



# Flutter Básico - Aula 3

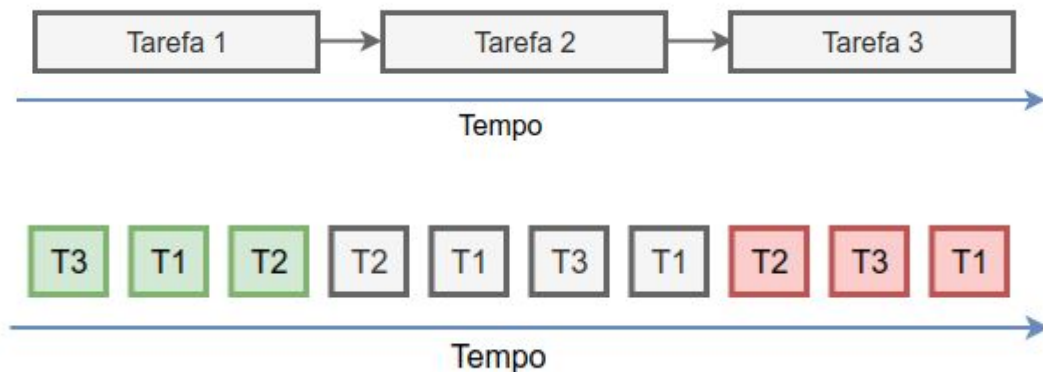
Exercícios 9 a 12



- [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;  
}
```



### **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.

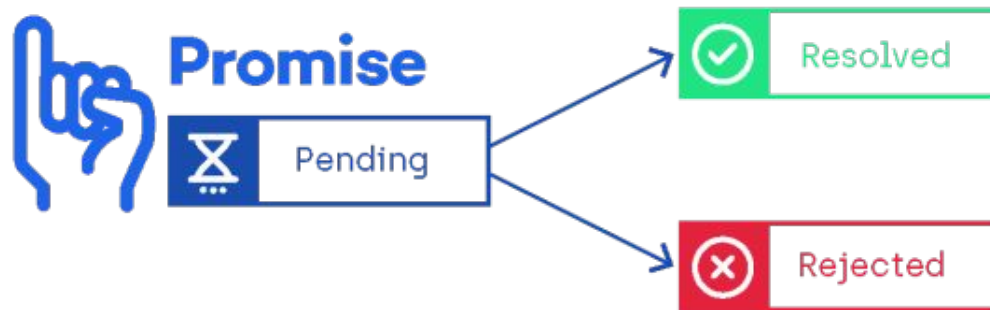
### **await**

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.



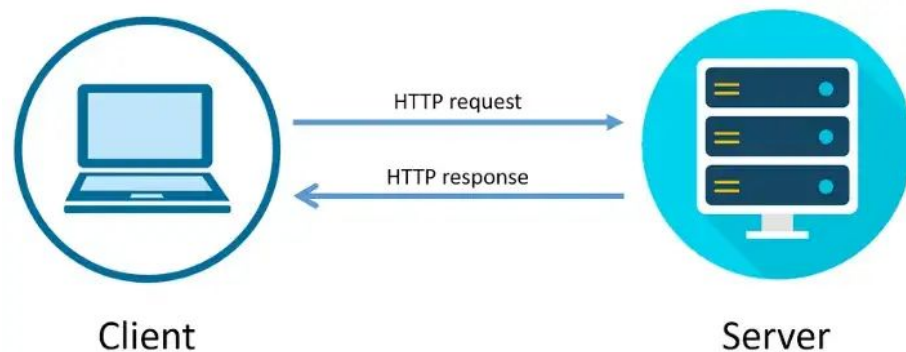
## Future

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);
}
```

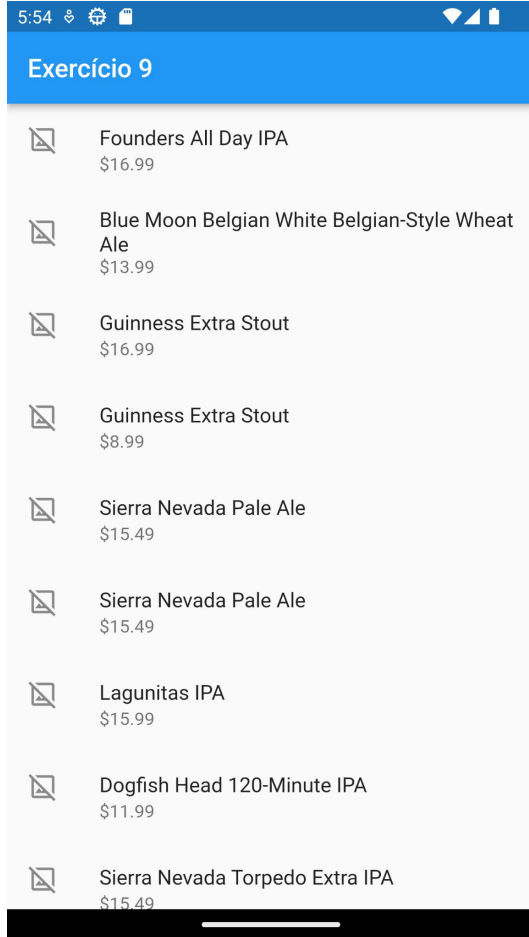




- Widget utilizado para construir um layout a partir de uma função assíncrona

```
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!);  
    }  
  }
```

- 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.

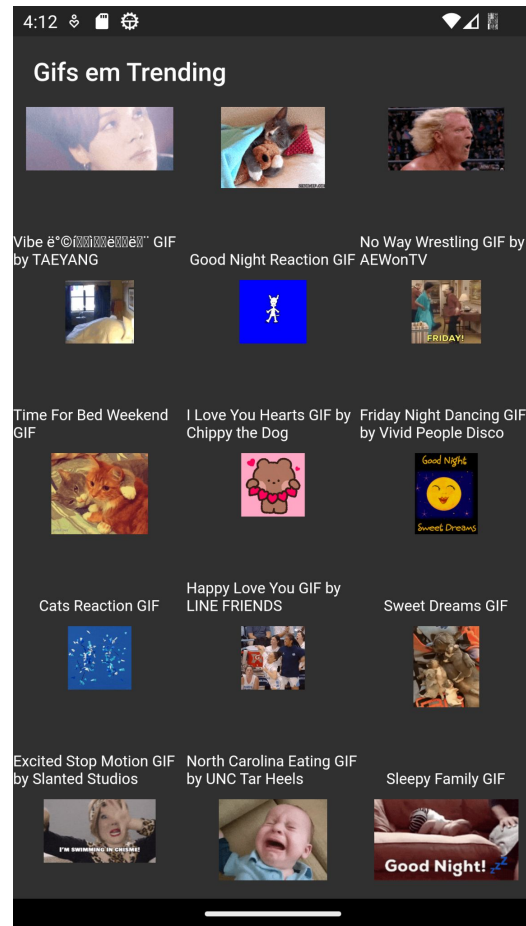
```
class User {  
    final String name;  
    final String email;  
  
    User(this.name, this.email);  
  
    User.fromJson(Map<String, dynamic> json)  
        : name = json['name'],  
          email = json['email'];  
  
}
```



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



- 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

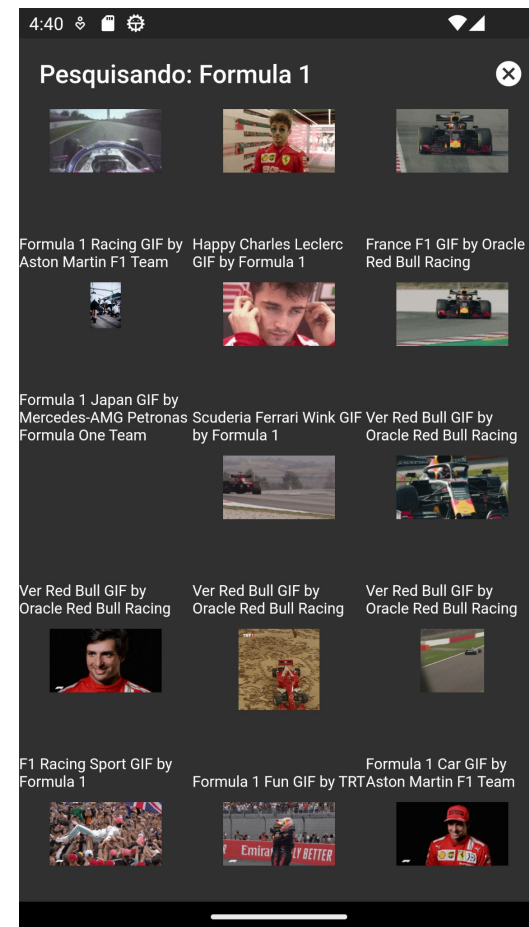
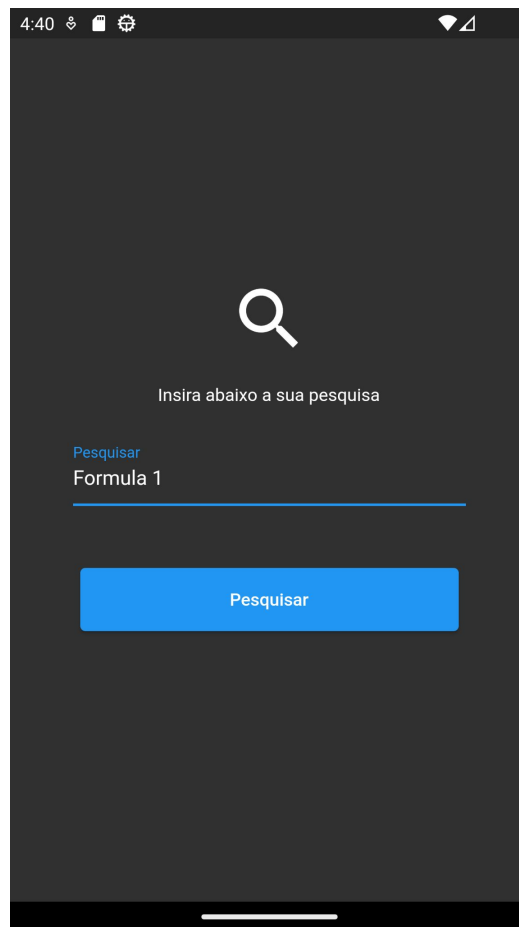
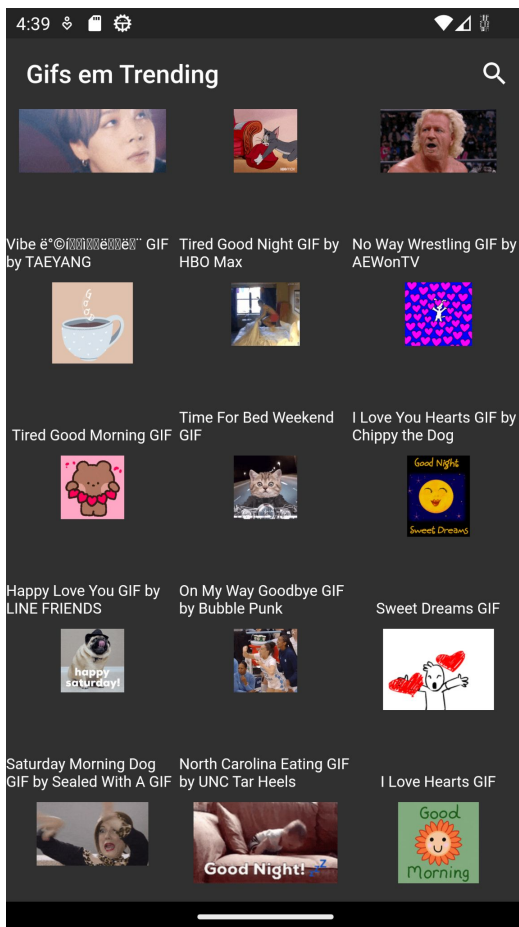




- Utilizando os fontes do exercício 10, vamos implementar a pesquisa de gifs
- Na AppBar, adicione um botão de Pesquisa
- Ao clicar nesse botão, vá para uma nova tela para que o usuário efetue a pesquisa
- Clicando em pesquisar, efetua uma requisição de pesquisa
- Caso haja uma pesquisa sendo efetuada,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

## Exercício 11 (cont)

[Ver mais](#)







- Neste aplicativo, vamos criar um formulário de cadastro de novos usuários
- Esses usuários serão integrados com uma API externa
- Após o retorno da API, o usuário deverá ser redirecionado para uma tela, avisando se deu tudo certo ou ocorreu um erro
- Você pode verificar se ocorreu um erro com a propriedade `statuscode = 200`, da resposta da requisição



## Modelo de requisição (cadastro do usuário)


The screenshot shows a REST client interface with a dark theme. At the top, the method is set to **POST** and the URL is `http://autc.com.br/api/index.php`. A **Send** button is on the right. Below the URL bar, there are tabs for **Multipart** (with a count of 3), **Auth**, **Query**, **Headers** (with a count of 1), and **Docs**. Under the **Multipart** tab, there are buttons for **Add**, **Delete All**, and **Toggle Description**. The main area displays three form fields:

| Field Name | Value | Dropdown | Checkmark | Delete |
|------------|-------|----------|-----------|--------|
| username   | value | ▼        | ☑         | 🗑      |
| name       | value | ▼        | ☑         | 🗑      |
| email      | value | ▼        | ☑         | 🗑      |

Lista com os usuários cadastrados:

[Ver mais](#)

6:13



Cadastre um novo usuário. Preencha os dados abaixo e clique em Cadastrar

Seu nome de usuário

---

Nome

---

E-mail

---

Cadastrar

