



**FLUTTER**

**TEMA: WEBSERVICES**



## INTRODUÇÃO

# CORES

---

Crie um novo projeto **flutter**:

```
flutter create --org br.com.heiderlopes dragon_ball_app
```



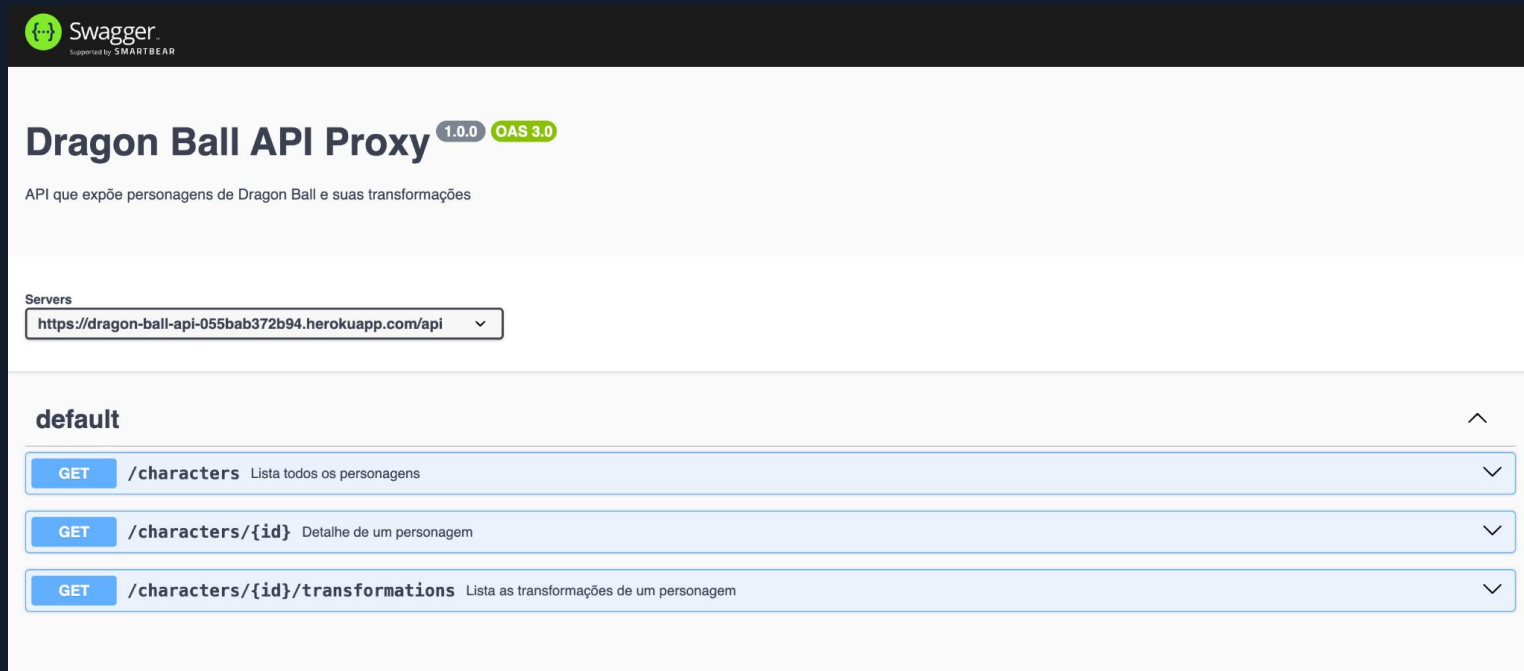
# FLUTTER

## DOCUMENTAÇÃO DA API



# DRAGON BALL APP

## A API



The image shows a Swagger UI interface for the 'Dragon Ball API Proxy'. At the top, the Swagger logo is visible with the text 'Supported by SMARTBEAR'. The main title is 'Dragon Ball API Proxy' with version '1.0.0' and 'OAS 3.0' tags. Below the title is a description: 'API que expõe personagens de Dragon Ball e suas transformações'. A 'Servers' section contains a dropdown menu with the URL 'https://dragon-ball-api-055bab372b94.herokuapp.com/api'. The 'default' section lists three endpoints: 'GET /characters' (Lista todos os personagens), 'GET /characters/{id}' (Detalhe de um personagem), and 'GET /characters/{id}/transformations' (Lista as transformações de um personagem). Each endpoint is in a blue box with a dropdown arrow on the right.

Swagger  
Supported by SMARTBEAR

## Dragon Ball API Proxy 1.0.0 OAS 3.0

API que expõe personagens de Dragon Ball e suas transformações

Servers

### default ^

- GET** **/characters** Lista todos os personagens
- GET** **/characters/{id}** Detalhe de um personagem
- GET** **/characters/{id}/transformations** Lista as transformações de um personagem

<https://dragon-ball-api-055bab372b94.herokuapp.com/api-docs/>

**FLUTTER**

CRIANDO O MODELO



DRAGON BALL APP

## CRIANDO O MODELO

---

Dentro de **lib** crie uma pasta chamada **models**.

Dentro da pasta **models** crie um arquivo chamado **character.dart**.

Esta classe será o mapeamento do nosso objeto baseado no webservice.

```
class Character {  
  final int id;  
  final String name;  
  final String image;  
  
  Character({  
    required this.id,  
    required this.name,  
    required this.image,  
  });  
}
```

## DRAGON BALL APP

# CRIANDO O MODELO

---

Dentro da character.dart adicione os métodos responsáveis em mapear um json em objeto e vice-versa.

```
// Factory para converter JSON -> Objeto
factory Character.fromJson(Map<String, dynamic>
json) {
    return Character(
        id: json['id'],
        name: json['name'],
        image: json['image'],
    );
}

// Método para converter Objeto -> JSON
Map<String, dynamic> toJson() {
    return {
        'id': id,
        'name': name,
        'image': image,
    };
}
```



**FLUTTER**

CRIANDO O SERVICES



## DRAGON BALL APP

# ADICIONANDO A LIB

Para consumir os dados de um webservice é necessário adicionar a dependência da lib http.

Abra o arquivo **pubspec.yaml** e adicione:

```
http: ^1.2.2
```

```
dependencies:  
  flutter:  
    sdk: flutter  
  http: ^1.2.2  
  
  # The following adds the Cupertino Icons font to your application.  
  # Use with the CupertinoIcons class for iOS style icons.  
  cupertino_icons: ^1.0.8  
  
dev_dependencies:  
  flutter_test:  
    sdk: flutter
```

## DRAGON BALL APP

# CRIANDO A APISERVICE

---

Dentro de **lib** crie uma pasta chamada **services**.

Dentro de **services** crie um arquivo chamado **api\_service.dart** e adicione o código ao lado.



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

class ApiService {
  static const String _baseUrl =
    'https://dragon-ball-api-055bab372b94.herokuapp.com/api';

  static Future<List<Character>> fetchCharacters() async {
    final uri = Uri.parse('${_baseUrl}/characters');
    final resp = await http.get(uri);

    if (resp.statusCode != 200) {
      throw Exception('Erro ao chamar API: ${resp.statusCode}');
    }

    List<dynamic> items = json.decode(resp.body);

    return items
      .map((e) => Character.fromJson(e))
      .whereType<Character>()
      .toList();
  }
}
```

## DRAGON BALL APP

# CRIANDO A APISERVICE

Criamos uma classe utilitária chamada **ApiService**, responsável por centralizar chamadas à API.

**\_baseUrl** é a URL base do servidor.

O **static const** significa que a **URL** é **constante** e pertence à classe, não a um objeto dela.

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

class ApiService {
  static const String _baseUrl =
    'https://dragon-ball-api-055bab372b94.herokuapp.com/api';

  static Future<List<Character>> fetchCharacters() async {
    final uri = Uri.parse('${_baseUrl}/characters');
    final resp = await http.get(uri);

    if (resp.statusCode != 200) {
      throw Exception('Erro ao chamar API: ${resp.statusCode}');
    }

    List<dynamic> items = json.decode(resp.body);

    return items
      .map((e) => Character.fromJson(e))
      .whereType<Character>()
      .toList();
  }
}
```

## DRAGON BALL APP

# MÉTODO DE BUSCA

O método **fetchCharacters**:

**static**: o método pertence à classe, não precisa criar instância

(**ApiService.fetchCharacters()** já funciona).

**Future<List<Character>>**: a função retorna assíncronamente (com Future) uma lista de objetos Character.

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

class ApiService {
  static const String _baseUrl =
    'https://dragon-ball-api-055bab372b94.herokuapp.com/api'

  static Future<List<Character>> fetchCharacters() async {
    final uri = Uri.parse('$_baseUrl/characters');
    final resp = await http.get(uri);

    if (resp.statusCode != 200) {
      throw Exception('Erro ao chamar API: ${resp.statusCode}');
    }

    List<dynamic> items = json.decode(resp.body);

    return items
      .map((e) => Character.fromJson(e))
      .whereType<Character>()
      .toList();
  }
}
```

## DRAGON BALL APP

# FAZENDO A REQUISIÇÃO

Aqui fazemos a chamada HTTP com GET.

- O await espera a resposta da API.
- O resultado (resp) contém:
  - resp.statusCode → código HTTP (200, 404, 500...)
  - resp.body → corpo da resposta (JSON vindo da API).

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

class ApiService {
  static const String _baseUrl =
    'https://dragon-ball-api-055bab372b94.herokuapp.com/api'

  static Future<List<Character>> fetchCharacters() async {
    final uri = Uri.parse('$_baseUrl/characters');

    final resp = await http.get(uri);

    if (resp.statusCode != 200) {
      throw Exception('Erro ao chamar API: ${resp.statusCode}');
    }

    List<dynamic> items = json.decode(resp.body);

    return items
      .map((e) => Character.fromJson(e))
      .whereType<Character>()
      .toList();
  }
}
```

## DRAGON BALL APP

# TRATANDO ERROS

---

Se a resposta não tiver 200 (sucesso), o código lança uma exceção.

Isso impede de continuar se a API falhar.

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

class ApiService {
  static const String _baseUrl =
    'https://dragon-ball-api-055bab372b94.herokuapp.com/api'

  static Future<List<Character>> fetchCharacters() async {
    final uri = Uri.parse('${_baseUrl}/characters');
    final resp = await http.get(uri);

    if (resp.statusCode != 200) {
      throw Exception('Erro ao chamar API: ${resp.statusCode}');
    }

    List<dynamic> items = json.decode(resp.body);

    return items
      .map((e) => Character.fromJson(e))
      .whereType<Character>()
      .toList();
  }
}
```

## DRAGON BALL APP

# CONVERTENDO O JSON

Aqui usamos `json.decode()` (precisa do `import 'dart:convert';`).

O retorno do endpoint é um array JSON ([  
{id:1,...}, {id:2,...} ]).

Então fazemos o parse e armazenamos  
como `List<dynamic>`.

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

class ApiService {
  static const String _baseUrl =
    'https://dragon-ball-api-055bab372b94.herokuapp.com/api'

  static Future<List<Character>> fetchCharacters() async {
    final uri = Uri.parse('$_baseUrl/characters');
    final resp = await http.get(uri);

    if (resp.statusCode != 200) {
      throw Exception('Erro ao chamar API: ${resp.statusCode}');
    }

    List<dynamic> items = json.decode(resp.body);

    return items
      .map((e) => Character.fromJson(e))
      .whereType<Character>()
      .toList();
  }
}
```



## DRAGON BALL APP

# CONVERTENDO EM OBJETO

**.map((e) => Character.fromJson(e)):**

percorre cada item do JSON e transforma em um objeto **Character**.

**.whereType<Character>():** garante que só objetos válidos **Character** entrem na lista.

**.toList():** transforma o iterável em uma lista **final (List<Character>)**.

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

class ApiService {
  static const String _baseUrl =
    'https://dragon-ball-api-055bab372b94.herokuapp.com/api'

  static Future<List<Character>> fetchCharacters() async {
    final uri = Uri.parse('${_baseUrl}/characters');
    final resp = await http.get(uri);

    if (resp.statusCode != 200) {
      throw Exception('Erro ao chamar API: ${resp.statusCode}');
    }

    List<dynamic> items = json.decode(resp.body);

    return items
      .map((e) => Character.fromJson(e))
      .whereType<Character>()
      .toList();
  }
}
```

**FLUTTER**

EXIBINDO OS PERSONAGENS



**FLUTTER**

CRIANDO O WIDGET DO PERSONAGEM




## DRAGON BALL APP

# CRIANDO O WIDGET

Como o app irá carregar imagens da internet será adicionada a dependência da lib **CachedImage** que irá ajudar no processo de realizar o cache da imagem deixando o app mais performático.

Abra o arquivo **pubspec.yaml** e adicione a dependência:

```
cached_network_image: ^3.3.1
```



```
dependencies:  
  flutter:  
    sdk: flutter  
  http: ^1.2.2  
  cached_network_image: ^3.3.1  
  
  # The following adds the Cupertino Icons font to your application.  
  # Use with the CupertinoIcons class for iOS style icons.  
  cupertino_icons: ^1.0.8
```

## DRAGON BALL APP

# CRIANDO O WIDGET

---

Crie uma pasta dentro de lib chamada **widgets**.

Dentro dele crie o arquivo chamado **character\_carousel\_item.dart**.

Adicione a estrutura base do **Widget**.



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

class CharacterCarouselItem extends StatelessWidget {
  final Character character;

  const CharacterCarouselItem({super.key, required
    this.character});

  @override
  Widget build(BuildContext context) {
    return Container(
      margin: const EdgeInsets.symmetric(
        horizontal: 16,
        vertical: 12,
      ), // margem do card
      width: double.infinity, // ocupa toda largura disponível no
        carrossel
      child: ,
    );
  }
}
```

DRAGON BALL APP

## CRIANDO O WIDGET

---

Adicione um **Card** no **Child** do **Container**.

```
Card(  
  elevation: 8,  
  shape: RoundedRectangleBorder(borderRadius:  
    BorderRadius.circular(16)),  
  child: Padding(  
    padding: const EdgeInsets.all(16), // padding interno  
    menor  
    child: Column(children: [  
      // Aqui serão adicionados os elementos do personagem  
    ]),  
  ),  
),
```

## DRAGON BALL APP

# CRIANDO O WIDGET

O primeiro elemento que será inserido na coluna é a imagem:

O **Hero** é um widget usado para criar **animações de transição de telas**. O tag é um identificador único. Quando você navega para outra tela que também tem um **Hero** com a mesma tag, o Flutter faz uma animação da imagem de uma tela para outra.

O **CachedNetworkImage**: carrega imagens da internet e faz cache automático.

```
Expanded(  
  child: Hero(  
    tag: 'character_${character.id}',  
    child: CachedNetworkImage(  
      imageUrl:  
        character.image.isNotEmpty  
          ? character.image  
          :  
            'https://placeholder.co/600x400?text=No%20Image',  
      fit: BoxFit.contain,  
      placeholder:  
        (context, url) =>  
          const Center(child:  
CircularProgressIndicator()),  
      errorWidget:  
        (context, url, error) => const  
Icon(Icons.error),  
    ),  
  ),  
)
```

## DRAGON BALL APP

# CRIANDO O WIDGET

---

Após o **Expanded** da imagem adicione um espaço e um **text** para exibir o nome do personagem:

```
const SizedBox(height: 8),
Text(
  character.name,
  textAlign: TextAlign.center,
  style: const TextStyle(
    fontSize: 18,
    fontWeight: FontWeight.bold,
    color: Colors.orange,
  ),
),
```



# FLUTTER

## CRIANDO A HOME



## DRAGON BALL APP

# CRIANDO A HOME

---

Crie uma pasta chamada **screens** dentro de **lib** e adicione um arquivo chamado **home\_screen.dart**.

No **initState** está sendo chamado o método para carregar a **lista de personagens**

```
class HomeScreen extends StatefulWidget {  
  const HomeScreen({super.key});  
  
  @override  
  State<HomeScreen> createState() => _HomeScreenState();  
}  
  
class _HomeScreenState extends State<HomeScreen> {  
  late Future<List<Character>>> _futureCharacters;  
  
  @override  
  void initState() {  
    super.initState();  
    _futureCharacters = ApiService.fetchCharacters();  
  }  
  
  @override  
  Widget build(BuildContext context) {  
    return Container();  
  }  
}
```

## DRAGON BALL APP

# CRIANDO A HOME

---

Altere o build para ter um **Scaffold**. Dentro do body foi adicionado o **FutureBuilder** que irá alterar de acordo com o **Future** criado para listar os personagens.

```
return Scaffold(  
  appBar: AppBar(  
    title: Text("DRAGON BALL", style: TextStyle(color: Colors.black)),  
    centerTitle: true,  
    backgroundColor: Color.fromARGB(255, 228, 172, 16),  
  ),  
  body: FutureBuilder<List<Character>>(  
    future: _futureCharacters,  
    builder: (context, snapshot) {  
      if (snapshot.connectionState == ConnectionState.waiting) {  
        return const Center(child: CircularProgressIndicator());  
      }  
      if (snapshot.hasError) {  
        return Center(child: Text('Erro: ${snapshot.error}'));  
      }  
      final chars = snapshot.data ?? [];  
      if (chars.isEmpty) {  
        return const Center(child: Text('Nenhum personagem  
encontrado'));  
      }  
  
      return Column(  
        children: [  
          //Aqui sera adicionado o carrossel  
        ],  
      );  
    },  
  ),  
);
```

## DRAGON BALL APP

# CRIANDO A HOME

---

Para criar o carrossel será adicionada a biblioteca **carousel\_slider**.

Abra o arquivo **pubspec.yaml** e adicione a **dependência necessária:**

```
carousel_slider: ^5.1.0
```

```
dependencies:
```

```
  flutter:
```

```
    sdk: flutter
```

```
  http: ^1.2.2
```

```
  cached_network_image: ^3.3.1
```

```
  carousel_slider: ^5.1.0
```

## DRAGON BALL APP

# CRIANDO A HOME

---

Adicione o **carrossel** dentro do **children** do **Column**

```
Expanded(  
  child: CarouselSlider.builder(  
    itemCount: chars.length,  
    itemBuilder: (context, index, realIdx) {  
      final ch = chars[index];  
      return Padding(  
        padding: const EdgeInsets.symmetric(  
          vertical: 12,  
          horizontal: 8,  
        ),  
        child: CharacterCarouselItem(character: ch),  
      );  
    },  
    options: CarouselOptions(  
      enlargeCenterPage: true,  
      autoPlay: true,  
      autoPlayInterval: const Duration(seconds: 3),  
      enableInfiniteScroll: true,  
      viewportFraction: 0.68,  
      height: double.infinity,  
      //enlargeStrategy: CenterPageEnlargeStrategy.height,  
    ),  
  ),  
),  
),
```

# FLUTTER

## CONFIGURANDO A MAIN



## DRAGON BALL APP

# CRIANDO A HOME

---

Abra o arquivo **main.dart** e ajuste-o para carregar a **Home**.

```
void main() {  
  runApp(const DragonBallApp());  
}  
  
class DragonBallApp extends StatelessWidget {  
  const DragonBallApp({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      title: 'DRAGON BALL APP',  
      theme: ThemeData.dark().copyWith(  
        scaffoldBackgroundColor: Colors.black,  
        appBarTheme: const AppBarTheme(centerTitle: true),  
      ),  
      debugShowCheckedModeBanner: false,  
      home: const HomeScreen(),  
    );  
  }  
}
```

**FLUTTER**

CRIANDO A TELA DE DETALHE






## DRAGON BALL APP

# CRIANDO O DETALHE

Crie uma arquivo chamado  
**detail\_screen.dart** dentro de **screens**.

Segue ao lado a estrutura da tela de  
detalhe.



```
class DetailScreen extends StatefulWidget {  
  final Character character;  
  const DetailScreen({super.key, required this.character});  
  @override  
  State<DetailScreen> createState() => _DetailScreenState();  
}  
  
class _DetailScreenState extends State<DetailScreen> {  
  @override  
  Widget build(BuildContext context) {  
    final character = widget.character;  
    return Scaffold(  
      appBar: AppBar(  
        title: Text(character.name, style: const TextStyle(color: Colors.black)),  
        iconTheme: const IconThemeData(color: Colors.black),  
        backgroundColor: const Color.fromARGB(255, 228, 172, 16),  
      ),  
      body: SingleChildScrollView(  
        padding: const EdgeInsets.all(20),  
        child: Center(  
          child: Column(  
            mainAxisAlignment: MainAxisAlignment.center,  
            children: [// Dados do personagem  
          ],),),  
        ),  
      );  
    }  
  }  
}
```

## DRAGON BALL APP

# CRIANDO O DETALHE

Dentro do **children** da **coluna** adicione a **imagem do personagem**:

```


    SizedBox(
      height: 320,
      child: Hero(
        tag: 'character_${character.id}',
        child: CachedNetworkImage(
          imageUrl:
            character.image.isNotEmpty
              ? character.image
              : 'https://placeholder.co/600x400?text=No%20Image',
          fit: BoxFit.contain,
          placeholder:
            (context, url) => const SizedBox(
              height: 320,
              child: Center(child: CircularProgressIndicator()),
            ),
          errorWidget:
            (context, url, error) => const SizedBox(
              height: 320,
              child: Center(child: Icon(Icons.error, size: 48)),
            ),
        ),
      ),
    ),
  ),
),

```

## DRAGON BALL APP

# CRIANDO O DETALHE

Após a imagem adicione o id e a descrição:



```
const SizedBox(height: 16),
```

```
// Nome e ID
```

```
Text(
```

```
  character.name,
```

```
  style: const TextStyle(
```

```
    fontSize: 28,
```

```
    fontWeight: FontWeight.bold,
```

```
  ),
```

```
),
```

```
const SizedBox(height: 6),
```

```
Text(
```

```
  'ID: ${character.id}',
```

```
  style: const TextStyle(color: Colors.grey),
```

```
),
```

```
const SizedBox(height: 18),
```

**FLUTTER**

CHAMANDO A TELA DE DETALHE



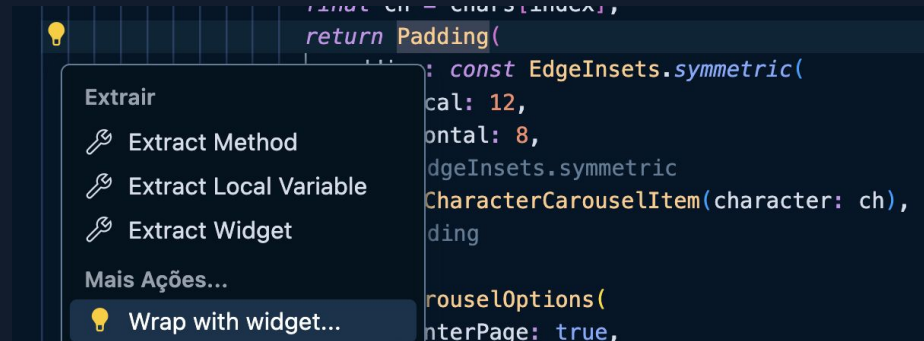
## DRAGON BALL APP

# CHAMANDO O DETALHE

Abra o arquivo `home_screen.dart`.

Posicione o cursor na linha do **Padding** do item.

Clique na lâmpada que irá aparecer e selecione **Wrap with widget**.



## CHAMANDO O DETALHE

Adicione o **GestureDetector** e o **onTap** e implemente o **Navigator** para navegar de uma tela para a outra.

```
GestureDetector(  
    onTap: () {  
        Navigator.push(context,  
            MaterialPageRoute(builder: (_) =>  
                DetailScreen(character: ch),  
            ),  
    );  
},  
child: Padding(  
    padding: EdgeInsets.all(10),  
    child: Text(character),  
));
```

**FLUTTER**

CARREGANDO MAIS DETALHES




## DRAGON BALL APP

# CRIANDO O MODELO

---

Dentro de **lib/models** crie um arquivo chamado **character\_detail.dart**.

Esta classe será o mapeamento do nosso objeto baseado no webservice.



```
class CharacterDetail {  
  final int id;  
  final String name;  
  final String ki;  
  final String maxKi;  
  final String race;  
  final String gender;  
  final String description;  
  final String image;  
  
  CharacterDetail({  
    required this.id,  
    required this.name,  
    required this.ki,  
    required this.maxKi,  
    required this.race,  
    required this.gender,  
    required this.description,  
    required this.image,  
  });  
}
```



## DRAGON BALL APP

# CRIANDO O MODELO

Dentro de **lib/models** crie um arquivo chamado **character\_detail.dart**.

Esta classe será o mapeamento do nosso objeto baseado no webservice.

```
// Factory para converter JSON -> Objeto
factory CharacterDetail.fromJson(Map<String, dynamic> json) {
  return CharacterDetail(
    id: json['id'],
    name: json['name'],
    ki: json['ki'],
    maxKi: json['maxKi'],
    race: json['race'],
    gender: json['gender'],
    description: json['description'],
    image: json['image'],
  );
}

// Método para converter Objeto -> JSON
Map<String, dynamic> toJson() {
  return {
    'id': id,
    'name': name,
    'ki': ki,
    'maxKi': maxKi,
    'race': race,
    'gender': gender,
    'description': description,
    'image': image,
  };
}
```

DRAGON BALL APP

## MÉTODO DO SERVICES

---

Adicione o método que irá retornar os detalhes do personagem no arquivo `api_service.dart`

```
static Future<CharacterDetail> fetchCharacterById(int
id) async {
    final url = Uri.parse("$_baseUrl/characters/$id");
    final response = await http.get(url);

    if (response.statusCode == 200) {
        final data = json.decode(response.body);
        return CharacterDetail.fromJson(data);
    } else {
        throw Exception("Erro ao buscar personagem:
${response.statusCode}");
    }
}
```

## DRAGON BALL APP

# MÉTODO DO SERVICES

Após o `id` do personagem adicione o seguinte código para realizar a busca dos detalhes do personagem e exibir a descrição.



```
FutureBuilder(  
  future: ApiService.fetchCharacterById(character.id),  
  builder: (context, snapshot) {  
    if (snapshot.connectionState == ConnectionState.waiting) {  
      return const Center(child: CircularProgressIndicator());  
    }  
    if (snapshot.hasError) {  
      return Center(child: Text('Erro: ${snapshot.error}'));  
    }  
    final characterDetail = snapshot.data;  
    if (characterDetail == null) {  
      return const Center(  
        child: Text('Sem informações conhecidas'),  
      );  
    } else {  
      return Column(  
        children: [Text(characterDetail.description)],  
      );  
    }  
  },  
)
```

# FLUTTER

## EXERCÍCIO 1



## DRAGON BALL APP

# EXERCÍCIO 1

Implemente as transformações abaixo da descrição na tela de detalhe.



## DICA PARA RESOLVER

---

O **FutureBuilder** no Flutter só aceita um **Future** como parâmetro.

Se você precisa aguardar dois (ou mais) retornos assíncronos ao mesmo tempo, existem algumas estratégias comuns:

- Usar **Future.await**
- Criar um **Future** que combina os dois resultados
- Usar dois **FutureBuilders**

DRAGON BALL APP

## USANDO FUTURE.AWAIT

O jeito mais simples é juntar vários Futures em um só, com **Future.wait**.

Ele retorna uma **List** com os resultados na mesma ordem dos **Futures**.



```
FutureBuilder(  
  future: Future.wait([  
    fetchUserData(),  
    fetchSettings(),  
  ]),  
  builder: (context, snapshot) {  
    if (snapshot.connectionState == ConnectionState.waiting) {  
      return const Center(child: CircularProgressIndicator());  
    }  
    if (snapshot.hasError) {  
      return Center(child: Text('Erro: ${snapshot.error}'));  
    }  
    // snapshot.data é uma lista com os resultados  
    final userData = snapshot.data![0];  
    final settings = snapshot.data![1];  
    return Column(  
      children: [  
        Text("Usuário: $userData"),  
        Text("Configurações: $settings"),  
      ],  
    );  
  },  
);
```

## DRAGON BALL APP

# FUTURE COM COMBINAÇÃO DE RESULTADO

Você pode criar uma função que aguarda ambos e retorna um objeto com os dois juntos:



```
class UserAndSettings {  
    final String user;  
    final String settings;  
  
    UserAndSettings(this.user, this.settings);  
}  
  
Future<UserAndSettings> fetchUserAndSettings() async {  
    final user = await fetchUserData();  
    final settings = await fetchSettings();  
    return UserAndSettings(user, settings);  
}
```



```
FutureBuilder<UserAndSettings>(  
    future: fetchUserAndSettings(),  
    builder: (context, snapshot) {  
        if (snapshot.connectionState == ConnectionState.waiting) {  
            return const CircularProgressIndicator();  
        }  
        if (snapshot.hasError) {  
            return Text("Erro: ${snapshot.error}");  
        }  
        final data = snapshot.data!  
        return Column(  
            children: [  
                Text("Usuário: ${data.user}"),  
                Text("Configurações: ${data.settings}"),  
            ],  
        );  
    },  
);
```



DRAGON BALL APP

## USAR 2 FUTURES BUILDER

Não necessariamente precisa ser encadeado. Depende do cenário do app.



```
FutureBuilder(  
  future: fetchUserData(),  
  builder: (context, snapshotUser) {  
    if (!snapshotUser.hasData) {  
      return const CircularProgressIndicator();  
    }  
    return FutureBuilder(  
      future: fetchSettings(),  
      builder: (context, snapshotSettings) {  
        if (!snapshotSettings.hasData) {  
          return const CircularProgressIndicator();  
        }  
  
        return Column(  
          children: [  
            Text("Usuário: ${snapshotUser.data}"),  
            Text("Configurações: ${snapshotSettings.data}"),  
          ],  
        );  
      },  
    );  
  },  
);
```

# FLUTTER

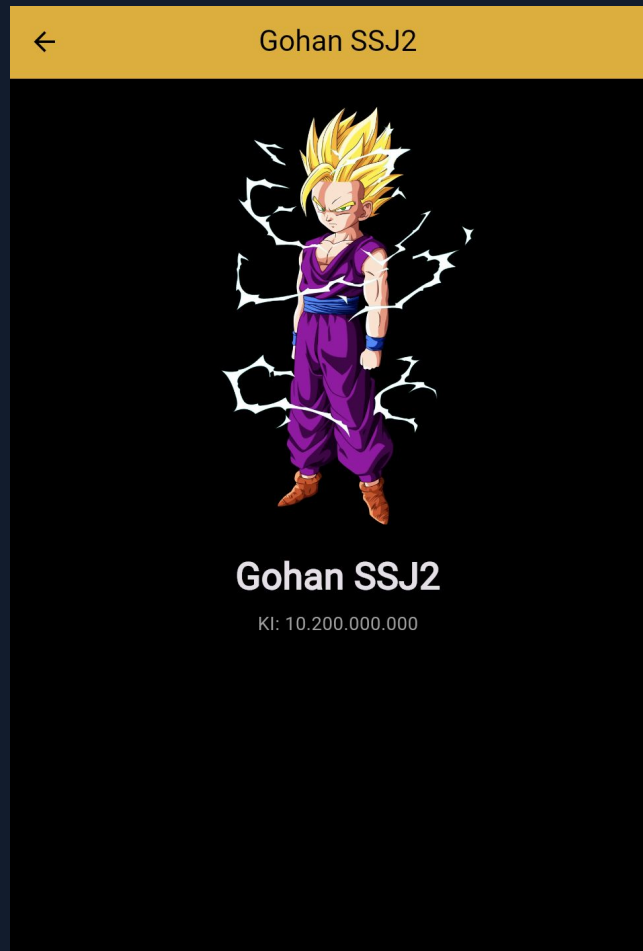
## EXERCÍCIO 2



## DRAGON BALL APP

### EXERCÍCIO 2

Ao clicar em uma das transformações abra uma nova tela que mostre os detalhes daquela transformação



# OBRIGADO



heider-lobes-a06b2869

Copyright © 2025 | Professor (a) Heider Lopes  
Todos os direitos reservados. Reprodução ou divulgação total ou parcial deste documento, é expressamente proibido sem consentimento formal, por escrito, do professor/autor.