

Desarrollo de una aplicación móvil con principios de diseño

Autores

Alexis Chimba, Javier Ramos, Ariel Reyes, Anthony Villarreal.

Ingeniera Doris Chicaiza

Universidad de las Fuerzas Armadas - ESPE

aschimba@espe.edu.ec, sjramos@espe.edu.ec, alreyes2@espe.edu.ec, anvillarreal@espe.edu.ec

Resumen

Este proyecto es una aplicación móvil desarrollada con Flutter cuyo objetivo es permitir a los usuarios verificar si un año dado es bisiesto o no. A través de una interfaz intuitiva y moderna, los usuarios pueden ingresar un año y obtener una respuesta inmediata sobre su condición bisiesta. La lógica del programa está organizada mediante un controlador central que gestiona la verificación de los años y conecta las diferentes vistas de la app. Además, se ha aplicado una estructura modular, lo que facilita su mantenimiento y escalabilidad. El diseño visual está personalizado mediante temas claros predefinidos, lo que mejora la experiencia del usuario. Esta aplicación es ideal como ejemplo educativo para quienes desean iniciarse en Flutter, ya que demuestra buenas prácticas de programación, separación de responsabilidades, y una navegación sencilla entre vistas.

Palabras clave - Flutter, Aplicación educativa, Verificación de años bisiestos, Diseño modular.

I. Introducción

En la actualidad, el desarrollo de aplicaciones móviles ha tomado gran relevancia, no solo en el ámbito comercial, sino también en el educativo. Este proyecto nace con el objetivo de crear una herramienta sencilla, funcional y visualmente amigable que permita a cualquier usuario verificar si un año es bisiesto. Aunque puede parecer una funcionalidad básica, su implementación sirve como una excelente base para comprender la estructura y el funcionamiento de una aplicación construida con Flutter.

La aplicación no solo se centra en la verificación, sino que también refleja buenas prácticas de programación, como la separación de lógica, el uso de controladores, y la navegación entre vistas. Además, se ha trabajado en un diseño limpio y un tema visual consistente que facilita la interacción del usuario. Esta herramienta representa una forma accesible de acercarse al mundo del desarrollo móvil, combinando funcionalidad y aprendizaje de forma práctica.

II. Trabajos Relacionados

Diversas aplicaciones educativas han sido desarrolladas con Flutter para enseñar conceptos básicos de programación y matemáticas. Proyectos similares, como calculadoras simples o convertidores de unidades, han servido como punto de partida para estudiantes que se inician en el desarrollo móvil. En este contexto, la verificación de años bisiestos se presenta como un ejercicio útil y concreto para aplicar lógica condicional y navegación entre vistas. A diferencia de otros enfoques más complejos, este proyecto prioriza la claridad del código y la experiencia del usuario, alineándose con buenas prácticas modernas.

III. Materiales y Métodos

- Android Studio
- Extensión Flutter
- Extensión Dart
- Sistema Operativo Windows 11

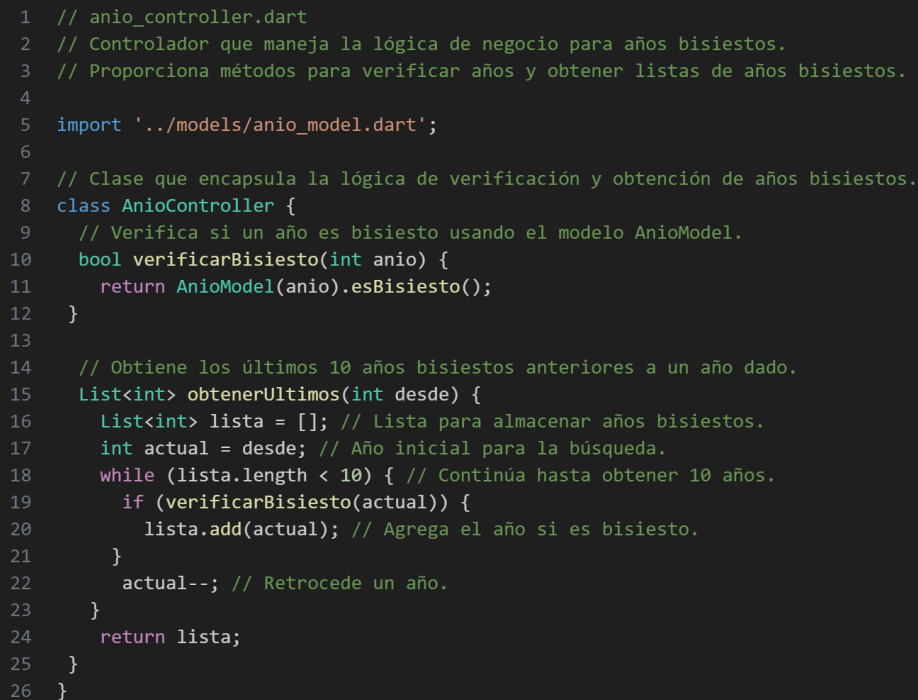
IV. Desarrollo

App móvil hecha en Flutter que te dice si un año es bisiesto, con diseño simple y buen orden en el código.

- **controllers:** Gestionan la lógica del programa, como la verificación de si un año es bisiesto, separando la vista del procesamiento.

Figura 1

Controlador de lógica para verificación de años bisiestos



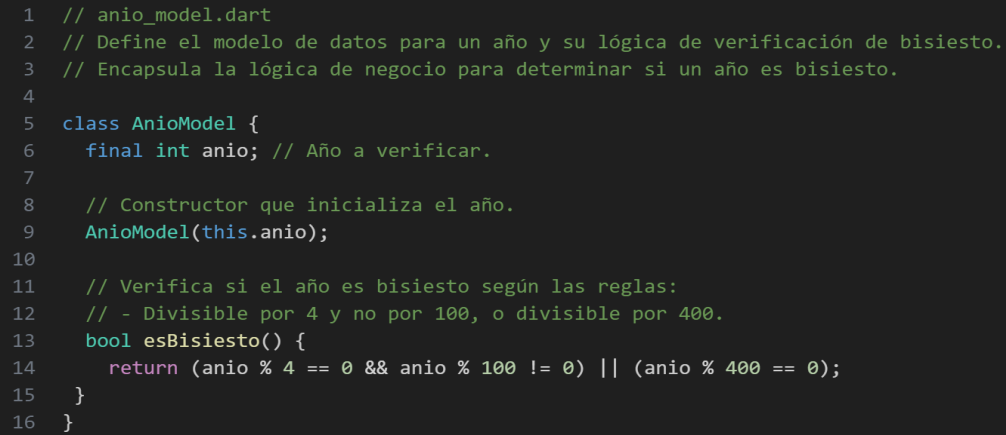
```
1 // anio_controller.dart
2 // Controlador que maneja la lógica de negocio para años bisiestos.
3 // Proporciona métodos para verificar años y obtener listas de años bisiestos.
4
5 import '../models/anio_model.dart';
6
7 // Clase que encapsula la lógica de verificación y obtención de años bisiestos.
8 class AnioController {
9   // Verifica si un año es bisiesto usando el modelo AnioModel.
10  bool verificarBisiesto(int anio) {
11    return AnioModel(anio).esBisiesto();
12  }
13
14  // Obtiene los últimos 10 años bisiestos anteriores a un año dado.
15  List<int> obtenerUltimos(int desde) {
16    List<int> lista = []; // Lista para almacenar años bisiestos.
17    int actual = desde; // Año inicial para la búsqueda.
18    while (lista.length < 10) { // Continúa hasta obtener 10 años.
19      if (verificarBisiesto(actual)) {
20        lista.add(actual); // Agrega el año si es bisiesto.
21      }
22      actual--; // Retrocede un año.
23    }
24    return lista;
25  }
26 }
```

Nota. Este controlador se encarga de la lógica para saber si un año es bisiesto. También puede mostrarte los últimos 10 años bisiestos antes de la fecha que elijas. Todo está organizado usando un modelo aparte, lo que ayuda a que el código sea más ordenado y fácil de entender.

- **models:** Representan estructuras de datos simples utilizadas para organizar la información necesaria en la aplicación.

Figura 2

Modelo de verificación de año bisiesto

A screenshot of a code editor window with a dark background and light-colored text. The code is written in Dart and defines a class named 'AnioModel'. It includes comments in Spanish explaining the purpose of the code and the logic for determining a leap year. The code is numbered from 1 to 16 on the left side of the editor.

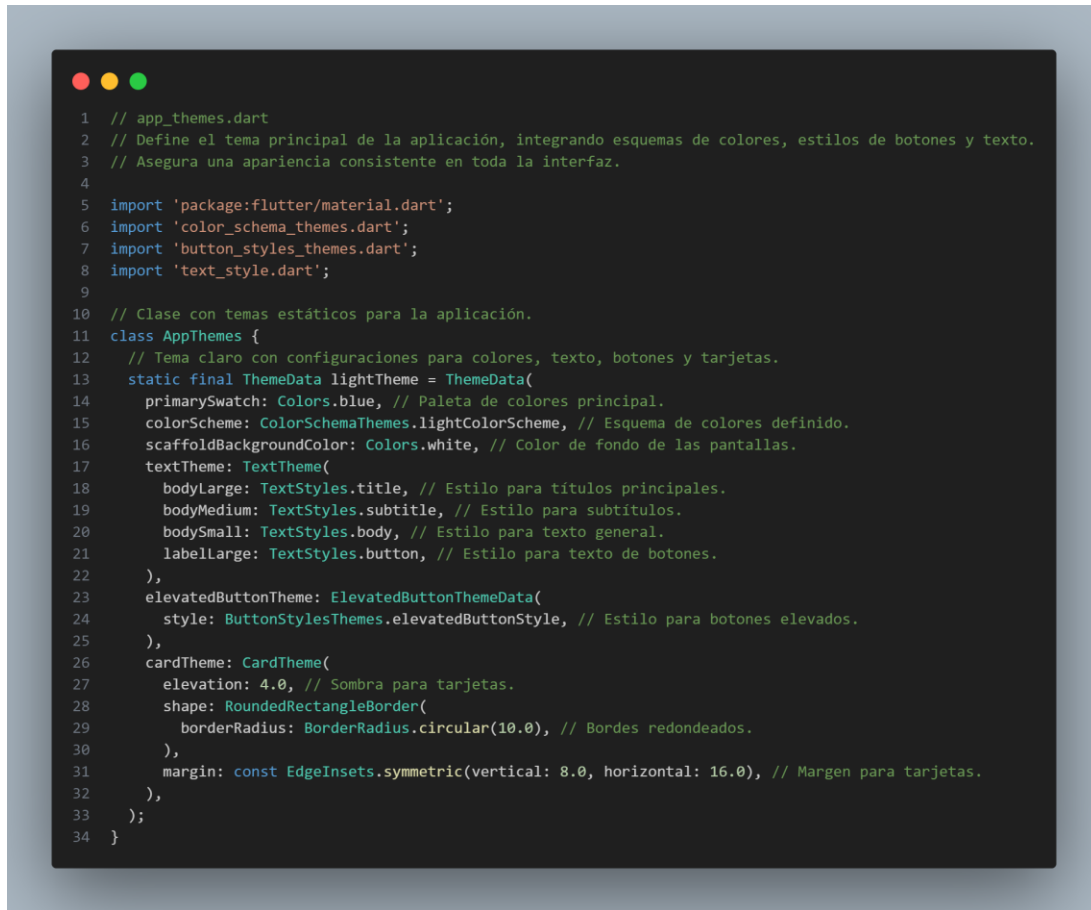
```
1 // anio_model.dart
2 // Define el modelo de datos para un año y su lógica de verificación de bisiesto.
3 // Encapsula la lógica de negocio para determinar si un año es bisiesto.
4
5 class AnioModel {
6   final int anio; // Año a verificar.
7
8   // Constructor que inicializa el año.
9   AnioModel(this.anio);
10
11   // Verifica si el año es bisiesto según las reglas:
12   // - Divisible por 4 y no por 100, o divisible por 400.
13   bool esBisiesto() {
14     return (anio % 4 == 0 && anio % 100 != 0) || (anio % 400 == 0);
15   }
16 }
```

Nota. Este modelo define la regla para saber si un año es bisiesto según criterios matemáticos.

- **themes:** Definen el estilo visual de la app, como colores, tipografía y componentes gráficos consistentes.

Figura 3

Configuración del tema visual de la aplicación



```
1 // app_themes.dart
2 // Define el tema principal de la aplicación, integrando esquemas de colores, estilos de botones y texto.
3 // Asegura una apariencia consistente en toda la interfaz.
4
5 import 'package:flutter/material.dart';
6 import 'color_schema_themes.dart';
7 import 'button_styles_themes.dart';
8 import 'text_style.dart';
9
10 // Clase con temas estáticos para la aplicación.
11 class AppThemes {
12   // Tema claro con configuraciones para colores, texto, botones y tarjetas.
13   static final ThemeData lightTheme = ThemeData(
14     primarySwatch: Colors.blue, // Paleta de colores principal.
15     colorScheme: ColorSchemaThemes.lightColorScheme, // Esquema de colores definido.
16     scaffoldBackgroundColor: Colors.white, // Color de fondo de las pantallas.
17     textTheme: TextTheme(
18       bodyLarge: TextStyles.title, // Estilo para títulos principales.
19       bodyMedium: TextStyles.subtitle, // Estilo para subtítulos.
20       bodySmall: TextStyles.body, // Estilo para texto general.
21       labelLarge: TextStyles.button, // Estilo para texto de botones.
22     ),
23     elevatedButtonTheme: ElevatedButtonThemeData(
24       style: ButtonStylesThemes.elevatedButtonStyle, // Estilo para botones elevados.
25     ),
26     cardTheme: CardTheme(
27       elevation: 4.0, // Sombra para tarjetas.
28       shape: RoundedRectangleBorder(
29         borderRadius: BorderRadius.circular(10.0), // Bordes redondeados.
30       ),
31       margin: const EdgeInsets.symmetric(vertical: 8.0, horizontal: 16.0), // Margen para tarjetas.
32     ),
33   );
34 }
```

Nota. Define los colores, estilos de texto y botones para mantener una apariencia uniforme en toda la app.

Figura 4

Estilo personalizado para botones elevados

```
1 // button_styles_themes.dart
2 // Define estilos reutilizables para botones elevados.
3 // Asegura consistencia en el diseño de botones en toda la aplicación.
4
5 import 'package:flutter/material.dart';
6
7 // Clase con estilos estáticos para botones.
8 class ButtonStylesThemes {
9   // Estilo para botones elevados con diseño uniforme.
10   static ButtonStyle elevatedButtonStyle = ElevatedButton.styleFrom(
11     backgroundColor: Colors.blue, // Color de fondo del botón.
12     foregroundColor: Colors.white, // Color de texto e iconos.
13     padding: const EdgeInsets.symmetric(horizontal: 24.0, vertical: 12.0), // Relleno intern
14     textStyle: const TextStyle(fontSize: 16.0), // Estilo de texto.
15     shape: RoundedRectangleBorder(
16       borderRadius: BorderRadius.circular(8.0), // Bordes redondeados.
17     ),
18   );
19 }
```

Nota. Define un diseño uniforme para los botones, mejorando la coherencia visual en toda la app.

Figura 5

```
1 // color_schema_themes.dart
2 // Define un esquema de colores para la aplicación, asegurando consistencia visual.
3 // Complementa el tema principal definido en app_themes.dart.
4
5 import 'package:flutter/material.dart';
6
7 // Clase con esquemas de colores estáticos para diferentes modos.
8 class ColorSchemaThemes {
9   // Esquema de colores para el tema claro.
10   static const ColorScheme lightColorScheme = ColorScheme(
11     brightness: Brightness.light, // Modo claro.
12     primary: Colors.blue, // Color principal para botones y acentos.
13     onPrimary: Colors.white, // Color de texto sobre el color principal.
14     secondary: Colors.blueAccent, // Color secundario para variaciones.
15     onSecondary: Colors.white, // Color de texto sobre el color secundario.
16     error: Colors.red, // Color para mensajes de error.
17     onError: Colors.white, // Color de texto sobre el color de error.
18     surface: Colors.white, // Color de fondo para tarjetas y superficies.
19     onSurface: Colors.black87, // Color de texto sobre superficies.
20     background: Colors.white, // Color de fondo general.
21     onBackground: Colors.black87, // Color de texto sobre el fondo.
22   );
23 }
```

Nota. Define colores base y de texto para mantener una apariencia visual coherente en toda la app.

Figura 6

Contenedores reutilizables con diseño personalizado

```
1 // contenedor_layouts.dart
2 // Proporciona widgets reutilizables para contenedores y tarjetas con estilos consistentes.
3 // Ayuda a mantener un diseño uniforme en toda la aplicación.
4
5 import 'package:flutter/material.dart';
6
7 // Clase con métodos estáticos para crear contenedores estilizados.
8 class ContenedorLayouts {
9   // Crea un contenedor con relleno personalizado.
10  static Widget paddedContainer({
11    required Widget child, // Widget hijo a envolver.
12    EdgeInsets padding = const EdgeInsets.all(16.0), // Relleno por defecto.
13  }) {
14    return Container(
15      padding: padding,
16      child: child,
17    );
18  }
19
20  // Crea una tarjeta con estilos consistentes para listas.
21  static Widget cardContainer({
22    required Widget child, // Widget hijo a envolver.
23    EdgeInsets margin = const EdgeInsets.symmetric(vertical: 8.0, horizontal: 16.0), // Margen por defecto.
24    double elevation = 4.0, // Elevación para sombra.
25  }) {
26    return Card(
27      margin: margin,
28      shape: RoundedRectangleBorder(
29        borderRadius: BorderRadius.circular(10.0), // Bordes redondeados.
30      ),
31      elevation: elevation,
32      child: child,
33    );
34  }
35 }
```

Nota. Permite crear tarjetas y contenedores con estilos uniformes y consistentes en toda la app.

Figura 7

Estilos tipográficos reutilizables para la aplicación

```
1 // text_style.dart
2 // Define estilos de texto reutilizables para la aplicación.
3 // Asegura consistencia tipográfica en toda la interfaz.
4
5 import 'package:flutter/material.dart';
6
7 // Clase con estilos de texto estáticos para diferentes usos.
8 class TextStyles {
9   // Estilo para títulos principales.
10  static const TextStyle title = TextStyle(
11    fontSize: 20.0,
12    fontWeight: FontWeight.bold,
13    color: Colors.black87,
14  );
15
16  // Estilo para subtítulos o texto secundario.
17  static const TextStyle subtitle = TextStyle(
18    fontSize: 16.0,
19    color: Colors.black54,
20  );
21
22  // Estilo para texto general del cuerpo.
23  static const TextStyle body = TextStyle(
24    fontSize: 14.0,
25    color: Colors.black87,
26  );
27
28  // Estilo para texto en botones.
29  static const TextStyle button = TextStyle(
30    fontSize: 16.0,
31    fontWeight: FontWeight.w500,
32    color: Colors.white,
33  );
34 }
```

Nota. Establece formatos de texto coherentes para títulos, subtítulos, cuerpo y botones.

- **views:** Contienen las interfaces gráficas que el usuario interactúa, mostrando resultados y recibiendo entradas.

Figura 8

Vista para mostrar años bisiestos en formato de lista

```
1 // resultado_views.dart
2 // Define la interfaz de usuario para mostrar una lista de años bisiestos anteriores.
3 // Usa un ListView para mostrar los años en tarjetas estilizadas.
4
5 import 'package:flutter/material.dart';
6 import '../controllers/anio_controller.dart';
7
8 // ResultadoView es un widget sin estado que muestra los últimos años bisiestos.
9 class ResultadoView extends StatelessWidget {
10   final AnioController controller; // Controlador para obtener años bisiestos.
11
12   // Constructor requiere un controlador para inyección de dependencias.
13   const ResultadoView({super.key, required this.controller});
14
15   @override
16   Widget build(BuildContext context) {
17     // Obtiene el año inicial desde los argumentos de la ruta, usa el año actual si no hay argumentos.
18     final args = ModalRoute.of(context)!.settings.arguments as Map<String, dynamic>;
19     final int desde = args?['anio'] ?? DateTime.now().year;
20
21     // Obtiene la lista de años bisiestos desde el controlador.
22     final lista = controller.obtenerUltimos(desde);
23
24     return Scaffold(
25       appBar: AppBar(
26         title: const Text('Lista de Años Bisiestos'), // Título de la barra de aplicación.
27         backgroundColor: Theme.of(context).primaryColor, // Color primario del tema.
28       ),
29       body: Padding(
30         padding: const EdgeInsets.all(16.0), // Relleno consistente para el contenido.
31         child: ListView.builder(
32           itemCount: lista.length, // Número de años en la lista.
33           itemBuilder: (context, index) {
34             final anio = lista[index]; // Año bisiesto en la posición actual.
35             return Card(
36               margin: const EdgeInsets.symmetric(vertical: 8.0), // Margen para separación visual.
37               shape: RoundedRectangleBorder(
38                 borderRadius: BorderRadius.circular(10.0), // Bordes redondeados para la tarjeta.
39               ),
40               elevation: 4.0, // Sombra para dar profundidad.
41               child: ListTile(
42                 title: Text(
43                   'Año $anio', // Muestra el año.
44                   style: Theme.of(context).textTheme.bodyLarge, // Estilo de texto del tema.
45                 ),
46                 subtitle: Text(
47                   'Es bisiesto', // Texto descriptivo.
48                   style: Theme.of(context).textTheme.bodyMedium, // Estilo de subtítulo del tema.
49                 ),
50               ),
51             );
52           },
53         ),
54       ),
55     );
56   }
57 }
```

Nota. Presenta los años bisiestos en tarjetas mediante una lista visualmente organizada y reutilizable.

Figura 9

Pantalla de verificación de años bisiestos

```
1 // verificar_views.dart
2 // Define la interfaz de usuario para la pantalla de verificación de años bisiestos.
3 // Permite al usuario ingresar un año y muestra una notificación con el resultado.
4
5 import 'package:flutter/material.dart';
6 import '../controllers/anio_controller.dart';
7
8 // VerificarView es un widget sin estado para ingresar y verificar un año.
9 class VerificarView extends StatelessWidget {
10   final AnioController controller; // Controlador para la lógica de años bisiestos.
11   final TextEditingController _anioC = TextEditingController(); // Gestiona la entrada de texto.
12
13   // Constructor requiere un controlador para inyección de dependencias.
14   VerificarView({super.key, required this.controller});
15
16   @override
17   Widget build(BuildContext context) {
18     return Scaffold(
19       appBar: AppBar(
20         title: const Text('Verificar Año Bisiesto'), // Título de la barra de aplicación.
21         backgroundColor: Theme.of(context).primaryColor, // Usa el color primario del tema.
22       ),
23       body: Padding(
24         padding: const EdgeInsets.all(16.0), // Relleno consistente para el contenido.
25         child: Column(
26           children: [
27             TextField(
28               controller: _anioC, // Vincula la entrada al controlador de texto.
29               keyboardType: TextInputType.number, // Restringe la entrada a números.
30               decoration: InputDecoration(
31                 labelText: 'Ingresa un año:', // Etiqueta del campo de entrada.
32                 labelStyle: Theme.of(context).textTheme.bodyMedium, // Estilo consistente con el tem
33               ),
34             ),
35           ],
36         ),
37       ),
38     );
39   }
40 }
```

Nota. Permite al usuario ingresar un año, verificar si es bisiesto y mostrar el resultado con una notificación visual.

Figura 10

Archivo principal y configuración de rutas de la app Flutter

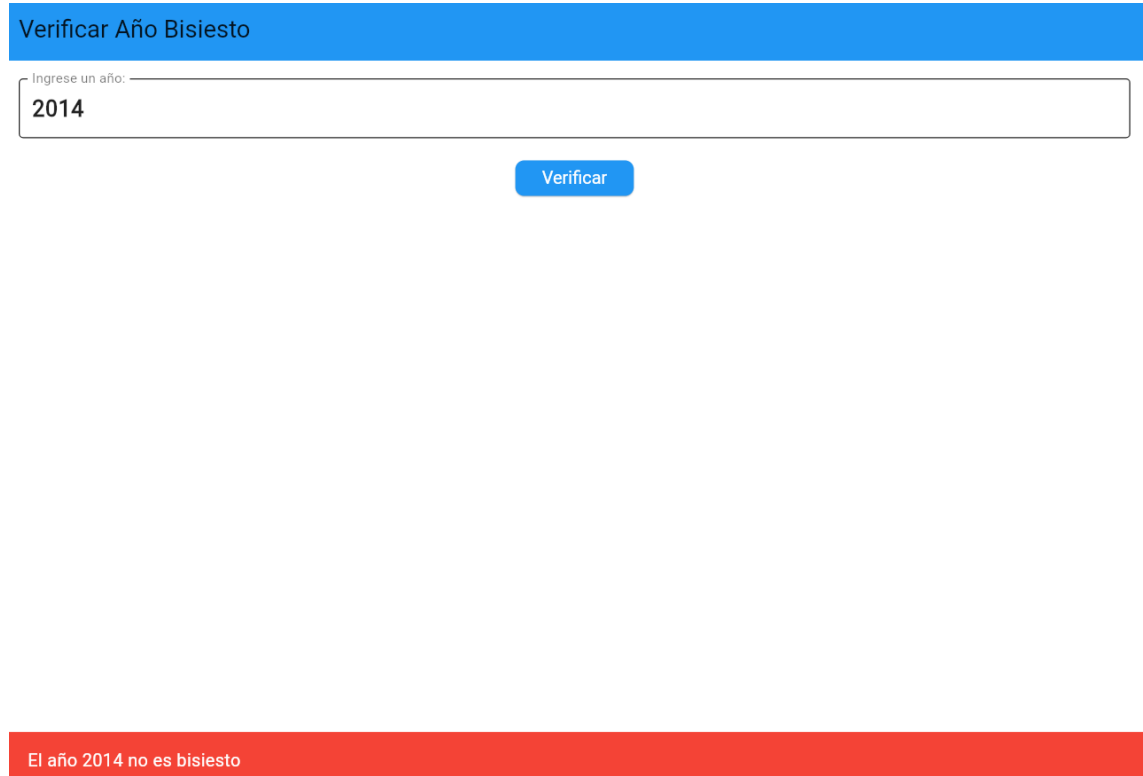
```
1 // main.dart
2 // Punto de entrada de la aplicación Flutter para verificar años bisiestos.
3 // Configura el tema y las rutas de navegación de la aplicación.
4
5 import 'package:flutter/material.dart';
6 import 'controllers/anio_controller.dart';
7 import 'views/resultado_views.dart';
8 import 'views/verificar_views.dart';
9 import 'themes/app_themes.dart';
10
11 void main() {
12   // Inicia y ejecuta la aplicación Flutter.
13   runApp(const MyApp());
14 }
15
16 // MyApp es el widget raíz que define el tema y las rutas de la aplicación.
17 class MyApp extends StatelessWidget {
18   const MyApp({super.key});
19
20   @override
21   Widget build(BuildContext context) {
22     // Instancia única del controlador para compartir lógica entre vistas.
23     final controller = AnioController();
24
25     return MaterialApp(
26       title: 'Año Bisiesto', // Título de la app para accesibilidad y marca.
27       theme: AppThemes.lightTheme, // Aplica tema claro consistente en toda la app.
28       initialRoute: '/', // Ruta inicial para la vista de verificación.
29       routes: {
30         // Mapea rutas a sus respectivas vistas, pasando el controlador compartido.
31         '/': (context) => VerificarView(controller: controller),
32         '/resultado': (context) => ResultadoView(controller: controller),
33       },
34       debugShowCheckedModeBanner: false, // Oculta la bandera de depuración en producción
35     );
36   }
37 }
```

Nota. Este código inicia la aplicación, aplica el tema visual y define las rutas para navegar entre las vistas de verificación y resultados.

V. Resultados

Figura 11

Verificación de año no bisiesto en la interfaz de la aplicación

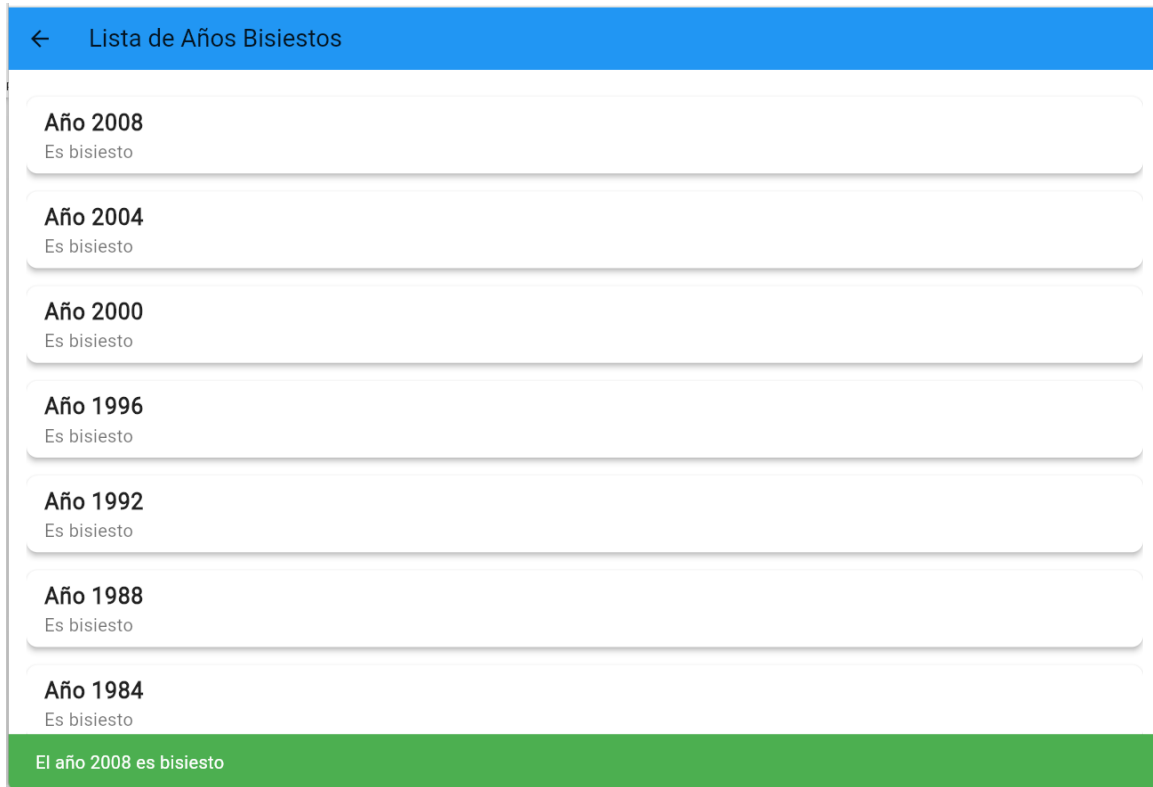


The screenshot shows a mobile application interface with a blue header bar containing the text "Verificar Año Bisiesto". Below the header is a text input field with the placeholder text "Ingrese un año:" and the value "2014". To the right of the input field is a blue button labeled "Verificar". At the bottom of the screen, there is a red banner with the text "El año 2014 no es bisiesto".

Nota. La figura 11 muestra la interfaz de una app en Flutter que verifica si un año es bisiesto. Al ingresar 2014 y presionar "Verificar", se despliega un mensaje indicando que no es bisiesto, resaltado en rojo. Esto confirma que la lógica del sistema funciona correctamente. La interfaz es clara, intuitiva y facilita la interacción del usuario. En general, se cumplen los objetivos del proyecto con una experiencia visual efectiva.

Figura 12

Visualización de los últimos años bisiestos



Nota. La figura 12 muestra una lista ordenada de años bisiestos como 2008, 2004 y 2000. Al seleccionar uno, aparece un mensaje en verde que confirma que es bisiesto. El diseño claro y la retroalimentación visual positiva facilitan la comprensión del usuario y demuestran que la app cumple correctamente su función.

VII. Conclusiones

- Este proyecto nos ayudó a entender cómo organizar bien una app en Flutter, separando cada parte del código para que sea más fácil de manejar y mejorar a futuro.
- Aunque la función principal es simple, fue una excelente forma de aplicar buenas prácticas de programación y aprender cómo se estructura una app real.
- El diseño limpio y fácil de usar hizo que la experiencia sea agradable. Para próximos proyectos, sería interesante agregar nuevas funciones que sigan enseñando sobre desarrollo móvil.

Referencias

[1] D. K. Chicaiza, Lección 1.3: Principales componentes en una aplicación móvil, Universidad de las Fuerzas Armadas ESPE. [Presentación], sin fecha.

Link del git hub

<https://github.com/ANTHONYNESTORVILLARREALMACIAS/Desarrollo-Aplicaciones-Moviles-Chimba-Ramos-Reyes-Villarreal.git>