Lección 1: Configuración del Entorno y Estructura del Proyecto con Flutter

Objetivos de la Lección

- **Comprender** los componentes esenciales para desarrollar aplicaciones móviles con Flutter.
- **Instalar y configurar** el entorno de desarrollo Flutter y Dart.
- **Crear** un nuevo proyecto en Flutter y estructurar su organización.
- **Explorar** los widgets básicos y la estructura de una aplicación Flutter.
- **Configurar** el proyecto para soportar modos de color (claro y oscuro).

Introducción a la Lección

En la era digital actual, el desarrollo de aplicaciones móviles es una habilidad altamente demandada. Flutter, un framework de código abierto desarrollado por Google, permite crear aplicaciones nativas para iOS y Android utilizando un solo código base. En esta lección, nos enfocaremos en configurar el entorno de desarrollo y en establecer la estructura base de nuestra aplicación móvil que simulará una red social básica utilizando los endpoints de **JSONPlaceholder**.

Esta configuración es esencial para sentar las bases de nuestro proyecto y asegurarnos de que estamos listos para implementar funcionalidades más avanzadas en lecciones posteriores.

Desarrollo del Proyecto

1. Creación de un Nuevo Proyecto Flutter

Paso 1: Creación del Proyecto

En la terminal, o través del opción en tu editor, navega al directorio donde deseas crear el proyecto y ejecuta:

```
flutter create red_social_basica
```

Esto creará una nueva aplicación Flutter con la estructura básica.

Paso 2: Exploración de la Estructura del Proyecto

El proyecto generado tendrá la siguiente estructura:

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

- **android/** y **ios/**: Archivos específicos de cada plataforma.
- **lib/**: Contiene el código fuente de la aplicación.
- **main.dart**: Punto de entrada de la aplicación.
- **pubspec.yaml**: Archivo de configuración del proyecto, donde se especifican dependencias y assets.

2. Ejecutar la Aplicación por Primera Vez

Paso 1: Iniciar un Emulador

- En Android Studio, inicia el emulador creado previamente.
- O conecta un dispositivo físico habilitando la depuración USB.

Paso 2: Ejecutar la Aplicación

En la opciones provistas por tu editor, dentro del directorio del proyecto, corre tu aplicación, o en la terminal ejecuta:

```
flutter run
```

Esto compilará y desplegará la aplicación en el dispositivo seleccionado.

Paso 3: Verificar la Aplicación

Deberías ver la aplicación Flutter por defecto, que muestra un contador incrementable al presionar un botón.

3. Configuración Inicial del Proyecto

Ahora, personalizamos el proyecto para adaptarlo a nuestras necesidades.

Paso 1: Modificar `pubspec.yaml`

Añadiremos dependencias que utilizaremos más adelante.

```
dependencies:
  flutter:
    sdk: flutter
  http: ^0.13.4
  shared_preferences: ^2.0.7
```

- **http**: Para realizar peticiones HTTP a los endpoints.
- **shared_preferences**: Para almacenar preferencias de usuario, como el modo de color seleccionado.

Guarda el archivo modificado, o en terminal, ejecuta:

```
flutter pub get
```

Para actualizar las dependencias.

Paso 2: Organizar el Código Fuente

Crearemos carpetas para organizar mejor nuestro código.

Dentro de **lib/**, crea las siguientes carpetas:

- **models/**: Para las clases modelo que representarán los datos.
- **screens/**: Para las pantallas de la aplicación.
- **widgets/**: Para componentes reutilizables de UI.
- **services/**: Para manejar las peticiones a las APIs.

4. Configuración del Soporte para Modo Claro y Oscuro

Configurar el tema de la aplicación para soportar modos de color.

Paso 1: Modificar `main.dart`

Abre **lib/main.dart** y ajusta el código para establecer los temas.

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

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

```
class MyApp extends StatefulWidget {
  // Este widget es la raíz de la aplicación.
  @override
  _MyAppState createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  ThemeMode _themeMode = ThemeMode.system;

    //  Aquí  agregaríamos  lógica  para  cargar  y  guardar  la
preferencia del usuario.

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Red Social Básica',
      theme: ThemeData(
        brightness: Brightness.light,
        primarySwatch: Colors.blue,
      ),
      darkTheme: ThemeData(
        brightness: Brightness.dark,
        primarySwatch: Colors.blue,
      ),
      themeMode: _themeMode, // Modo de tema seleccionado
      home: MyHomePage(
        onThemeChanged: _toggleThemeMode,
      ),
    );
  }
```

```
  void _toggleThemeMode(ThemeMode mode) {
    setState(() {
      _themeMode = mode;
    });
    // Aquí guardaríamos la preferencia del usuario
  }
}

class MyHomePage extends StatelessWidget {
  final Function(ThemeMode) onThemeChanged;

  MyHomePage({required this.onThemeChanged});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Red Social Básica'),
        actions: [
          PopupMenuButton<ThemeMode>(
            onSelected: onThemeChanged,
            itemBuilder: (context) => [
              PopupMenuItem(
                value: ThemeMode.system,
                child: Text('Predeterminado'),
              ),
              PopupMenuItem(
                value: ThemeMode.light,
                child: Text('Modo Claro'),
```

```
              ),
              PopupMenuItem(
                value: ThemeMode.dark,
                child: Text('Modo Oscuro'),
              ),
            ],
          ),
        ],
      ),
      body: Center(
        child: Text('Bienvenido a la Red Social Básica'),
      ),
    );
  }
}
```

Explicación:

- **StatefulWidget**: Convertimos `MyApp` en un `StatefulWidget` para manejar cambios en el estado del tema.
- **ThemeMode**: Usamos `_themeMode` para almacenar el modo de tema actual.
- **PopupMenuButton**: Añadimos un menú en la `AppBar` para cambiar entre modos.

Paso 2: Configuración de Preferencias

En lecciones posteriores, implementaremos la funcionalidad para guardar y cargar las preferencias del usuario respecto al modo de color utilizando `shared_preferences`.

5. Accesibilidad y Organización del Código

Para facilitar el entendimiento y mantenimiento del código:

- Añade comentarios claros y descriptivos.
- Sigue una convención de nombres consistente.
- Divide el código en archivos y clases pequeñas y manejables.

Ejemplos en Código

Ejemplo: Creación de un Modelo de Usuario

Crea un archivo **lib/models/user.dart** para definir la clase `User`.

```
class User {
  final int id;
  final String name;
  final String username;
  final String email;

      User({required   this.id,   required   this.name,   required
this.username, required this.email});

  factory User.fromJson(Map<String, dynamic> json) {
    return User(
      id: json['id'],
      name: json['name'],
      username: json['username'],
      email: json['email'],
    );
  }
}
```

Explicación:

- **Modelo User**: Representa la estructura de un usuario obtenida del endpoint `/users`.
- **fromJson**: Método de fábrica para crear una instancia de `User` a partir de un mapa JSON.

Relación con Otros Temas

Esta lección se relaciona con conceptos previos de programación orientada a objetos y estructuras de datos. La organización del proyecto y la creación de modelos refuerzan la importancia de un código bien estructurado y mantenible. Además, sienta las bases para consumir APIs REST, que se explorarán en la siguiente lección.

Resumen de la Lección

En esta lección, hemos configurado el entorno de desarrollo para Flutter y creado la estructura inicial de nuestro proyecto. Instalamos las herramientas necesarias, creamos un nuevo proyecto y organizamos el código fuente para facilitar el desarrollo futuro. También configuramos temas claros y oscuros para la aplicación, preparando el terreno para implementar la persistencia de preferencias del usuario.

Actividad de la Lección

Objetivo de la Actividad:

- Asegurar que has configurado correctamente el entorno de desarrollo.
- Demostrar que puedes crear y organizar un proyecto Flutter.
- Prepararte para las siguientes lecciones, donde implementaremos funcionalidades más avanzadas.

Tarea Práctica:

1. **Instala y configura** Flutter y las herramientas necesarias en tu sistema.
2. **Crea el proyecto** `red_social_basica` siguiendo los pasos indicados.
3. **Ejecuta la aplicación** en un emulador o dispositivo físico para verificar que funciona correctamente.
4. **Organiza el proyecto** creando las carpetas y archivos mencionados.
5. **Implementa el soporte** para temas claro y oscuro en la aplicación.
6. **Documenta tu proceso**: Crea un documento en PDF que incluya capturas de pantalla de cada paso y explica cualquier dificultad que hayas encontrado y cómo la solucionaste.
7. **Entrega**: Sube el PDF y el código fuente del proyecto en una carpeta comprimida.