado para construir um layout a partir de uma função assíncrona

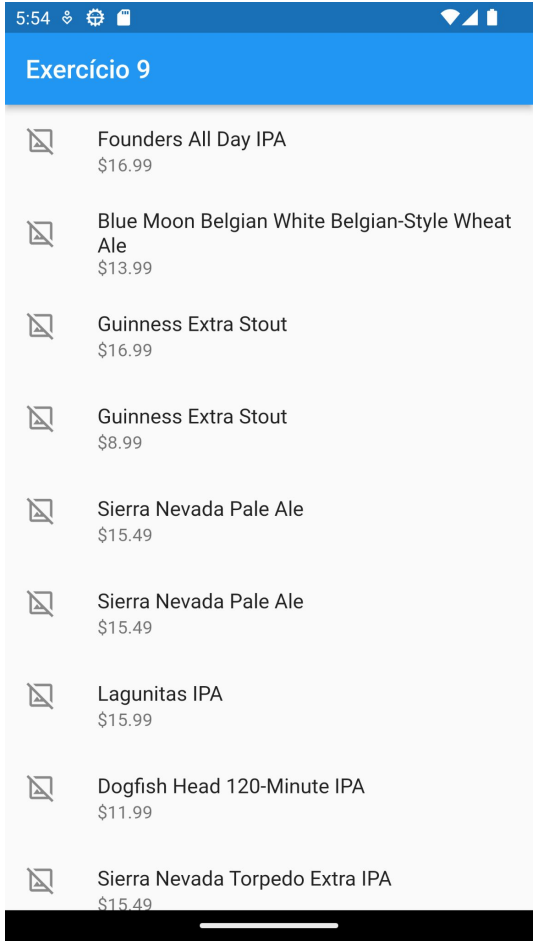
```
FutureBuilder(
          future: this.Internal_GetWineList(),
          builder: (context, snapshot) {
            switch (snapshot.connectionState) {
              case ConnectionState.none:
              case ConnectionState.waiting:
                return Container(
                  child: CircularProgressIndicator(),
                );

              default:
                return this.Internal_ShowList(snapshot.data!);
            }
```

Exercício 9

Ver mais

- Vamos criar um app que busque e exiba uma lista de tipos de cerveja
- A lista está na API https://api.sampleapis.com/beers/ale
- Você deverá exibir o nome e o preço da cerveja



Serialização

Transformar uma estrutura de dados em uma *string*.

```
class User {
  final String name;
  final String email;

  User(this.name, this.email);

  Map<String, dynamic> toJson() => {
        'name': name,
        'email': email,
      };
}
```

Deserialização

Transformar uma string em uma estrutura de dados.

```
class User {
  final String name;
  final String email;

  User(this.name, this.email);

  User.fromJson(Map<String, dynamic> json)
        : name = json['name'],
          email = json['email'];

}
```

- Vamos nos cadastrar no site
- Criar um App, do tipo API
- Ler a documentação

Exercício 10

Ver mais

- Utilizando a API da Giphy, vamos criar um buscador de gifs
- Nesse exercício, liste os gifs que estão em *trending*
- Vamos utilizar os fontes escritos nesse exercício também para o próximo

- Utilizando os fontes do exercício 10, vamos implementar a pesquisa de gifs
- Na AppBar, adicione um botão de Pesquisa
- Ao clicar nesse botão, vá para uma nova tela para que o usuário efetue a pesquisa
- Clicando em pesquisar, efetua uma requisição de pesquisa
- Caso haja uma pesquisa sendo efetuada, troque o botão de pesquisa por um botão de limpeza da pesquisa

Exercício 11 (cont)

Ver mais