Módulo 4: Elementos Visuales y Flujo de Interacción entre Pantallas

Lección 1: Uso de Widgets de Botones y Navegación entre Pantallas

Objetivos de la Lección

- Comprender el concepto de widgets de botones en Flutter.
- Aprender a implementar diferentes tipos de botones en una aplicación Flutter.
- Conocer las propiedades y funcionalidades de los widgets de botones.
- Crear una actividad práctica para implementar botones y manejar eventos en Flutter.

Introducción de la Lección

Los widgets de botones son elementos interactivos fundamentales en Flutter que permiten a los usuarios realizar acciones en la aplicación. Estos widgets se utilizan para desencadenar eventos, como enviar un formulario o navegar a otra pantalla. Flutter proporciona diferentes tipos de botones que se adaptan a distintas situaciones, como ElevatedButton, TextButton, OutlinedButton, e IconButton.

En esta lección, explicaremos los diferentes tipos de widgets de botones disponibles en Flutter, sus características y cómo implementarlos de manera efectiva en una aplicación móvil.

Tipos de Widgets de Botones en Flutter

Flutter proporciona varios widgets de botones, cada uno con características específicas. A continuación, se describen algunos de los más comunes:

1. **ElevatedButton**: Es un botón que tiene un fondo elevado, lo que le da un efecto tridimensional. Se utiliza para acciones primarias que necesitan resaltar en la interfaz.
2. **TextButton**: Es un botón plano sin elevación. Es ideal para acciones secundarias o enlaces.
3. **OutlinedButton**: Es similar al TextButton, pero con un borde alrededor. Se utiliza para destacar opciones que no son las principales.
4. **IconButton**: Este tipo de botón muestra un ícono en lugar de texto, y es útil para acciones donde la representación visual es suficiente. Es común verlo en barras de aplicaciones o como parte de interacciones rápidas.

Propiedades Comunes de los Widgets de Botones

Los widgets de botones en Flutter comparten algunas propiedades comunes que permiten personalizarlos:

- **child**: Define el contenido del botón, que puede ser un texto o un ícono.
- **style**: Permite personalizar la apariencia del botón, como el color de fondo, la forma, el tamaño del texto, etc.
- **onPressed**: Es la función que se ejecuta cuando se presiona el botón. Si se deja como null, el botón aparecerá deshabilitado.

Ejemplos y Explicaciones Detalladas

A continuación, presentamos un ejemplo simple donde se implementan los cuatro tipos de botones en una aplicación Flutter. Cada botón realiza una acción específica cuando es presionado:

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: ButtonScreen(),
    );
  }
}

class ButtonScreen extends StatefulWidget {
  @override
  _ButtonScreenState createState() => _ButtonScreenState();
}

class _ButtonScreenState extends State<ButtonScreen> {
  Color backgroundColor = Colors.white;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Botones en Flutter'),
      ),
      body: Center(
```

```
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            ElevatedButton(
              onPressed: () {
                print("ElevatedButton presionado");
              },
              child: Text('ElevatedButton'),
            ),
            SizedBox(height: 10),
            TextButton(
              onPressed: () {
                setState(() {
                  backgroundColor = Colors.blueAccent;
                });
                print("TextButton presionado - Color de fondo
cambiado");
              },
              child: Text('TextButton'),
            ),
            SizedBox(height: 10),
            OutlinedButton(
              onPressed: () {
                Navigator.push(
                  context,
                  MaterialPageRoute(builder: (context) =>
NewScreen()),
                );
                print("OutlinedButton presionado - Navegación a
nueva pantalla");
```

```
              },
              child: Text('OutlinedButton'),
            ),
            SizedBox(height: 10),
            IconButton(
              icon: Icon(Icons.thumb_up),
              onPressed: () {
                print("IconButton presionado - Acción rápida
realizada");
              },
            ),
          ],
        ),
      ),
      backgroundColor: backgroundColor,
    );
  }
}

class NewScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Nueva Pantalla'),
      ),
      body: Center(
        child: Text('Has llegado a una nueva pantalla'),
      ),
    );
```

```
    }
}
```

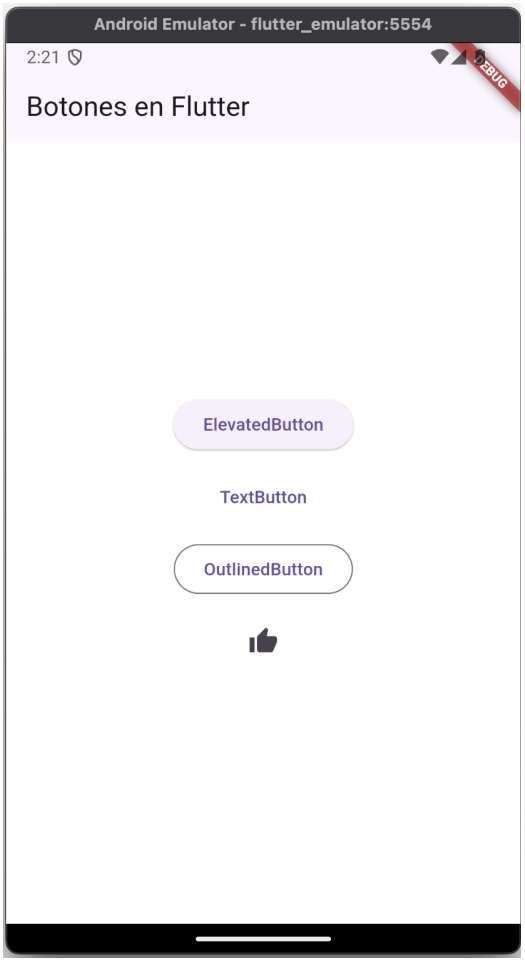

Imagen de la Captura de la pantalla del código de ejemplo.
Creado por Javier A. Dastas (2024)

Comprendiendo el uso de la instrucción Navigator.push

En Flutter, la navegación entre pantallas es una parte esencial del desarrollo de aplicaciones móviles. Una aplicación puede tener varias pantallas, y los usuarios deben poder moverse de una a otra de manera eficiente. Flutter proporciona una clase llamada Navigator, que administra un stack (pila) de rutas o pantallas. La función Navigator.push se utiliza para agregar una nueva pantalla a este stack, lo que permite la transición hacia una nueva página o vista.

En la línea de código:

```
Navigator.push(
  context,
  MaterialPageRoute(builder: (context) => NewScreen()),
);
```

Cada componente tiene un papel específico en el proceso de navegación. A continuación, desglosamos esta línea para entender cómo funciona dentro de Flutter:

- **Navigator.push:** El método push de la clase Navigator agrega una nueva ruta (o pantalla) en la parte superior del stack de navegación. Esto significa que la nueva pantalla será visible, mientras que la pantalla anterior quedará debajo en la pila y estará oculta. Si el usuario desea volver a la pantalla anterior, puede usar el botón de retroceso o invocar Navigator.pop para eliminar la pantalla actual del stack y mostrar la anterior.
- **context:** El context es un objeto de tipo BuildContext, que se utiliza para acceder a la jerarquía de widgets en Flutter. Cuando pasamos el context al Navigator, le estamos indicando en qué parte del árbol de widgets se encuentra la pantalla actual, lo que permite al Navigator saber desde dónde se inicia la nueva ruta.
- **MaterialPageRoute:** MaterialPageRoute es una implementación de Route en Flutter que gestiona la transición entre pantallas con un estilo material design. Al usar MaterialPageRoute, podemos crear una nueva ruta y especificar qué widget (pantalla) se debe mostrar cuando se navega hacia esa ruta.
- **`builder: (context) => NewScreen()`** es un callback que devuelve el widget de la nueva pantalla que se va a mostrar. En este caso, estamos creando una nueva instancia de NewScreen, que es una pantalla simple en este ejemplo.

Flujo de la Navegación

Cuando el usuario presiona el botón asociado con este código, Flutter ejecuta Navigator.push, lo que añade la ruta NewScreen al stack de navegación. Esto provoca que la interfaz cambie de la pantalla actual a la nueva pantalla (NewScreen). Si el usuario luego desea regresar a la pantalla anterior, puede presionar el botón de retroceso, que eliminará NewScreen del stack (a través de Navigator.pop automáticamente), mostrando la pantalla original.

Ejemplo Visual del Stack de Navegación:

```
Pantalla 1 (actual) -> Navigator.push() -> Pantalla 2 (NewScreen)
```

Cuando Navigator.push es llamado, NewScreen se coloca en la cima de la pila.

La pantalla actual permanece en la pila, pero queda inactiva hasta que se llame a Navigator.pop, que quitaría NewScreen de la pila y volvería a mostrar la pantalla original.

Importancia de Navigator.push en el desarrollo de Aplicaciones

El método Navigator.push es fundamental para crear aplicaciones móviles con múltiples pantallas, permitiendo una experiencia de usuario fluida. Sin él, las aplicaciones estarían limitadas a una sola pantalla, lo que no es práctico en el desarrollo moderno. Al comprender cómo gestionar rutas y navegación, los desarrolladores pueden construir aplicaciones móviles complejas y bien estructuradas.

Relación con Otros Conceptos o Lecciones

Los widgets de botones están estrechamente relacionados con otros conceptos en Flutter, como la **navegación entre pantallas** y el **manejo de estado**. Los botones se utilizan comúnmente para navegar a otras pantallas o para cambiar el estado de la

aplicación, como al enviar datos o desencadenar acciones en respuesta a eventos del usuario.

Resumen de la Lección

En esta lección, hemos explorado los diferentes tipos de widgets de botones en Flutter, sus propiedades y cómo implementarlos en una aplicación. Aprendimos sobre **ElevatedButton**, **TextButton**, **OutlinedButton** e **IconButton**, y cómo personalizar sus estilos y funcionalidades para adaptarse a las necesidades de la aplicación.

Además, en esta lección se introdujo el método Navigator.push que permite a los usuarios navegar entre pantallas en una aplicación Flutter a través de las gestión de rutas dentro de una pila (stack). La clase MaterialPageRoute facilita la creación de rutas que siguen las guías de diseño material design, mientras que el uso de context le da a Flutter la información necesaria sobre el lugar en el árbol de widgets desde donde se está realizando la navegación. Con esta funcionalidad, puedes crear aplicaciones móviles dinámicas y fáciles de usar.

Actividad de la Lección

En esta actividad, el estudiante logrará aplicar los conocimientos adquiridos sobre los diferentes tipos de botones en Flutter. Al completar la actividad, podrán implementar botones interactivos que desencadenan acciones variadas, demostrando así su comprensión y habilidad para utilizar estos widgets de manera efectiva.

Instrucciones

1. **Implementa una pequeña aplicación en Flutter** siguiendo los siguientes pasos:
    - Crea un nuevo proyecto en Flutter.
    - Copia el código que aparece en esta lección.
    - Modifica el código para que cada Botón en la pantalla tenga asignada una función que llame una pantalla con un color de fondo distinto.

      Lista de colores recomendados para un Diseño Moderno:

      1. **Blanco Suave** - `Color(0xFFF5F5F5)`
      2. **Azul Pizarra Claro** - `Color(0xFFE0F7FA)`
      3. **Verde Menta** - `Color(0xFFD1F2EB)`
      4. **Amarillo Pastel** - `Color(0xFFFFF9C4)`
      5. **Lavanda** - `Color(0xFFE1BEE7)`
      6. **Aqua Suave** - `Color(0xFFB2EBF2)`
      7. **Coral Ligero** - `Color(0xFFFF8A65)`

    - Prueba la aplicación en un simulador o dispositivo físico y verifica que cada botón realice la acción correspondiente.
2. **Documenta tus resultados** en un archivo PDF, donde describas las acciones realizadas por cada botón y cualquier desafío que hayas enfrentado durante el desarrollo. Incluye copia de tu código e imágenes de todas las pantallas de la aplicación.