# fluter

Al crear un proyecto nuestro código se va a encontrar en la carpeta /lib

Se van a ver un carpeta por cada plataforma

Los test van a estar un carpeta /test

# pubspec.lock

Almacena información sobre las librerias usadas

# pubspec.yaml

tienes varias configuraciones generales del proyecto, como el nombre o la versión de andriod objetivo

# main

import 'package:flutter/material.dart';

void main() {

  runApp(MyApp());

}

runApp recibe un Widget como argumento

# Widget

Hay de dos tipos

## Stateless

Es una pieza de lego que se construye muy rápido y no mantiene el estado por si mismo. Flutter sabe cuando se de be de volver a dibujar, es recomendado que en lugar de crear funciones y métodos que retornan Widget, es mejor crear Clases que extiendan de estos StatelesWidgets.

No tiene estado (entiéndase como variables de estado al nivel de este widget o inferior)

class MyApp extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    // TODO: implement build

    return MaterialApp(

      home: Center(child: Text("hola mundo")),

    );

  }

}

### Constructor con constante

Por alguna razón es buena idea declarar el constructor como a continuación, se le agrega el const si el widget una vez construido nunca va a cambiar

class MyApp extends StatelessWidget {

  const MyApp({super.key});

  @override

  Widget build(BuildContext context) {

    // TODO: implement build

    return const MaterialApp(

      debugShowCheckedModeBanner: false,

      home: CounterScreem(),

    );

  }

}

## Stateful

Es similar al stateless en cuanto a que es un Widget, pero este permite mantener un estado interno y ciclo de vida como su inicialización y destrucción. Muchos lo tachan de que jamás se deben de usar pero eso no es cierto, los stateful básicamente son el corazón de cualquier animación que suceda.

Si tiene estado (entiéndase como variables de estado al nivel de este widget o inferior)

class CounterScreem extends StatefulWidget {

  const CounterScreem({super.key});

  @override

  State<CounterScreem> createState() => \_CounterScreemState();// clase que representa el estado

}

class \_CounterScreemState extends State<CounterScreem> {

  final int clickCounter = 0;

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        title: const Text('el titulo'),

      ),

      body: Center(

        child: Column(

          mainAxisAlignment: MainAxisAlignment.center,

          children: [

            Text(

              "a",

              style: TextStyle(fontSize: 10, fontWeight: FontWeight.w100),

            ),

            Text("b")

          ],

        ),

      ),

      floatingActionButton: FloatingActionButton(

        onPressed: () {},

        child: Icon(Icons.plus\_one),

      ),

    );

  }

}

## Text

Text("hola mundo")

## Center

Center(child: Text("hola mundo"))

## MaterialApp

MaterialApp(

      home: Center(child: Text("hola mundo")),

    );

### Con temas y sin debug

MaterialApp(

      debugShowCheckedModeBanner: false,

      home: CounterScreem(),

      theme: ThemeData(useMaterial3: true, colorSchemeSeed: Colors.red),

    );

## Column

Column(

          mainAxisAlignment: MainAxisAlignment.center,

          children: [Text("a"), Text("b")],

        )

## Icon

Icon(Icons.plus\_one)

## Botones

### Propiedades comunes

#### enableFeedback

ofrece una reacción a las acciones del usuario (en este caso presionarlo) osea si se puede el dispositivo vibra, sino suena un sonido suabe

FloatingActionButton(

      enableFeedback: true,

      onPressed: onPressed,

      child: Icon(icon),

    );

#### Elevation

Aumenta la profundidad de la sombra

FloatingActionButton(

      enableFeedback: true,

      elevation: 5,

      onPressed: onPressed,

      child: Icon(icon),

    );

### FloatingActionButton

Botón flotante

FloatingActionButton(

            shape: const StadiumBorder(),

            onPressed: () {

              clickCounter++;

              setState(() {});

            },

            child: Icon(Icons.plus\_one),

          ),

### IconButton

IconButton(

          onPressed: () {},

          icon: Icon(Icons.refresh\_rounded),

        ),

### FilledButton

illedButton.tonal(

              onPressed: () {}, child: const Text('Click me')),

        )

## Scaffold

Scaffold(

      appBar: AppBar(

        title: const Text('el titulo'),

      ),

      body: Center(

        child: Column(

          mainAxisAlignment: MainAxisAlignment.center,

          children: [

            Text(

              "a",

              style: TextStyle(fontSize: 10, fontWeight: FontWeight.w100),

            ),

            Text("b")

          ],

        ),

      ),

      floatingActionButton: FloatingActionButton(

        onPressed: () {},

        child: Icon(Icons.plus\_one),

      ),

    );

## AppBar

AppBar(

        title: const Text('el titulo'),

      )

AppBar(

        title: const Text('el titulo'),

        leading: IconButton(

          onPressed: () {},

          icon: Icon(Icons.refresh\_rounded),

        ),

      )

AppBar(

        title: const Text('el titulo'),

        leading: IconButton(

          onPressed: () {},

          icon: const Icon(Icons.refresh\_rounded),

        ),

        actions: [

          IconButton(

            onPressed: () {},

            icon: const Icon(Icons.refresh\_rounded),

          ),

          IconButton(

            onPressed: () {},

            icon: const Icon(Icons.refresh\_rounded),

          )

        ],

      ),

## SizedBox

Un seperador, generalmente se usa entre elementos en una lista de widgets

const SizedBox(

            height: 10,

          )

## ClipRRect

ClipRRect(

      borderRadius: BorderRadius.circular(20),

      child: Image.network(

          "https://yesno.wtf/assets/yes/15-3d723ea13af91839a671d4791fc53dcc.gif"),

    );

# return const

si en una funcion se pone de esta manera en el return flutter optimiza el código resultante para no esperar un cambio de este código ?

class MyApp extends StatelessWidget {

  const MyApp({super.key});

  @override

  Widget build(BuildContext context) {

    // TODO: implement build

    return const MaterialApp(

      home: Scaffold(

        body: Center(

          child: Text('hola mundo'),

        ),

      ),

    );

  }

}

# TextStyle

Algunos componentes los reciben el parámetro style

Text("el texto",

              style: TextStyle(fontSize: 10, fontWeight: FontWeight.w100),

            ),

# ThemeData

ThemeData(useMaterial3: true, colorSchemeSeed: Colors.red),

## Modo oscuro

ThemeData(

        useMaterial3: true,

        colorSchemeSeed: \_colorThemes[selectedColor],

        brightness: Brightness.dark);

  }

# Estado a nivel de widget

Sus hijos pueden acceder a el

# Estado global de la aplicación

Es un estado que todos los widgets sin importar el lugar donde se encuentren dentro del árbol de widtgets

# Actualizar estado

Para que se actualice se tiene que llamar a: setState

onPressed: () {

          setState(() {

            clickCounter++;

          });

        },

O solo llamarlo a parte

onPressed: () {

          clickCounter++;

          setState(() {});

        },

Pq cuando se llama busca estados desactualizados y los actualiza

Se renderiza todo el widget a partir del estado que se modificó (osea la clase donde se encuentra el atributo)

Aunque en teoría al actualizar el componente comprueba si es necesario hacer cambios en cada subcomponente y en caso de que lo sea lo actualiza

# Const argumentos

Se puede poner const a argumentos específicos cuyo valor no va a cambiar

Text('Click${clickCounter > 1 ? "s" : ""}',

              style: const TextStyle(fontSize: 25),

            )

# Bordes

## StadiumBorder

Boredes circulares

const StadiumBorder(),

FloatingActionButton(

            shape: const StadiumBorder(),

            onPressed: () {

              clickCounter++;

              setState(() {});

            },

            child: Icon(Icons.plus\_one),

          ),

BorderRadius

BorderRadius.circular(20)

Container(

          decoration: BoxDecoration(

              color: Colors.black, borderRadius: BorderRadius.circular(20)),

          child: Padding(

            padding: const EdgeInsets.symmetric(horizontal: 20, vertical: 10),

            child: Text(

              'el texto',

              style: TextStyle(color: Colors.blue),

            ),

          ),

        ),

# VoidCallback

final VoidCallback? onPressed;

FloatingActionButton(

      onPressed: onPressed,

      child: Icon(icon),

    );

Colores

const Color \_customColor = Color.fromARGB(15, 211, 109, 107);

const List<Color> \_colorThemes = [

 Color.fromARGB(255, 44, 36, 156),

  Colors.red

];

# Padding

Padding(padding: const EdgeInsets.all(8.0),

          child: CircleAvatar(),

        )

Padding(

        padding: const EdgeInsets.symmetric(horizontal: 10),

        child: Column(

          children: [

            Expanded(

                child: Container(

              color: Colors.red,

            )),

            Text("mundo")

          ],

        ),

      )

# CircleAvatar

CircleAvatar(

            backgroundImage: NetworkImage("la direccion en internet"),

          )

CircleAvatar(

            backgroundColor: Colors.red,

          ),

# ListView

Expanded(child: ListView.builder(itemBuilder: (contex, index) {

              return Text("$index");

            }))

## Cantidad limite itemCount

ListView.builder(

                    itemCount: 100,

                    itemBuilder: (contex, index) {

                      return Text("$index");

                    })

## Controlador

La idea es crear el controlador dentro del proveedor para que al notificar el cambio la barra baje

Para el movimiento se puede usar el método animateTo

class ChatProvider extends ChangeNotifier {

  final chatScrollController = ScrollController();

  List<Message> messageList = [

    Message(text: "Hola", fromWho: FromWho.me),

    Message(text: "dime", fromWho: FromWho.her)

  ];

  Future<void> sendMessage(String text) async {

    final newMessage = Message(text: text, fromWho: FromWho.me);

    messageList.add(newMessage);

    notifyListeners();

    moveToScrollToBottom();

  }

  void moveToScrollToBottom() {

    chatScrollController.animateTo(

        chatScrollController.position.maxScrollExtent,

        duration: const Duration(seconds: 300),

        curve: Curves.easeOut);

  }

}

  @override

  Widget build(BuildContext context) {

    final chatProvider = context.watch<ChatProvider>();

…

ListView.builder(

                    controller: chatProvider.chatScrollController,

                    itemCount: chatProvider.messageList.length,

                    itemBuilder: (contex, index) {

                      final message = chatProvider.messageList[index];

                      return message.fromWho == FromWho.her

                          ? const HerMessageBubble()

                          : MyMessageBubble(message: message);

                    })),

# Container

Es como un div

Para definir los estilos de su caja es el atributo decoration al que se le puede pasar un BoxDecoration

Container(

          decoration: BoxDecoration(

              color: Colors.black, borderRadius: BorderRadius.circular(20)),

          child: Padding(

            padding: const EdgeInsets.symmetric(horizontal: 20, vertical: 10),

            child: Text(

              'el texto',

              style: TextStyle(color: Colors.blue),

            ),

          ),

        ),

# Escema de colores

Para obtener todos los colores del tema principal

class MyMessageBubble extends StatelessWidget {

  const MyMessageBubble({super.key});

  @override

  Widget build(BuildContext context) {

    final colors = Theme.of(context).colorScheme;

    return Column(

      crossAxisAlignment: CrossAxisAlignment.end,

      children: [

        Container(

          decoration: BoxDecoration(

              color: colors.primary, borderRadius: BorderRadius.circular(20)),

          child: Padding(

            padding: const EdgeInsets.symmetric(horizontal: 20, vertical: 10),

            child: Text(

              'el texto',

              style: TextStyle(color: Colors.blue),

            ),

          ),

        ),

        SizedBox(

          height: 10,

        )

      ],

    );

  }

}

# Imagenes

Image.network(

          "https://yesno.wtf/assets/yes/15-3d723ea13af91839a671d4791fc53dcc.gif"),

    )

## Cover

Image.network(

          "https://yesno.wtf/assets/yes/15-3d723ea13af91839a671d4791fc53dcc.gif",

          width: size.width \* 0.7,

          height: 150,

          fit: BoxFit.cover)

# MediaQuery

Con el y el context se pueden obtener las propiedades del dispositivo

## Size

final size = MediaQuery.of(context).size;

# Entrada de texto

TextFormField();

TextFormField(

      decoration: InputDecoration(

          enabledBorder: OutlineInputBorder(

              borderSide: BorderSide(color: colors.primary),

              borderRadius: BorderRadius.circular(20)),

          filled: true,

          suffixIcon: IconButton(

            icon: const Icon(Icons.send\_outlined),

            onPressed: () {},

          )),

    );

## TextEditingController

El input se controla a travez de un TextEditingController y este se le pasa en el parámetro controller

final textController = TextEditingController();

    final colors = Theme.of(context).colorScheme;

    final outlineInputBorder = UnderlineInputBorder(

        borderSide: const BorderSide(color: Colors.transparent),

        borderRadius: BorderRadius.circular(20));

    final inputDecoration = InputDecoration(

        hintText: 'termina tu mensaje con ??',

        enabledBorder: outlineInputBorder,

        focusedBorder: outlineInputBorder,

        filled: true,

        suffixIcon: IconButton(

          icon: const Icon(Icons.send\_outlined),

          onPressed: () {

            final textValue = textController.value.text;

          },

        ));

    return TextFormField(

      controller: textController,

      decoration: inputDecoration,

      onFieldSubmitted: (value) {

        textController.clear();

      },

      onChanged: (value) {},

    );

### clear

textController.clear();

vacía el contenido

### obtener el texto

final textValue = textController.value.text;

## FocusNode

final textController = TextEditingController();

    final colors = Theme.of(context).colorScheme;

    final focusNode = FocusNode();

    final outlineInputBorder = UnderlineInputBorder(

        borderSide: const BorderSide(color: Colors.transparent),

        borderRadius: BorderRadius.circular(20));

    final inputDecoration = InputDecoration(

        hintText: 'termina tu mensaje con ??',

        enabledBorder: outlineInputBorder,

        focusedBorder: outlineInputBorder,

        filled: true,

        suffixIcon: IconButton(

          icon: const Icon(Icons.send\_outlined),

          onPressed: () {

            final textValue = textController.value.text;

          },

        ));

    return TextFormField(

      onTapOutside: (event) {

        focusNode.unfocus();

      },

      focusNode: focusNode,

      controller: textController,

      decoration: inputDecoration,

      onFieldSubmitted: (value) {

        textController.clear();

        focusNode.requestFocus();

      },

      onChanged: (value) {},

    );

  }

### obtener el foco

focusNode.requestFocus();

### Perder el foco

focusNode.unfocus();

# Gestores de estado

Manejar el estado de la aplicación es algo de nivel intermedio, hay muchas formas de manejarlo, desde third-party y first-party.

A continuación una breve descripción sobre los más populares. (En base a experiencia personal)

## Provider:

Es el recomendado por Flutter, en lo personal lo recomiendo para empezar, es robusto, fácil de aprender y usar. (Flutter Favorite)

### Usarlo

Se debe crear una clase que extienda de

import 'package:flutter/material.dart';

class ChatProvider extends ChangeNotifier {}

hay que instalar

* flutter pub add provider

se va a ver reflejado en pubspec.yaml

dependencies:

  flutter:

    sdk: flutter

  # The following adds the Cupertino Icons font to your application.

  # Use with the CupertinoIcons class for iOS style icons.

  cupertino\_icons: ^1.0.6

  provider: ^6.1.2

#### MultiProvider

En el padre desde donde se quieren que sus hijos tengan acceso al provider (puede ser el build del que este en el main) se utiliza el MultiProvider que recibe una lista de proveedores ChangeNotifierProvider

MultiProvider(

      providers: [ChangeNotifierProvider(create: (\_) => ChatProvider())],

      child: MaterialApp(

        debugShowCheckedModeBanner: false,

        theme: AppTheme(selectedColor: 2).theme(),

        title: 'Material App',

        home: const ChatScreen(),

      ),

    );

Luego en algun hijo

import 'package:provider/provider.dart';

  @override

  Widget build(BuildContext context) {

    final chatProvider = context.watch<ChatProvider>();

}

#### notifyListeners();

Cuando existe un cambio en algun elemento de los proveedores se llama a notifyListeners(); para que se manade dentro de un ChangeNotifier para mandar a actualizar a todos los widgets que escuchan a este proveedor

class ChatProvider extends ChangeNotifier {

  List<Message> messageList = [

    Message(text: "Hola", fromWho: FromWho.me),

    Message(text: "dime", fromWho: FromWho.her)

  ];

  Future<void> sendMessage(String text) async {

    final newMessage = Message(text: text, fromWho: FromWho.me);

    messageList.add(newMessage);

    notifyListeners();

  }

}

## Riverpod:

Es otro muy bueno y recomendado, fácilmente testeable.

## InheritedWidget & InheritedModel:

Esta es la opción nativa de Flutter, no dependencias de terceros, algo tediosa a mi parecer, pero es la solución oficial.

## BLoC / RX y Flutter BLoC + Cubits:

Es un gestor de estado basado en Streams y Observables, no es el más fácil para empezar, pero da mucho control. A muchas personas no les gusta por que se escribe mucho código. (Flutter Farorite)

## Get\_it:

Técnicamente no es un gestor de estado, pero es muy útil para ir puntualmente a un servicio u objeto sin necesidad del BuildContext.

## MobX:

Es popular y basado en observables y reacciones. (Flutter Favorite)

## GetX:

Actualmente el paquete y gestor de estado más popular, hace muchas cosas aparte de manejar estados, tiene navegación, snackbars, internacionalización, temas, validaciones. Es considerado un Framework dentro de Flutter. Yo en lo personal NO lo recomiendo para aprender primero ya que oculta conceptos que son necesarios aprender (Como el BuildContext), pero es muy poderoso.

## Pro Tip:

Prueben varios gestores de estado, miren cuál es el que más les gusta, sirve y resuelve su necesidad. Creen su propia opinion sobre ellos y escuchen a la comunidad, pero al final del día, ustedes tienen la última palabra, NADIE de la comunidad le dará mantenimiento a tu código, por eso deben de saber cuál es mejor para ti.

# Dio

* flutter pud add dio

Para comunicación http