Lección: Uso del Widget Image en Aplicaciones con Flutter

Objetivos de la Lección

- Comprender el propósito y uso del widget **Image** en Flutter para mostrar imágenes en la interfaz de usuario.
- Identificar las diferentes formas de cargar imágenes utilizando el widget **Image**.
- Implementar ejemplos básicos de visualización de imágenes en una aplicación Flutter.
- Personalizar y ajustar imágenes con propiedades del widget **Image**.

Introducción

En Flutter, el widget **Image** es uno de los widgets más utilizados para mostrar imágenes en una aplicación. El widget **Image** admite varias formas de cargar y visualizar imágenes, tanto locales como remotas (desde Internet), y proporciona una variedad de propiedades para personalizar su comportamiento, tamaño, y apariencia en la interfaz de usuario.

Flutter facilita la integración de imágenes dentro de la aplicación utilizando rutas de archivo, recursos en línea y activos declarados en el proyecto.

Formas de Cargar Imágenes en Flutter

El widget **Image** en Flutter permite cargar imágenes de varias fuentes. Las más comunes son:

1. **Image.asset**: Para mostrar imágenes locales desde los activos del proyecto.
2. **Image.network**: Para cargar y mostrar imágenes desde una URL o recurso en línea.

3. **Image.file**: Para cargar imágenes desde el sistema de archivos del dispositivo (útil para aplicaciones que permiten cargar imágenes desde la galería).
4. **Image.memory**: Para cargar imágenes a partir de datos binarios en memoria.

En esta lección, nos centraremos en los dos métodos más utilizados: **Image.asset** y **Image.network**.

Ejemplo Básico de Image.asset (Imágenes Locales)

Para utilizar **Image.asset**, primero se debe añadir la imagen al proyecto en la carpeta `assets` y declarar su ruta en el archivo `pubspec.yaml`.

Paso 1: Configurar el archivo `pubspec.yaml`

Agrega la imagen en la carpeta `assets` (por ejemplo, `assets/images/imagen.png`) y asegúrate de declararla en el archivo `pubspec.yaml`:

```
flutter:
  assets:
    - assets/images/imagen.png
```

Paso 2: Código para mostrar la imagen local con Image.asset

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

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
```

```
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Imagen Local en Flutter'),
        ),
        body: Center(
          child: Image.asset('assets/images/imagen.png'),
        ),
      ),
    );
  }
}
```

Explicación del Código:

1. **Image.asset**: Se utiliza para mostrar una imagen desde los activos locales de la aplicación. En este caso, la imagen `imagen.png` está almacenada en la carpeta `assets/images`.
2. **AppBar**: Muestra una barra de aplicación con el título "Imagen Local en Flutter".
3. **Center**: El widget `Image.asset` está centrado en el cuerpo de la aplicación.

Captura de la pantalla del ejemplo presentado con el Widget Image.
Creado por Javier A. Dastas (2024)

Ejemplo Básico de Image.network (Imágenes Remotas)

El widget **Image.network** es ideal para cargar imágenes desde una URL o recursos en línea. Esto es útil cuando las imágenes se encuentran alojadas en servidores o se obtienen dinámicamente.

Código para cargar una imagen desde una URL con Image.network

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


void main() {
```

```
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Imagen desde Internet'),
        ),
        body: Center(
          child: Image.network(
            'https://flutter.dev/images/flutter-logo.png',
          ),
        ),
      ),
    );
  }
}
```

Explicación del Código:

1. **Image.network**: Este widget se utiliza para cargar y mostrar una imagen desde una URL. En este caso, se carga el logotipo de Flutter desde una URL pública.
    a. Utiliza la siguiente dirección: https://static-00.iconduck.com/assets.00/flutter-icon-2048x2048-ufx4idi8.png

2. **Center**: La imagen se centra en el cuerpo de la aplicación.

Captura de la pantalla del ejemplo presentado con el Widget Image utilizando imagen leída a través de Internet. Creado por Javier A. Dastas (2024)

Propiedades Clave del Widget Image

El widget **Image** ofrece varias propiedades que permiten personalizar cómo se visualizan las imágenes. Algunas de las propiedades más útiles son:

1. fit: Ajustar el Comportamiento de la Imagen

La propiedad `fit` controla cómo se ajusta la imagen al contenedor. Las opciones más comunes incluyen:

- **BoxFit.cover**: Escala la imagen para cubrir todo el contenedor.

- **BoxFit.contain**: Escala la imagen para que quepa dentro del contenedor sin perder su proporción.
- **BoxFit.fill**: Escala la imagen para llenar el contenedor, ignorando su proporción.

Ejemplo con la propiedad fit:

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

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Ajuste de Imagen con BoxFit'),
        ),
        body: Center(
          child: Image.network(
            'https://flutter.dev/images/flutter-logo.png',
            fit: BoxFit.cover, // Cambia el ajuste de la imagen
            width: 200,
            height: 200,
          ),
        ),
```

```
        ),
      );
    }
}
```

Explicación del Código:

- **fit: BoxFit.cover**: Escala la imagen para cubrir el tamaño del contenedor de `200x200` píxeles, ajustándose para llenar todo el espacio.
- **width y height**: Se especifica el ancho y alto del contenedor de la imagen.

Captura de la pantalla del ejemplo presentado con el Widget Image utilizando la propiedad “fit” en una imagen leída a través de Internet.
Creado por Javier A. Dastas (2024)

2. width y height: Cambiar el Tamaño de la Imagen

La propiedad `width` y `height` permite ajustar el tamaño de la imagen, sin importar cuál sea su tamaño original.

Ejemplo con width y height:

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

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Tamaño Personalizado de Imagen'),
        ),
        body: Center(
          child: Image.network(
            'https://flutter.dev/images/flutter-logo.png',
            width: 150,   // Ancho personalizado
            height: 150,  // Alto personalizado
          ),
        ),
      ),
```

```
    );
  }
}
```

Explicación del Código:

- **width y height**: La imagen se ajusta a un tamaño de `150x150` píxeles, independientemente de su tamaño original.

3. color y colorBlendMode: Aplicar Efectos de Color a la Imagen

Las propiedades `color` y `colorBlendMode` permiten aplicar un filtro de color sobre la imagen para crear efectos visuales.

Ejemplo con color y colorBlendMode:

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

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Efecto de Color en Imagen'),
        ),
        body: Center(
```

```
          child: Image.network(
            'https://flutter.dev/images/flutter-logo.png',
            color: Colors.blue.withOpacity(0.5),  // Filtro azul
con opacidad
            colorBlendMode: BlendMode.colorBurn,  // Modo de
mezcla de color
          ),
        ),
      ),
    );
  }
}
```

Explicación del Código:

- **color**: Aplica un filtro de color azul sobre la imagen.
- **colorBlendMode**: Utiliza `BlendMode.colorBurn` para mezclar el color con la imagen, creando un efecto visual personalizado.

Captura de la pantalla del ejemplo presentado con el Widget Image utilizando la color y colorBlendMode en una imagen leída a través de Internet.
Creado por Javier A. Dastas (2024)

Manejo de Errores al Cargar Imágenes

Es posible que algunas imágenes remotas no se carguen correctamente (por ejemplo, si la URL es incorrecta o la imagen no está disponible). Para manejar estos errores, se puede usar la propiedad `errorBuilder` para mostrar un widget alternativo (como un ícono o texto).

Ejemplo con errorBuilder:

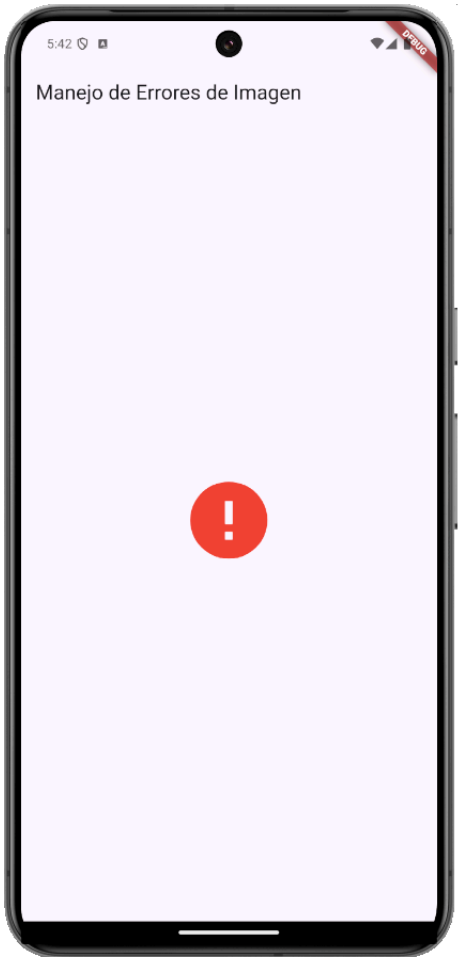
```
import 'package:flutter/material.dart';
```

```
void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Manejo de Errores de Imagen'),
        ),
        body: Center(
          child: Image.network(
            'https://flutter.dev/images/imagen-inexistente.png',
// URL incorrecta
            errorBuilder: (context, error, stackTrace) {
              return Icon(Icons.error, size: 100, color:
Colors.red);
            },
          ),
        ),
      ),
    );
  }
}
```

Explicación del Código:

- **errorBuilder**: Si la imagen no se carga correctamente, muestra un ícono de error rojo en lugar de la imagen.



Captura de la pantalla del ejemplo presentado con el Widget Image con el icono de error rojo al no encontrar la imagen indicada en el código.
Creado por Javier A. Dastas (2024)

Resumen de la Lección

En esta lección, hemos aprendido a utilizar el widget **Image** en Flutter para mostrar imágenes en la interfaz de usuario. Exploramos dos métodos principales para cargar imágenes: **Image.asset** (para imágenes locales) y **Image.network** (para imágenes remotas). También vimos cómo personalizar imágenes utilizando propiedades como

`fit`, `width`, `height`, y `colorBlendMode`, así como cómo manejar errores al cargar imágenes desde una URL.

El widget **Image** es esencial en Flutter para crear aplicaciones visualmente atractivas que integren contenido gráfico, y su flexibilidad lo hace adecuado para una amplia variedad de aplicaciones.

Actividad de la Lección

Estas actividades te permitirán aplicar los conceptos aprendidos y practicar la manipulación de imágenes en Flutter, creando interfaces más ricas y dinámicas.

Instrucciones:

1. Crea una aplicación en Flutter que muestre dos imágenes: una desde los activos locales (usando **Image.asset**) y otra desde una URL (usando **Image.network**). Ajusta el tamaño de las imágenes para que se vean uniformes en la pantalla.
2. Personaliza una imagen añadiendo un filtro de color con las propiedades `color` y `colorBlendMode`, y cambia su ajuste utilizando `BoxFit.contain`.
3. Entrega un documento en formato PDF con copia de tu código y copia de imágenes o capturas de pantalla demostrando que tu código funciona.