11-2023

code
URJC

Desarrollo de Aplicaciones para Dispositivos Móviles

Tema 3: Desarrollo Híbrido

Tema 3.3: Programación asíncrona con Flutter

Universidad Rey Juan Carlos

Michel Maes
michel.maes@urjc.es

11-2023

code
URJC

Programación asíncrona

- La **programación asíncrona** permite continuar con la ejecución de nuestro código (hilo principal) mientras ejecuta una **tarea en segundo plano**
- Estas tareas asíncronas suelen ser
  - Obtención de datos a través de red (p.e. API)
  - Escribir en una base de datos
  - Leer datos de un archivo del sistema de ficheros

https://dart.dev/codelabs/async-await

Programación asíncrona

- Una función asíncrona devuelve
  - Un Future
  - Un Stream (si el resultado tiene varias partes)
- Para interactuar con estos resultados asíncronos, en Dart (de manera similar a JS) utilizaremos las palabras clave **async** y **await**

https://dart.dev/codelabs/async-await

Programación asíncrona

code URJC

- En el siguiente ejemplo veremos como **NO** utilizar un Future

```
String createOrderMessage() {
  var order = fetchUserOrder();
  return 'Your order is: $order';
}

Future<String> fetchUserOrder() ⇒
    Future.delayed(
      const Duration(seconds: 2),
      () ⇒ 'Large Latte',
    );

void main() {
  print(createOrderMessage());
}
```

```
Your order is: Instance of 'Future<String>'

Process finished with exit code 0
```

Programación asíncrona

code URJC

- En el siguiente ejemplo veremos como **NO** utilizar un Future

```
String createOrderMessage() {
  var order = fetchUserOrder();
  return 'Your order is: $order';
}

Future<String> fetchUserOrder() ⇒
    Future.delayed(
      const Duration(seconds: 2),
      () ⇒ 'Large Latte',
    );

void main() {
  print(createOrderMessage());
}
```

Como la tarea no se ha completado, devuelve un objeto **Future** incompleto

```
Your order is: Instance of 'Future<String>'

Process finished with exit code 0
```

Programación asíncrona

code URJC

Future

- Representa el resultado de una operación asíncrona
- Puede tener dos estados
  - **Completo** → Se ha finalizado la tarea (el futuro se completa **con un valor o con un error**)
    - Una instancia de **Future<T>** produce un valor de tipo T
    - Puede usarse **Future<void>** si no devuelve nada
  - **Incompleto** → El Future sigue esperando que se finalice la tarea

Programación asíncrona

- En el siguiente ejemplo, podemos ver como el **Future** se completa en segundo plano

```dart
Future<void> fetchUserOrder() {
  return Future.delayed(const Duration(seconds: 2), () => print('Large Latte'));
}

void main() {
  fetchUserOrder();
  print('Fetching user order...');
}
```

```
Fetching user order...
Large Latte

Process finished with exit code 0
```

Programación asíncrona

code URJC

- En este otro ejemplo vemos que la excepción salta en segundo plano (un Future con error)

```
Future<void> fetchUserOrder() {
  return Future.delayed(const Duration(seconds: 2),
          () => throw Exception('Logout failed: user ID is invalid'));
}

void main() {
  fetchUserOrder();
  print('Fetching user order...');
}
```

```
Fetching user order...
Unhandled exception:
Exception: Logout failed: user ID is invalid
...

Process finished with exit code 255
```

Programación asíncrona

code URJC

Async/Await

- Estas palabras reservadaspermiten definir funciones asíncronas y utilizar sus resultados
- Una función puede ser anotada con **async** antes del cuerpo de la misma
- La palabra clave await solo funcione en funciones anotadas con async

Programación asíncrona

code URJC

- Repasemos el ejemplo **síncrono**

```
String createOrderMessage() {
  var order = fetchUserOrder();
  return 'Your order is: $order';
}

Future<String> fetchUserOrder() ⇒
    // Imagine that this function is more complex and slow.
    Future.delayed(
      const Duration(seconds: 2),
      () ⇒ 'Large Latte',
    );

void main() {
  print(createOrderMessage());
}
```

```
Your order is: Instance of 'Future<String>'

Process finished with exit code 0
```

Programación asíncrona

code URJC

- Los transformamos en **asíncrono** con async/await

```dart
Future<String> createOrderMessage() async {
  var order = await fetchUserOrder();
  return 'Your order is: $order';
}

Future<String> fetchUserOrder() =>
  Future.delayed(
    const Duration(seconds: 2),
        () => 'Large Latte',
  );

Future<void> main() async {
  print('Fetching user order...');
  print(await createOrderMessage());
}
```

```
Fetching user order...
Your order is: Large Latte

Process finished with exit code 0
```

Programación asíncrona

code URJC

Ejemplo guiado

- A continuación llevaremos a cabo un ejemplo guiado paso a paso para aplicar los conocimientos aprendidos sobre programación asíncrona
- Veremos cómo trabajar con funciones asíncronas en Flutter
- Recuperaremos un lista de personajes de una serie a través de un servicio externo (API Pública)

Programación asíncrona

code URJC

Ejemplo guiado

- API pública de personajes de Rick y Morty
  - https://rickandmortyapi.com/
- Podemos hacer peticiones GET y recibir información sobre personajes o sobre un personaje concreto (JSON)

```
GET https://rickandmortyapi.com/api/character/1
```

```
{
    "id": 1,
    "name": "Rick Sanchez",
    ...
    "image": "https://rickandmortyapi.com/api/character/avatar/1.jpeg",
    …
}
```

Programación asíncrona

Ejemplo guiado

- Crearemos un nuevo proyecto Flutter vacío en Android Studio o VSCode
- Añadiremos la librería **http** al archivo **pubspec.yaml** para trabajar con peticiónes a servicios externos

```
dependencies:
  flutter:
    sdk: flutter
  http: ^1.1.0
```

- Tendremos que hacer un **pub get** para descargar la libreria

Programación asíncrona

code URJC

Ejemplo guiado

- Sustituimos el código de **lib/main.dart** por el siguiente

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

void main() async{
  final response = await http.get(Uri.parse('https://rickandmortyapi.com/api/character/'));
  if (response.statusCode == 200) {
    var json_response = json.decode(response.body);
    print(json_response);
  } else {
    throw Exception('Failed to load Character');
  }
}
```

- Devuelve un objeto JSON

```
{info: {count: 826, pages: 42, next: https://rickandmortyapi.com/api/character/?
page=2, prev: null}, results: [{ … }]}
```

Programación asíncrona

Ejemplo guiado

- Sustituimos el código de **lib/main.dart** por el siguiente

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

void main() async{
  final response = await http.get(Uri.parse('https://rickandmortyapi.com/api/character/'));
  if (response.statusCode == 200) {
    var json_response = json.decode(response.body);
    print(json_response);
  } else {
    throw Exception('Failed to load Character');
  }
}
```

Realizamos la petición (asíncrona) y esperamos por la respuesta con **await**

- Devuelve un objeto JSON

```
{info: {count: 826, pages: 42, next: https://rickandmortyapi.com/api/character/?
page=2, prev: null}, results: [{ … }]}
```

Programación asíncrona

Ejemplo guiado

- Sustituimos el código de **lib/main.dart** por el siguiente

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

void main() async{
  final response = await http.get(Uri.parse('https://rickandmortyapi.com/api/character/'));
  if (response.statusCode == 200) {
    var json_response = json.decode(response.body);
    print(json_response);
  } else {
    throw Exception('Failed to load Character');
  }
}
```

Si obetenemos un código de estado 200 OK, convertimos el cuerpo de la respuesta en un JSON manejable

- Devuelve un objeto JSON

```
{info: {count: 826, pages: 42, next: https://rickandmortyapi.com/api/character/?
page=2, prev: null}, results: [{ … }]}
```

Programación asíncrona

Ejemplo guiado

- Sustituimos el código de **lib/main.dart** por el siguiente

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

void main() async{
  final response = await http.get(Uri.parse('https://rickandmortyapi.com/api/character/'));
  if (response.statusCode == 200) {
    Iterable l = json.decode(response.body)['results'];
    print(l);
  } else {
    throw Exception('Failed to load Character');
  }
}
```

Accedemos al campo “results” que contiene la lista de personajes y obtenemos un iterable

- Devuelve una lista JSON

```
[{id: 1, name: Rick Sanchez, … }, {id: 2, name: Morty Smith, …} ...]
```

Programación asíncrona

code URJC

Ejemplo guiado

- El siguiente paso será crear una clase para almacenar la información de los personajes (**lib/Character.dart**)

```dart
class Character {
  final int id;
  final String name;
  final String image;

  const Character({
    required this.id,
    required this.name,
    required this.image,
  });

  factory Character.fromJson(Map<String, dynamic> json) {
    return Character(
      id: json['id'] as int,
      name: json['name'] as String,
      image: json['image'] as String,
    );
  }
}
```

Programación asíncrona

Ejemplo guiado

- El siguiente paso será crear una clase para almacenar la información de los personajes (**lib/Character.dart**)

```dart
class Character {
  final int id;
  final String name;
  final String image;

  const Character({
    required this.id,
    required this.name,
    required this.image,
  });

  factory Character.fromJson(Map<String, dynamic> json) {
    return Character(
      id: json['id'] as int,
      name: json['name'] as String,
      image: json['image'] as String,
    );
  }
}
```

Obtendremos solos algunos campos relevantes (pero podríamos añadir más)

Programación asíncrona

Ejemplo guiado

- El siguiente paso será crear una clase para almacenar la información de los personajes (**lib/Character.dart**)

```dart
class Character {
  final int id;
  final String name;
  final String image;

  const Character({
    required this.id,
    required this.name,
    required this.image,
  });

  factory Character.fromJson(Map<String, dynamic> json) {
    return Character(
      id: json['id'] as int,
      name: json['name'] as String,
      image: json['image'] as String,
    );
  }
}
```

Creamos un método que pueda convertir un JSON (que extiende Map) a un Character

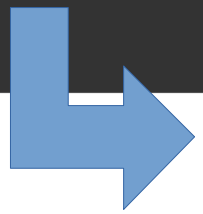
Programación asíncrona

Ejemplo guiado

- Utilizaremos la nueva clase y su método para convertir la lista de personajes obtenida y mostrar por pantalla los nombres

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

void main() async{
  final response = await http.get(Uri.parse('https://rickandmortyapi.com/api/character/'));
  if (response.statusCode == 200) {
    Iterable l = json.decode(response.body)['results'];
    var characters = List<Character>.from(l.map((item) => Character.fromJson(item)));
    characters.forEach((character) => print(character.name) );
  } else {
    throw Exception('Failed to load Character');
  }
}
```



```
Rick Sanchez
Morty Smith
Summer Smith
Beth Smith
Jerry Smith
```

Checkpoint: https://github.com/Maes95/DADM/commit/0e6f194f5e22da6dc766b48ff1b4ae72dc4287f3

Programación asíncrona

code URJC

Ejemplo guiado

- Ya tenemos los datos, pero aún nos queda mostrarlos en la interfaz
- Crearemos una función fetchCharacters() que contenga la funcionalidad creada hasta ahora, la cual devolverá un Future
- Crearemos un StatefulWidget llamado MyApp que contendrá nuestra aplicación
- Crearemos el estado _MyAppState, que será el estado de MyApp

Programación asíncrona

code URJC

Ejemplo guiado

- Creamos una función **fetchCharacters()** que contenga la funcionalidad creada hasta ahora, la cual devolverá un Future

```
Future<List<Character>> fetchCharacters() async {
  final response = await http.get(Uri.parse('https://rickandmortyapi.com/api/character/'));
  if (response.statusCode == 200) {
    Iterable l = json.decode(response.body)['results'];
    return List<Character>.from(l.map((item) => Character.fromJson(item)));
  } else {
    throw Exception('Failed to load Character');
  }
}
```

Programación asíncrona

code URJC

Ejemplo guiado

- Crearemos un StatefulWidget llamado **MyApp** que contendrá nuestra aplicacióny su estado, **_MyAppState**.

```
void main() ⇒ runApp(const MyApp());

class MyApp extends StatefulWidget {
  const MyApp({super.key});

  @override
  State<MyApp> createState() ⇒ _MyAppState();
}

class _MyAppState extends State<MyApp> {

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Rick and Morty Characters App',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
      ),
      home: const Text('Rick and Morty Characters')
    );
  }
}
```

Programación asíncrona

Ejemplo guiado

- En **_MyAppState** inicializaremos el estado con una llamada al método que obtene los Characters de la API

```
class _MyAppState extends State<MyApp> {

  late Future<List<Character>> charactersFuture;

  @override
  void initState() {
    super.initState();
    charactersFuture = fetchCharacters();
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Rick and Morty Characters App',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
      ),
      home: const Text('Rick and Morty Characters')
    );
  }
}
```

Programación asíncrona

Ejemplo guiado

- En **_MyAppState** inicializaremos el estado con una llamada al método que obtene los Characters de la API

```
class _MyAppState extends State<MyApp> {

  late Future<List<Character>> charactersFuture;

  @override
  void initState() {
    super.initState();
    charactersFuture = fetchCharacters();
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Rick and Morty Characters App',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
      ),
      home: const Text('Rick and Morty Characters')
    );
  }
}
```

Con la keyword **late** declaramos que la variable tomará valor más adelante

Programación asíncrona

Ejemplo guiado

- En **_MyAppState** inicializaremos el estado con una llamada al método que obtene los Characters de la API

```
class _MyAppState extends State<MyApp> {

  late Future<List<Character>> charactersFuture;

  @override
  void initState() {
    super.initState();
    charactersFuture = fetchCharacters();
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Rick and Morty Characters App',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
      ),
      home: const Text('Rick and Morty Characters')
    );
  }
}
```

En la función initState, llamamos la función fetchCharacters(). En este punto **no esperamos** por el resultado con await

Programación asíncrona

code URJC

```
@override
Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Rick and Morty Characters App',
    theme: ThemeData(
      colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
    ),
    home: FutureBuilder<List<Character>>(
      future: charactersFuture,
      builder: (context, snapshot) {
        if (snapshot.hasData) {
          return ListView.builder(
            itemCount: snapshot.data!.length,
            itemBuilder: (context, index) {
              Character character = snapshot.data![index];
              return Text(character.name);
            },
          );
        } else if (snapshot.hasError) {
          return Text('${snapshot.error}');
        }
        return const CircularProgressIndicator();
      },
    ),
  );
}
```

Programación asíncrona

code URJC

```dart
@override
Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Rick and Morty Characters App',
    theme: ThemeData(
      colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
    ),
    home: FutureBuilder<List<Character>>(
      future: charactersFuture,
      builder: (context, snapshot) {
        if (snapshot.hasData) {
          return ListView.builder(
            itemCount: snapshot.data!.length,
            itemBuilder: (context, index) {
              Character character = snapshot.data![index];
              return Text(character.name);
            },
          );
        } else if (snapshot.hasError) {
          return Text('${snapshot.error}');
        }
        return const CircularProgressIndicator();
      },
    ),
  );
}
```

No podemos utilizar directamente la variable **charactersFuture**

Utilizaremos un Widget especial llamado **FutureBuilder<T>**

Programación asíncrona

code URJC

```dart
@override
Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Rick and Morty Characters App',
    theme: ThemeData(
      colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
    ),
    home: FutureBuilder<List<Character>>(
      future: charactersFuture,
      builder: (context, snapshot) {
        if (snapshot.hasData) {
          return ListView.builder(
            itemCount: snapshot.data!.length,
            itemBuilder: (context, index) {
              Character character = snapshot.data![index];
              return Text(character.name);
            },
          );
        } else if (snapshot.hasError) {
          return Text('${snapshot.error}');
        }
        return const CircularProgressIndicator();
      },
    ),
  );
}
```

La construcción del Widget queda postergada a la resolución del **Future**

Programación asíncrona

code URJC

```dart
@override
Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Rick and Morty Characters App',
    theme: ThemeData(
      colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
    ),
    home: FutureBuilder<List<Character>>(
      future: charactersFuture,
      builder: (context, snapshot) {
        if (snapshot.hasData) {
          return ListView.builder(
            itemCount: snapshot.data!.length,
            itemBuilder: (context, index) {
              Character character = snapshot.data![index];
              return Text(character.name);
            },
          );
        } else if (snapshot.hasError) {
          return Text('${snapshot.error}');
        }
        return const CircularProgressIndicator();
      },
    ),
  );
}
```

Dentro del builder obtendremos la información del Future como parámetro

Programación asíncrona

code URJC

```dart
@override
Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Rick and Morty Characters App',
    theme: ThemeData(
      colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
    ),
    home: FutureBuilder<List<Character>>(
      future: charactersFuture,
      builder: (context, snapshot) {
        if (snapshot.hasData) {
          return ListView.builder(
            itemCount: snapshot.data!.length,
            itemBuilder: (context, index) {
              Character character = snapshot.data![index];
              return Text(character.name);
            },
          );
        } else if (snapshot.hasError) {
          return Text('${snapshot.error}');
        }
        return const CircularProgressIndicator();
      },
    ),
  );
}
```

Comprobaremos si se ha resuelto correctamente o no

Programación asíncrona

code URJC

```dart
@override
Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Rick and Morty Characters App',
    theme: ThemeData(
      colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
    ),
    home: FutureBuilder<List<Character>>(
      future: charactersFuture,
      builder: (context, snapshot) {
        if (snapshot.hasData) {
          return ListView.builder(
            itemCount: snapshot.data!.length,
            itemBuilder: (context, index) {
              Character character = snapshot.data![index];
              return Text(character.name);
            },
          );
        } else if (snapshot.hasError) {
          return Text('${snapshot.error}');
        }
        return const CircularProgressIndicator();
      },
    ),
  );
}
```

Si se ha resuelto correctamente, podremos acceder a los datos (**snapshot.data**)

Programación asíncrona

code URJC

```dart
@override
Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Rick and Morty Characters App',
    theme: ThemeData(
      colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
    ),
    home: FutureBuilder<List<Character>>(
      future: charactersFuture,
      builder: (context, snapshot) {
        if (snapshot.hasData) {
          return ListView.builder(
            itemCount: snapshot.data!.length,
            itemBuilder: (context, index) {
              Character character = snapshot.data![index];
              return Text(character.name);
            },
          );
        } else if (snapshot.hasError) {
          return Text('${snapshot.error}');
        }
        return const CircularProgressIndicator();
      },
    ),
  );
}
```

Utilizaremos una **ListView** para iterar y mostrar el nombre de los personajes

Programación asíncrona

code URJC

```dart
@override
Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Rick and Morty Characters App',
    theme: ThemeData(
      colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
    ),
    home: FutureBuilder<List<Character>>(
      future: charactersFuture,
      builder: (context, snapshot) {
        if (snapshot.hasData) {
          return ListView.builder(
            itemCount: snapshot.data!.length,
            itemBuilder: (context, index) {
              Character character = snapshot.data![index];
              return Text(character.name);
            },
          );
        } else if (snapshot.hasError) {
          return Text('${snapshot.error}');
        }
        return const CircularProgressIndicator();
      },
    ),
  );
}
```

Si hubo algún error, mostraremos el mensaje en la vista

Programación asíncrona

code URJC

```
@override
Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Rick and Morty Characters App',
    theme: ThemeData(
      colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
    ),
    home: FutureBuilder<List<Character>>(
      future: charactersFuture,
      builder: (context, snapshot) {
        if (snapshot.hasData) {
          return ListView.builder(
            itemCount: snapshot.data!.
            itemBuilder: (context, ind
              Character character = sn
              return Text(character.na
            },
          );
        } else if (snapshot.hasError) {
          return Text('${snapshot.error}');
        }
        return const CircularProgressIndicator();
      },
```

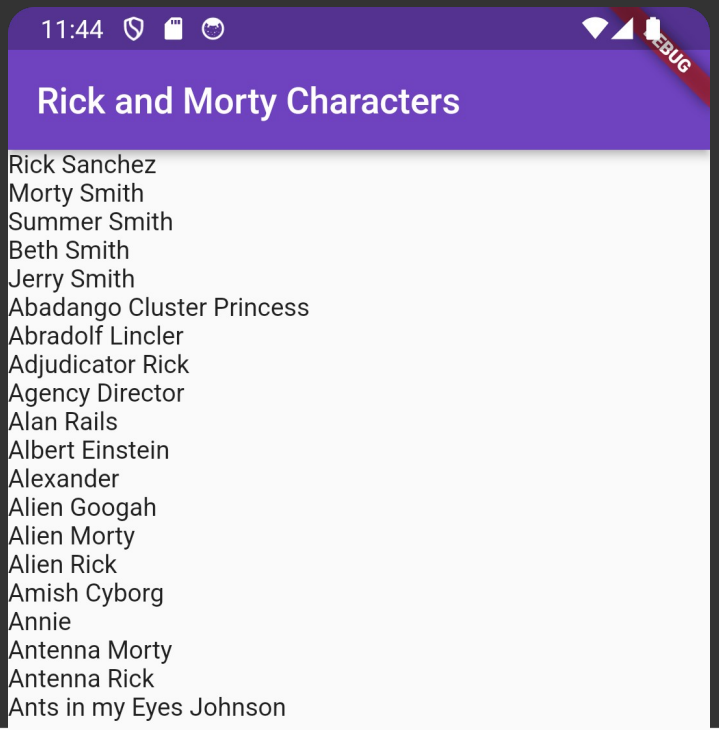
Si aún no se ha resuelto el resultado de Future, devolveremos un Widget por defecto (un indicador visual como **CircularProgressIndicator**)

Podemos añadir `await Future.delayed(const Duration(seconds: 1));` en fetchCharacters para comprobarlo

Programación asíncrona

Refactorizamos nuestra aplicación para que use **Scaffold** y utilice una estructura definida

```dart
@override
Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Rick and Morty Characters App',
    theme: ThemeData(
      colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
    ),
    home: Scaffold(
      appBar: AppBar(
        title: const Text('Rick and Morty Characters'),
      ),
      body: FutureBuilder<List<Character>>(
        future: charactersFuture,
        builder: (context, snapshot) {
          if (snapshot.hasData) {
            return ListView.builder(
              itemCount: snapshot.data!.length,
              itemBuilder: (context, index) {
                Character character = snapshot.data![index];
                return Text(character.name);
              },
            );
          } else if (snapshot.hasError) {
            return Text('${snapshot.error}');
          }
          return const CircularProgressIndicator();
        },
      ),
    ),
  );
}
```



Checkpoint:https://github.com/Maes95/DADM/commit/2f4eea2ac7bbd6750072dfd6c10931c1dc8b5e56

Programación asíncrona

Ejemplo guiado

- Hasta ahora solo estamos mostrando los nombres, pero hay mucha más información que podemos mostrar
- Dentro de Character habríamos recogido en una propiedad la URL de la imagen del personaje
- Podemos mostrar imágenes desde una URL con el siguiente Widget

```
Image.network(<URL>)
```

- Refactorizaremos de nuevo la app para que utilice un Widget llamado **CharacterWidget**

Programación asíncrona

code URJC

```
@override
Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Fetch Data Example',
    theme: ThemeData(
      colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
    ),
    home: Scaffold(
      appBar: AppBar(
        title: const Text('Rick and Morty Characters'),
      ),
      body: Center(
        child: FutureBuilder<List<Character>>(
          future: charactersFuture,
          builder: (context, snapshot) {
            if (snapshot.hasData) {
              return ListView.builder(
                itemCount: snapshot.data!.length,
                itemBuilder: (context, index) {
                  Character character = snapshot.data![index];
                  return CharacterWidget(character: character);
                },
              );
            } else if (snapshot.hasError) {
              return Text('${snapshot.error}');
            }
            return const CircularProgressIndicator();
          },
        ),
      ),
    ),
  );
}
```

Programación asíncrona

code URJC

Ejercicio 1

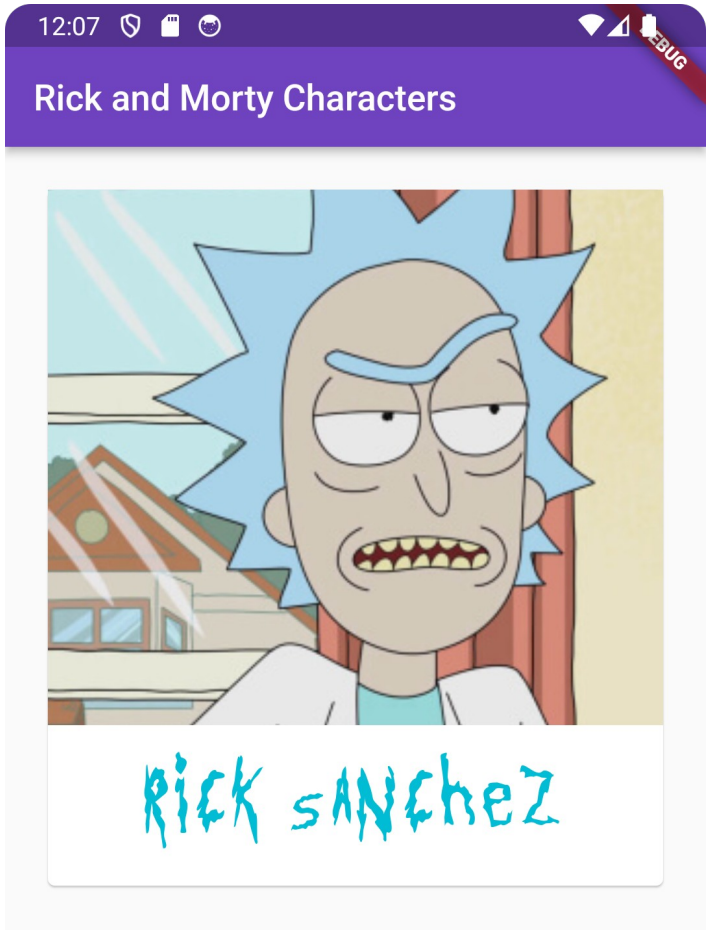
- A partir del código anterior, crea **CharacterWidget** para que reciba un objeto de la clase Character y muestre por pantalla tanto el nombre como su imagen
- Prueba diferentes componentes para mejorar la vista y hacerla más atractiva
  - Card: https://api.flutter.dev/flutter/material/Card-class.html
  - Ink: https://api.flutter.dev/flutter/material/Ink-class.html
  - SizedBox: https://api.flutter.dev/flutter/widgets/SizedBox-class.html
- Prueba a utilizar una fuente distinta para los nombres con GoogleFonts (añade la libreria **google_fonts: ^6.1.0** al pubspec.yaml)

Programación asíncrona

Ejercicio 1 - Solución

- La solución propuesta utiliza una fuente descargada en **fonts/**
- Hay que añadirla al **pubspec.yaml**

```
flutter:
  fonts:
    - family: RickAndMorty
      fonts:
        - asset: fonts/get_schwifty.ttf
          style: normal
  uses-material-design: true
```



Solución: https://github.com/Maes95/DADM/commit/789e1cee986d3b87a177bf72d988bc5594d26e40

Programación asíncrona

code URJC

Ejemplo guiado (continuación)

- Los resultados de la consulta muestran solamente la primera página
- La URL permite paginar las respuestas de la API

```
GET https://rickandmortyapi.com/api/character/?page=2
```

- Añadiremos a la app un botón que permita consultar las siguientes páginas de la respuesta

Programación asíncrona

code URJC

Ejemplo guiado

- Modificamos la función **fetchCharacters()** para que admita un parámetro que indique la página a consultar

```
Future<List<Character>> fetchCharacters(int page) async {
  final response =
  await http.get(Uri.parse('https://rickandmortyapi.com/api/character/?page=$page'));
  if (response.statusCode == 200) {
    Iterable l = json.decode(response.body)['results'];
    return List<Character>.from(l.map((item) => Character.fromJson(item)));
  } else {
    throw Exception('Failed to load Character');
  }
}
```

Programación asíncrona

code URJC

Ejemplo guiado

- Modificamos la clase **_MyAppState** para que mantenga la página actual en una variable y al lanzarse carge la primera página

```
class _MyAppState extends State<MyApp> {

  late Future<List<Character>> charactersFuture;
  int pageCount = 1;

  @override
  void initState() {
    super.initState();
    charactersFuture = fetchCharacters(pageCount);
  }
```

Programación asíncrona

code URJC

Ejemplo guiado

- Añadimos el botón a la vista

```
home: Scaffold(
  appBar: AppBar(
    title: const Text('Rick and Morty Characters'),
  ),
  body: Center(
    child: Column(
      children: [
        Padding(
          padding: const EdgeInsets.all(20.0),
          child: SizedBox(
            width: 200,
            height: 50,
            child: ElevatedButton(onPressed: ()⇒{
              setState(() {
                pageCount+=1;
                charactersFuture = fetchCharacters(pageCount);
              })
            }, child: const Text("Next page", style: TextStyle(fontSize: 30))),
          ),
        ),
        Expanded(
          child: FutureBuilder<List<Character>>(
```

Programación asíncrona

Ejemplo guiado

- Añadimos el botón a la vista

```
home: Scaffold(
  appBar: AppBar(
    title: const Text('Rick and Morty Characters'),
  ),
  body: Center(
    child: Column(
      children: [
        Padding(
          padding: const EdgeInsets.all(20.0),
          child: SizedBox(
            width: 200,
            height: 50,
            child: ElevatedButton(onPressed: ()⇒{
              setState(() {
                pageCount+=1;
                charactersFuture = fetchCharacters(pageCount);
              })
            }, child: const Text("Next page", style: TextStyle(fontSize: 30))),
          ),
        ),
        Expanded(
          child: FutureBuilder<List<Character>>(
```

Creamos un **Column** que contenga el botón y un componente **Expanded** (que contiene a su vez FutureBuilder). Si no introducimos FutureBuilder en este componente obtendremos un error.

Programación asíncrona

Ejemplo guiado

- Añadimos el botón a la vista

```
home: Scaffold(
  appBar: AppBar(
    title: const Text('Rick and Morty Characters'),
  ),
  body: Center(
    child: Column(
      children: [
        Padding(
          padding: const EdgeInsets.all(20.0),
          child: SizedBox(
            width: 200,
            height: 50,
            child: ElevatedButton(onPressed: ()⇒{
              setState(() {
                pageCount+=1;
                charactersFuture = fetchCharacters(pageCount);
              })
            }, child: const Text("Next page", style: TextStyle(fontSize: 30))),
          ),
        ),
        Expanded(
          child: FutureBuilder<List<Character>>(
```

Añadimos un **ElevatedButton**. Utilizamos **Padding** y **SizedBox** para añadirle estilo

Programación asíncrona

Ejemplo guiado

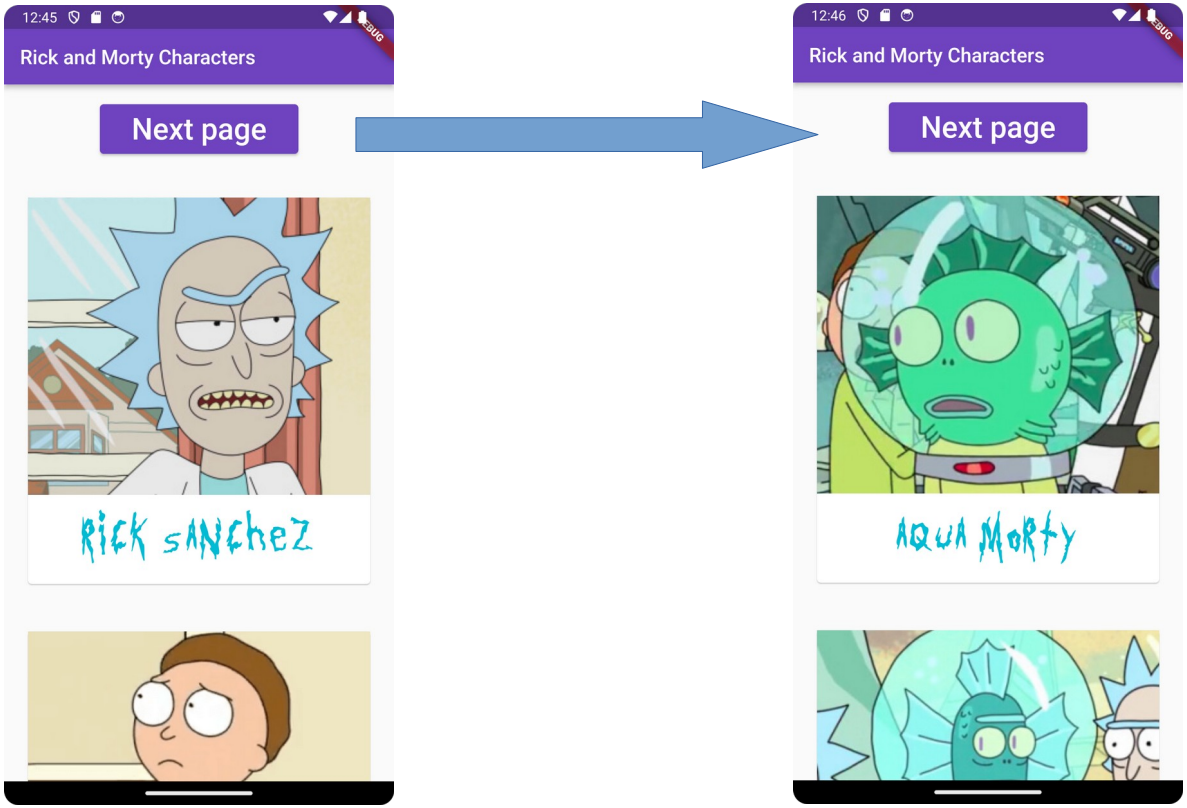
- Añadimos el botón a la vista

```
home: Scaffold(
  appBar: AppBar(
    title: const Text('Rick and Morty Characters'),
  ),
  body: Center(
    child: Column(
      children: [
        Padding(
          padding: const EdgeInsets.all(20.0),
          child: SizedBox(
            width: 200,
            height: 50,
            child: ElevatedButton(onPressed: ()⇒{
              setState(() {
                pageCount+=1;
                charactersFuture = fetchCharacters(pageCount);
              })
            }, child: const Text("Next page", style: TextStyle(fontSize: 30))),
          ),
        ),
        Expanded(
          child: FutureBuilder<List<Character>>(
```

Modificamos el estado del componente realizando otra llamada a la API para obtener la siguiente página

Programación asíncrona

Ejemplo guiado



Solución: https://github.com/Maes95/DADM/commit/3855ce7d49f852f2937f22c0a18f75df69a31bdd

Programación asíncrona

Ejercicio 2

- Selecciona una API públicaque no requiera credenciales (Auth=No)
  - https://github.com/public-apis/public-apis
- Comprueba lo que devuelve haciendo una consulta desde el navegador (cómo es el JSON resultante)
- Modifica el ejemplo anterior:
  - Cambia el método **fetchCharacters** para que haga una consulta a la nueva API
  - Crea una clase (como **Character**) para recoger y mapear la información que llegue desde la nueva API
  - Crea un Widget nuevo para sustituir a **CharacterWidget** que muestre la información de los elementos de la nueva API