Módulo 2: Fundamentos para el Desarrollo de Aplicaciones Móviles

Lección 7

: Comprendiendo la Estructura Básica de una Aplicación con Flutter y Dart

Objetivos de la Lección

- Comprender la estructura básica de una aplicación móvil creada con **Flutter** y **Dart**.
- Identificar los componentes principales de una aplicación Flutter.
- Escribir y ejecutar una aplicación Flutter simple utilizando Dart.
- Reconocer el flujo básico de ejecución en una aplicación Flutter.

Introducción a Flutter y Dart

Flutter es un framework desarrollado por Google que permite crear aplicaciones móviles nativas para Android e iOS con un solo código base. Flutter se utiliza junto con el lenguaje de programación **Dart**. Es popular por su alto rendimiento y su capacidad para crear interfaces de usuario rápidas y atractivas.

Dart es un lenguaje de programación optimizado para aplicaciones que requieren interfaces de usuario rápidas en múltiples plataformas. En Flutter, Dart actúa como el lenguaje para escribir la lógica y la interfaz de la aplicación.

En esta lección, aprenderemos la estructura básica de una aplicación móvil Flutter, con el código más simple posible.

Estructura Básica de una Aplicación Flutter

Una aplicación Flutter sigue una estructura básica de archivos y carpetas. Al crear una nueva aplicación Flutter, el sistema genera varios archivos y carpetas, pero el archivo más importante es el archivo **main.dart**, que contiene el punto de entrada de la aplicación.

El código más básico de una aplicación Flutter consta de una función `main()`, que ejecuta el método `runApp()`. Este método arranca la aplicación Flutter y toma un widget como argumento. Un **widget** es un componente visual en Flutter.

Componentes principales:

1. **main()**: La función de entrada principal que ejecuta la aplicación.
2. **runApp()**: Inicia la aplicación y coloca el widget raíz en la pantalla.
3. **StatelessWidget**: Un widget que no tiene estado mutable (su contenido no cambia).
4. **MaterialApp**: Un widget que envuelve toda la aplicación, proporcionando un diseño básico.

Código más Básico en Flutter

Aquí tienes un ejemplo del código más simple que puedes usar para crear una aplicación en Flutter:

```
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('Mi Primera App en Flutter'),
        ),
        body: Center(
          child: Text('¡Hola, mundo!'),
        ),
      ),
    );
  }
}
```

Explicación del Código:

- **import 'package**
  /material.dart';: Importa el paquete de Flutter que contiene los widgets visuales como botones, texto, columnas, filas, etc.

- **void main() { runApp(MyApp()); }**: La función main() es el punto de entrada de la aplicación. El método runApp() toma un widget (en este caso, MyApp) y lo ejecuta en la pantalla.
- **class MyApp extends StatelessWidget**: Aquí se define la clase MyApp, que extiende de StatelessWidget. Un StatelessWidget es un widget que no cambia su estado mientras se está ejecutando.
- **MaterialApp**: Es el widget raíz que proporciona una estructura de diseño Material (diseño estándar de Google para aplicaciones Android).
- **Scaffold**: Es un widget que proporciona la estructura visual básica de la aplicación, como el app bar (barra superior) y el cuerpo de la aplicación.
- **AppBar**: El widget que crea una barra de título en la parte superior de la pantalla.
- **Center**: Alinea su hijo (en este caso, el texto "¡Hola, mundo!") en el centro de la pantalla.
- **Text('¡Hola, mundo!')**: Muestra un simple texto en la pantalla con el contenido "¡Hola, mundo!".

Flujo de la Aplicación

- La aplicación se inicia ejecutando la función `main()`.
- `runApp()` toma el widget `MyApp` y lo muestra en la pantalla.
- `MyApp` es un widget que construye un árbol de otros widgets.
- El widget `MaterialApp` crea el contexto de la aplicación y, dentro de él, `Scaffold` proporciona la estructura básica (app bar y cuerpo).
- El contenido de la aplicación (un texto en este caso) se centra en la pantalla con el widget `Center`.

Estructura de Archivos de una Aplicación Flutter

Cuando creas una aplicación Flutter, obtienes la siguiente estructura de archivos:

```
my_flutter_app/
  ├── android/
  ├── ios/
  ├── lib/
      └── main.dart
  ├── test/
  └── pubspec.yaml
```

- **lib/main.dart**: El archivo principal donde escribes el código Dart.
- **pubspec.yaml**: El archivo de configuración donde se definen dependencias y activos de la aplicación.
- **android/** y **ios/**: Carpetas que contienen el código específico para Android y iOS, pero se gestionan automáticamente al compilar.

Resumen de la Lección

En esta lección, hemos aprendido la estructura básica de una aplicación móvil utilizando **Flutter** y **Dart**. Vimos que la aplicación comienza con la función `main()`, que ejecuta el widget raíz de la aplicación a través de `runApp()`. La aplicación básica contiene un widget `MaterialApp` que proporciona una estructura de diseño estándar y usa widgets como `Scaffold`, `AppBar` y `Text` para mostrar contenido en la pantalla.

Esta estructura básica puede ser expandida para crear aplicaciones móviles más complejas con interfaces dinámicas e interactivas, todo utilizando el poder de Flutter y Dart.

Actividad Tipo Tutorial: Creando una Aplicación con Texto Vertical en Flutter

Instrucciones

En esta actividad, crearás una aplicación Flutter que mostrará tres textos centrados en la pantalla, dispuestos verticalmente utilizando un widget de **Columna** (`Column`). Los textos mostrarán:

- "Mi nombre es <Nombre>"
- "COMP2850 Computación Móvil"
- "Hola desde Flutter"

Cada texto tendrá un color diferente y un tamaño de fuente distinto. A lo largo del tutorial, te guiaré paso a paso y explicaré los widgets que necesitas para completar la actividad.

Paso 1: Crear el Proyecto Flutter

1. Desarrolla un nuevo proyecto de Flutter en tu editor preferido para desarrollo (Android Studio o Visual Studio Code) con la estructura básica de archivos de tu proyecto Flutter.
2. Navega a la carpeta `lib/` y abre el archivo `main.dart` en tu editor de código preferido.

Paso 2: Modificar el Archivo main.dart

Ahora vamos a modificar o reescribir el archivo `main.dart` para agregar tres textos en una columna centrada. Sigue los pasos a continuación.

1. Importar el Paquete de Flutter

Asegúrate de que el archivo `main.dart` tenga la línea de importación básica que carga los widgets de Flutter.

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

2. Escribir la Función Principal

La función `main()` es el punto de entrada de la aplicación. El método `runApp()` se encargará de iniciar tu aplicación.

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

3. Crear el Widget Principal MyApp

Aquí vamos a definir el widget principal de la aplicación, llamado `MyApp`. Este widget es de tipo `StatelessWidget` porque no tendrá estado mutable.

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

```
          child: MyColumn(),  // Aquí colocamos la columna
personalizada
        ),
      ),
    );
  }
}
```

Paso 3: Utilizar el Widget Column

Para organizar los tres textos de forma vertical, utilizaremos el widget **Column**. El widget `Column` permite disponer elementos (widgets) uno debajo del otro, de forma vertical.

1. Crear el Widget MyColumn

Vamos a crear un nuevo widget llamado `MyColumn`, que contendrá nuestros tres textos. Colocamos este widget dentro del cuerpo de la aplicación (`body`).

```
class MyColumn extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Column(
      mainAxisAlignment: MainAxisAlignment.center,  // Centra
los textos verticalmente
      crossAxisAlignment: CrossAxisAlignment.center, // Centra
los textos horizontalmente
      children: <Widget>[
        Text(
```

```
    'Mi nombre es <Nombre>',
    style: TextStyle(
      fontSize: 24.0,    // Tamaño del texto
      color: Colors.blue, // Color del texto
    ),
  ),
  SizedBox(height: 20),  // Espacio entre los textos
  Text(
    'COMP2850 Computación Móvil',
    style: TextStyle(
      fontSize: 20.0,
      color: Colors.green,
    ),
  ),
  SizedBox(height: 20),
  Text(
    'Hola desde Flutter',
    style: TextStyle(
      fontSize: 18.0,
      color: Colors.red,
    ),
  ),
],
```

```
    );
  }
}
```

Explicación de los Nuevos Widgets

1. Column:

El widget `Column` se utiliza para alinear widgets verticalmente. En este caso, estamos utilizando `Column` para organizar los tres textos uno debajo del otro. Tiene dos propiedades importantes:

- **mainAxisAlignment**: Alinea los widgets a lo largo del eje principal, que en una columna es el eje vertical. Usamos `MainAxisAlignment.center` para centrar los textos verticalmente en la pantalla.
- **crossAxisAlignment**: Alinea los widgets a lo largo del eje transversal (horizontal). Usamos `CrossAxisAlignment.center` para centrar los textos horizontalmente.

2. Text:

El widget `Text` se utiliza para mostrar texto en la pantalla. Usamos la propiedad `style` para personalizar el tamaño y color del texto con el widget `TextStyle`.

- **fontSize**: Define el tamaño del texto.
- **color**: Define el color del texto. Aquí estamos utilizando colores predeterminados como `Colors.blue`, `Colors.green` y `Colors.red`.

3. SizedBox:

El widget `SizedBox` se usa para agregar espacio entre los widgets. En este caso, lo utilizamos para agregar 20 píxeles de espacio entre cada texto. La propiedad `height` define la altura del espacio.

Paso 4: Ejecutar la Aplicación

1. Guarda el archivo `main.dart`.
2. En tu editor de preferencia, ejecuta la aplicación en un emulador o dispositivo conectado.

Deberías ver tres textos verticalmente alineados en la pantalla, cada uno con un color y tamaño diferentes.

Resumen de la Actividad

En esta actividad tutorial, hemos creado una aplicación básica con **Flutter** que muestra tres textos dispuestos verticalmente utilizando el widget **Column**. Hemos aprendido cómo centrar los elementos en la pantalla y cómo aplicar estilos personalizados a cada texto. Esta estructura básica puede ser utilizada para crear layouts más complejos y personalizables en aplicaciones móviles.

Al entender el uso de widgets como **Column**, **Text** y **SizedBox**, ahora puedes organizar contenido de manera eficiente en una interfaz de usuario.