Módulo 4: Interacción de Pantallas y Elementos para Captura de Datos

Lección 3: Captura de Gestos o Eventos en Flutter

Objetivos de la Lección

- Comprender cómo funciona el sistema de gestos en Flutter.
- Aprender a utilizar el widget `GestureDetector` para capturar diferentes tipos de gestos.
- Implementar gestos comunes como toques, deslizamientos y mantenimientos prolongados en una aplicación Flutter.
- Saber cuándo y cómo utilizar la captura de gestos para mejorar la interacción del usuario.

Introducción de la Lección

En Flutter, la captura de gestos es esencial para crear aplicaciones interactivas y responsivas. Los gestos permiten a los usuarios interactuar con la interfaz a través de toques, deslizamientos, mantenimientos prolongados y otros movimientos comunes en pantallas táctiles. Flutter proporciona el widget `GestureDetector`, que nos permite capturar y reaccionar a estos gestos.

En esta lección, aprenderás cómo utilizar el `GestureDetector` para capturar gestos y cómo implementar interacciones que respondan a las acciones del usuario, mejorando la experiencia en tus aplicaciones.

El Widget `GestureDetector`

El `GestureDetector` es un widget que puede envolver cualquier otro widget para captar los gestos del usuario. Puedes usarlo para detectar una variedad de gestos, como:

- **onTap**: Captura un toque simple.
- **onDoubleTap**: Captura un doble toque.
- **onLongPress**: Detecta una pulsación prolongada.
- **onPanUpdate**: Captura el movimiento continuo de un dedo (deslizamiento).

Uso Básico de `GestureDetector`:

El siguiente código muestra cómo usar `GestureDetector` para capturar un toque simple.

```
GestureDetector(
  onTap: () {
    print("Se detectó un toque");
  },
  child: Container(
    width: 100,
    height: 100,
    color: Colors.blue,
    child: Center(child: Text('Toque aquí')),
  ),
)
```

En este ejemplo, al tocar el área azul, el mensaje "Se detectó un toque" aparecerá en la consola.

Gestos Comunes:

1. **onTap**:
    - Utilizado para capturar un toque simple, ideal para botones u otros elementos que requieran interacción táctil básica.

```
GestureDetector(
  onTap: () {
    print("Se hizo un toque simple");
  },
  child: Container(
    color: Colors.green,
    width: 200,
    height: 200,
    child: Center(child: Text('Toque')),
  ),
)
```

2. **onDoubleTap**:
    - Este gesto se detecta cuando el usuario toca dos veces rápidamente. Es común en interacciones como hacer zoom o "me gusta" en una imagen.

```
GestureDetector(
  onDoubleTap: () {
    print("Doble toque detectado");
  },
  child: Container(
    color: Colors.yellow,
    width: 200,
    height: 200,
```

```
    child: Center(child: Text('Doble Toque')),
  ),
)
```

3. **onLongPress**:
    - Se detecta cuando el usuario mantiene presionada una parte de la pantalla por un tiempo prolongado. Suele utilizarse para activar menús contextuales o acciones adicionales.

```
GestureDetector(
  onLongPress: () {
    print("Pulsación prolongada detectada");
  },
  child: Container(
    color: Colors.red,
    width: 200,
    height: 200,
    child: Center(child: Text('Pulsación Prolongada')),
  ),
)
```

4. **onPanUpdate**:
    - Este gesto detecta el arrastre o deslizamiento del dedo a través de la pantalla. Es útil para mover objetos o realizar desplazamientos personalizados.

```
GestureDetector(
  onPanUpdate: (details) {
    print("Movimiento detectado: ${details.localPosition}");
  },
```

```
  child: Container(
    color: Colors.purple,
    width: 200,
    height: 200,
    child: Center(child: Text('Desliza aquí')),
  ),
)
```

Ejemplos y Explicaciones Detalladas

Capturando Múltiples Gestos en un Widget:

Un widget puede capturar varios gestos simultáneamente. Aquí tienes un ejemplo que detecta un toque simple, un doble toque y una pulsación prolongada en el mismo widget.

```
GestureDetector(
  onTap: () {
    print("Toque simple");
  },
  onDoubleTap: () {
    print("Doble toque");
  },
  onLongPress: () {
    print("Pulsación prolongada");
  },
  child: Container(
    color: Colors.orange,
    width: 200,
```

```
    height: 200,
    child: Center(child: Text('Diferentes Gestos')),
  ),
)
```

Deslizamientos con onPanUpdate:

El gesto onPanUpdate es particularmente útil cuando necesitas capturar movimientos continuos. El ejemplo siguiente actualiza la posición de un contenedor mientras el usuario desliza el dedo.

```
double xPosition = 0;
double yPosition = 0;

GestureDetector(
  onPanUpdate: (details) {
    xPosition += details.delta.dx;
    yPosition += details.delta.dy;
    // Llama a setState si estás en un StatefulWidget para
actualizar la interfaz
  },
  child: Transform.translate(
    offset: Offset(xPosition, yPosition),
    child: Container(
      width: 100,
      height: 100,
      color: Colors.blue,
    ),
  ),
```

)

Ejemplo y Explicación Detallada: Uso de onPanUpdate

El gesto `onPanUpdate` es muy útil cuando necesitamos capturar movimientos continuos del dedo en la pantalla, como al arrastrar objetos o realizar desplazamientos personalizados en una interfaz. Este gesto detecta el movimiento del dedo y proporciona información detallada de la posición en la que el dedo se encuentra, lo que permite actualizar la interfaz de manera dinámica según la interacción del usuario.

Ejemplo: Arrastrar un Contenedor con onPanUpdate

En el siguiente ejemplo, implementamos un contenedor que puede ser arrastrado por la pantalla al capturar el movimiento del dedo con el gesto `onPanUpdate`. La posición del contenedor se actualiza en tiempo real a medida que el usuario lo desplaza.

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

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

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

```
class PanUpdateExample extends StatefulWidget {
  @override
  _PanUpdateExampleState createState() =>
_PanUpdateExampleState();
}

class _PanUpdateExampleState extends State<PanUpdateExample> {
  double xPosition = 0;
  double yPosition = 0;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Ejemplo de onPanUpdate'),
      ),
      body: Stack(
        children: [
          Positioned(
            left: xPosition,
            top: yPosition,
            child: GestureDetector(
              onPanUpdate: (details) {
                setState(() {
                  xPosition += details.delta.dx;
                  yPosition += details.delta.dy;
                });
              },
              child: Container(
                width: 100,
```

```
                height: 100,
                color: Colors.blue,
                child: Center(
                  child: Text('Arrástrame'),
                ),
              ),
            ),
          ),
        ],
      ),
    );
  }
}
```

Explicación del Código:

1. **Inicialización de las coordenadas**: Utilizamos dos variables (`xPosition` y `yPosition`) para rastrear la posición actual del contenedor. Estas variables se inicializan en 0, lo que sitúa el contenedor en la parte superior izquierda de la pantalla al inicio.
2. **Uso de `GestureDetector`**: El contenedor azul está envuelto en un widget `GestureDetector`, que nos permite capturar el movimiento continuo del dedo sobre el widget mediante el gesto `onPanUpdate`.
3. **Actualizar la posición**: En el callback `onPanUpdate`, la posición del contenedor se actualiza con `details.delta.dx` (movimiento en el eje horizontal) y `details.delta.dy` (movimiento en el eje vertical). Cada vez que el usuario mueve el dedo, la interfaz se actualiza utilizando `setState`, lo que refleja el nuevo desplazamiento del contenedor.

4. **Posicionamiento dinámico**: El widget `Positioned` permite situar el contenedor en la posición específica indicada por `xPosition` y `yPosition`, facilitando el movimiento en la pantalla.

Este ejemplo muestra cómo implementar un arrastre básico en Flutter mediante el gesto `onPanUpdate`. A medida que el usuario arrastra el contenedor, este sigue su dedo, lo que genera una interacción visual inmediata y personalizada, mejorando la experiencia del usuario.

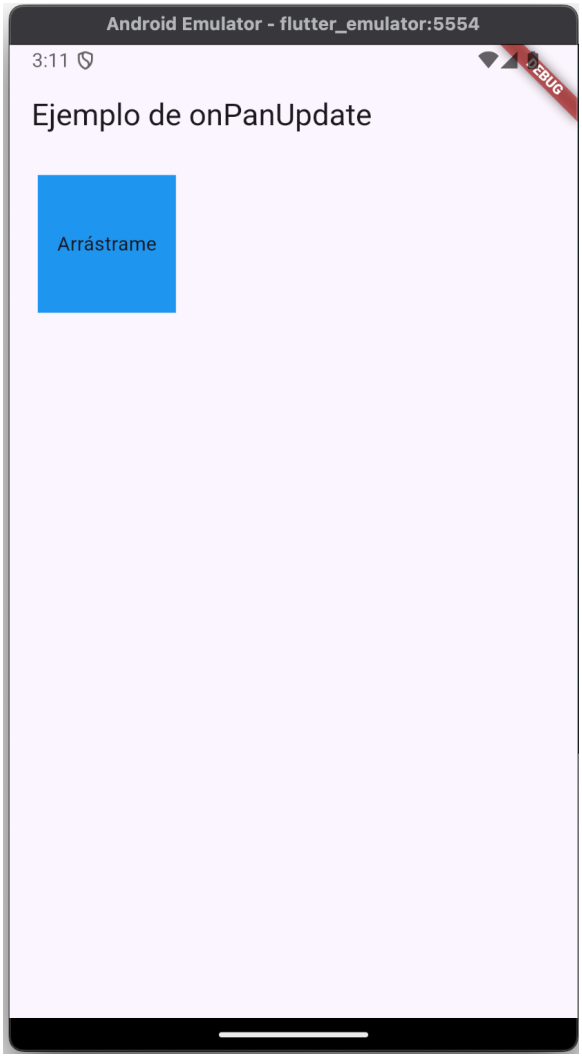

Imagen de la aplicación del código de ejemplo anterior que presenta el uso del onPanUpdate que permite arrastrar un elemento visual en pantalla.

Creado por Javier A. Dastas (2024)

Este tipo de gestos es muy útil en aplicaciones que requieren interfaces interactivas, como juegos, aplicaciones de diseño o mapas, donde los usuarios interactúan directamente con los elementos en pantalla.

Conceptos Relacionados con los Gestos de Arrastre (Pan)

Estos son los conceptos clave relacionados con los gestos de arrastre (*Pan*) en Flutter:

- **onPanStart**: Se activa cuando el dedo toca la pantalla y puede comenzar a moverse en cualquier dirección (horizontal o vertical). Si se utilizan callbacks como `onHorizontalDragStart` o `onVerticalDragStart`, este callback causará un conflicto.
- **onPanUpdate**: Detecta el movimiento continuo del dedo mientras está en contacto con la pantalla y se mueve en dirección horizontal o vertical. Si `onHorizontalDragUpdate` o `onVerticalDragUpdate` están definidos, este callback generará un error.
- **onPanEnd**: Se dispara cuando el dedo deja de tocar la pantalla, informando la velocidad del movimiento en el momento de levantar el dedo. Al igual que con los anteriores, si `onHorizontalDragEnd` o `onVerticalDragEnd`están en uso, causará un conflicto.

El **widget Stack** en el Diseño de Pantallas

El widget **`Stack`** en Flutter es una herramienta que permite apilar widgets uno encima del otro, creando una disposición en capas. A diferencia de otros widgets de disposición como `Column` o `Row`, que organizan elementos en una sola dimensión (vertical u horizontal), **`Stack`** permite posicionar widgets de forma libre en ambas dimensiones (horizontal y vertical).

Cada widget dentro de un **`Stack`** puede colocarse utilizando el widget **`Positioned`**, que permite especificar exactamente dónde debe estar cada elemento en relación con los bordes del contenedor. Esto es útil cuando necesitas superponer widgets, como

imágenes con texto encima, íconos en una esquina específica, o cuando creas interfaces complejas con múltiples capas.

Ejemplo básico de Stack:

```
Stack(
  children: <Widget>[
    Container(
      width: 200,
      height: 200,
      color: Colors.blue,
    ),
    Positioned(
      left: 50,
      top: 50,
      child: Icon(
        Icons.star,
        color: Colors.white,
        size: 50,
      ),
    ),
  ],
)
```

En este ejemplo, el contenedor azul actúa como la capa base, y el ícono de estrella blanca se superpone a una posición específica dentro de esa capa. Esto muestra cómo el **`Stack`** permite una disposición flexible y precisa de los elementos dentro de una interfaz.

Relación con Otros Conceptos o Lecciones

La captura de gestos está relacionada con la interacción humano-computadora (IHC) y la mejora de la experiencia de usuario (UX). Implementar gestos adecuados permite que los usuarios interactúen de manera intuitiva con la aplicación. Además, el uso de `GestureDetector` es un buen ejemplo de cómo manejar eventos y estados en Flutter, ya que muchos gestos requieren cambios inmediatos en la interfaz, lo que involucra la actualización del estado de los widgets.

Resumen de la Lección

En esta lección, hemos cubierto los aspectos fundamentales de la captura de gestos en Flutter utilizando el widget `GestureDetector`. Hemos explorado gestos comunes como toques simples, doble toque, pulsaciones prolongadas y deslizamientos, y cómo aplicarlos en una aplicación Flutter. La captura de gestos es una parte clave del desarrollo de aplicaciones táctiles modernas, y ahora deberías tener una comprensión más clara de cómo implementar interacciones táctiles en tus proyectos.

Actividad de la Lección

Esta actividad te permitirá aplicar lo aprendido sobre la captura de gestos en Flutter y demostrar que comprendes cómo implementar interacciones táctiles en una aplicación.

Instrucciones:

1. Implementa una aplicación Flutter con los siguientes requisitos:
    - Debe tener 2 contenedores (Container) de colores diferentes que incluyan un texto dentro que respondan a los siguientes gestos:
        i. El primer contenedor debe detectar un toque simple.
        ii. El segundo contenedor debe detectar un doble toque.
    - Debe tener una imagen que debe detectar o permitir que sea deslizada a través de la pantalla.
        i. Al soltar la imagen debe aparecer una Snackbar con un mensaje. Para esta acción necesitas buscar referencia o ayuda con relación al uso de “**onPanEnd”.**
2. Asegúrate de que cada contenedor imprima un mensaje con un Diálogo o Snackbard al capturar el gesto correspondiente.
3. Documenta tu código e imágenes de las pantallas en un documento en un formato de PDF.
4. Entrega el PDF en el enlace provisto por el profesor para esta actividad.