

ÍNDICE

| | |
|---|-----------|
| Vídeo 1. Primer paso del curso. | 2 |
| Vídeo 2 y 3. Estructura de una App. Principales Widgets. | 4 |
| Vídeo 4. Filas , Columnas y Botones. | 4 |
| Vídeo 5. Widgets separados. | 6 |
| Vídeo 6. Navegación | 8 |
| main.dart | 8 |
| pagina02.dart | 8 |
| Vídeo 7. Alert Dialog | 10 |
| body | 10 |
| mostrarAlerta | 11 |
| Vídeo 8. Consumiendo una API REST >> clases | 12 |
| Vídeo 9. Consumiendo API REST >> Listas | 14 |
| Primero, creamos una clase. Al final del programa, fuera de todo Widget | 14 |
| Segundo, creamos un array (se llaman List) de objetos de la clase Persona. | 14 |
| Tercero, dentro del body creamos un objeto ListView con builder. | 15 |
| Cuarto, por fin la función _borrarpersona | 15 |
| Vídeo 10 y 11. API REST >> componente http | 16 |
| Por fin, llamada a la función _misGifs | 19 |
| Vídeo 12. Barra de navegación inferior | 19 |
| Quitar etiqueta de desarrollo | 19 |
| Parte del Scaffold, el bottomNavigationBar, | 19 |
| Dos contenidos distintos, del tipo statelesswidget | 20 |
| Selección del contenido | 21 |
| A) Si hay sólo dos widgets | 21 |
| B) Si hay más de dos widgets | 21 |
| Vídeo 13. Animar contenedores | 22 |
| Vídeo 14. Carrusel de imágenes o swiper. | 24 |
| Vídeo 15. Persistencia de datos | 26 |
| Vídeo 16. Pasar una información de un widget a otro (página a otra). | 28 |
| Página principal | 28 |
| Página de tipo stateless | 30 |
| Página de tipo statefull | 30 |
| Vídeo 18. Firebase | 31 |
| Vídeo 21. Splashscreen (android) | 33 |
| Vídeo 22. Splashscreen con IOS → No lo hago | 33 |
| Vídeo 23. Cambiar el logo de tu APP (Android) y 24. Como un pro | 34 |
| Cambiar icono de forma manual | 34 |
| Cambiar icono con un paquete | 34 |

| | |
|--|-----------|
| Cambiar icono de forma profesional | 34 |
| Vídeo 25. Drawer | 35 |
| Vídeo 26 y 27. Textformfield personalizados | 36 |
| Vídeo 28. Notificaciones locales | 37 |
| Vídeo 29. Sistemas de rutas | 40 |
| 1ª forma) | 40 |
| 2ª forma) | 40 |
| Por si hay errores en la llamada | 41 |
| Por si necesito que haya una página en nivel superior sin la ← | 41 |
| Export páginas | 42 |
| Vídeo 30-34. Firebase CRUD | 42 |
| Vídeo 35. Mini Galería | 42 |
| Vídeo 36 - 37. FireBase Storage | 47 |
| Segunda parte, subir a Firebase | 48 |
| Vídeo 38. Provider (estado general del programa) | 50 |
| Vídeo 39. Navegación tabs superior | 53 |
| Vídeo 40. Badges | 54 |
| Vídeo 41. Panel desplegable (SlidingUpPanel). | 56 |
| Vídeo 42. Validar formularios en Flutter | 58 |

Del curso de Código Correcto, en youtube
<https://www.youtube.com/playlist?list=PLutrh4gv1YI8ap4JO23lN81JOSZJ3i5OO>

Vídeo 1. Primer paso del curso.

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

`void main() => runApp(const MiApp());` → cómo se define el main, llamando a mi aplicación

```
class MiApp extends StatelessWidget { → se genera una clase StatelessWidget que representa mi app  Escribir stless
  const MiApp({super.key});
  // este es el constructor.
  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      title: "Mi aplicación",
      home: Inicio(),
    );
  }
}
```

```
}  
} // fin de la clase mi app → fin de esa clase
```

`class` Inicio `extends` StatefulWidget { → se genera un statefulwidget Escribir **stful**

```
const Inicio({super.key});  
@override  
_InicioState createState() => _InicioState();  
} // fin de la clase inicio
```

`class` _InicioState `extends` State<Inicio> {

`@override`

Widget build(BuildContext context) {

`return` Scaffold(→ **importante, objeto Scaffold que mantiene la estructura completa de la aplicación.**

appBar: AppBar(→ **zona de la barra de la aplicación, parte superior.**

title: `const` Text("Mi primera aplicación"),

),

body: `const` Center(→ **widget center**

child: Text("Mi primer texto"), → **widget texto**

), // **Widget que se llama center**

);

}

} // fin de la clase inicio extends

Vídeo 2 y 3. Estructura de una App. Principales Widgets.

Trabajamos con el body

```
/* ===== */  
/* Empezamos trabajando en el body */  
/* ===== */
```

body: **ListView**(→ podemos poner containers **Column**, **Row**, etc.

children: [→ todos esos widget aceptan un conjunto de contenedores. ListView puede hacerse scroll, column no.

Container(→ contenedor

padding: **const** EdgeInsets.all(20.0), → espacios alrededor

child: Image.network("<https://www.seritium.es/.../56.32-e1683469980621.jpeg>"), → obtener imagen
) , // Container 2

],

),

Vídeo 4. Filas , Columnas y Botones.

Column → el contenido se distribuye arriba-abajo. Con mainAxisAlignment se le da el tamaño mínimo o hasta el final de la pantalla (max). Con crossAxisAlignment se apilan arriba, centrados, o con espacios entre ellos. crossAxisAlignment es la distribución horizontal.

body: Column(

mainAxisAlignment: MainAxisAlignment.spaceEvenly,

crossAxisAlignment: CrossAxisAlignment.end,

mainAxisSize: MainAxisSize.max,

children: <Widget>[

Container(→ importante para darle un ancho a la columna

width: MediaQuery.of(context).size.width/3, → así se consigue el ancho de la columna

height: 50,

color: **const** Color.fromARGB(161, 212, 183, 104),

child: **const** Text("Hola 01", textAlign: TextAlign.center),

),

const Text("Hola 02"),

// Texto

TextButton (// Raised button no funciona →

Tipo de botón. Los hay ElevatedButton y
OutlinedButton, y algunos más

```
// onPressed: null, // con null, no hace nada
child: const Row ( → distribución por filas dentro del botón
  mainAxisAlignment: MainAxisAlignment.min, //Mínimo para que tenga el mínimo tamaño posible
  children: [
    Icon(Icons.punch_clock), // Se puede ver un icono → Tres objetos. Un icono
    SizedBox( width: 7,), // Se parador con una caja con tamaño → Una caja separadora
    Text("Ver la hora"), // Se puede poner texto. → Por fin el texto del botón.
  ],
),
onPressed: () { → Método al presionar el botón.
  var t = DateTime.now(); // llama a la fecha y hora.
  print(t); // Imprime en la consola de depuración, no en la pantalla.
}, // función anónima, y puedo pulsarlo

), // Fin de textButton
], // Fin del array de Widgets
), // Fin del objeto column y del body.
```

Vídeo 5. Widgets separados.

Trabajamos con widgets separados

En body llamamos a cuerpo

```
body: cuerpo(),
```

Y la definición de cuerpo es

```
Widget cuerpo() { // recuerda que es una función. Abre llaves y cierra llaves
```

```
  return Container ( → esto es una función. Devolverá un objeto Container.
```

```
    decoration: const BoxDecoration( → forma de incluir una imagen en un contenedor
```

```
      image: DecorationImage( → usar un decorationimage
```

```
        image: NetworkImage("https://upload.wiki....La_noche_estrellada%22_de_Van_gogh.jpg"),
```

```
        fit: BoxFit.cover, // está dentro del DecorationImage
```

```
      ),
```

```
    ),
```

```
    child: Center( // Esto permite que se expanda.
```

```
      child: Column(
```

```
        mainAxisAlignment: MainAxisAlignment.center,
```

```
        children: <Widget>[
```

```
          nombre(), → llamo a otros widgets
```

```
          campousuario(),
```

```
          campocontrasena(),
```

```
          const SizedBox(height: 80), → separador
```

```
          botonentrar(),
```

```
        ],
```

```
      ), // fin del Column
```

```
    ), // fin del Center
```

```
  ); // fin del Container
```

```
} // Fin del cuerpo.
```

```
Widget nombre() { // función que devuelve el nombre de la aplicación
  return const Text(
    "SIGN IN",
    style: TextStyle( color: Color.fromARGB(255, 219, 201, 201), fontSize: 50.0), // Cambiar tamaño y color
  );
}
```

```
Widget campousuario() { → campos de texto.
  return Container(
    padding: const EdgeInsets.symmetric(horizontal: 20.0, vertical: 9),
    child: const TextField( → campo de texto
      decoration: InputDecoration(
        hintText: "Usuario",
        fillColor: Colors.white, filled: true,
      ), ), ); }
```

```
Widget campocontrasena() {
  return Container(
    padding: const EdgeInsets.symmetric(horizontal: 20.0, vertical: 9),
    child: const TextField(
      obscureText: true, → para enmascararlo con contraseña.
      decoration: InputDecoration(
        hintText: "Password",
        fillColor: Colors.white, filled: true,
      ), ), ); }
```

```
Widget botonentrar() { → botón textbutton y cambio de su estilo.
  return TextButton(
    style: TextButton.styleFrom( → obtener style con StyleFrom
      foregroundColor: Colors.blue,
      backgroundColor: Colors.cyanAccent,
      fixedSize: const Size(200, 50), ),
    onPressed: (){} , → al presionar, no hace nada. Pero necesito la función
    child: const Text("Boton entrar", style: TextStyle(fontSize: 20)),
  );
}
```

Vídeo 6. Navegación

Conecta dos ventanas main y pagina02

main.dart

import 'package:a_primer_programa/paginas/pagina02.dart'; → si existe esta página, este import se añade automáticamente

.....

```
ElevatedButton(
  onPressed: () { → al presionar el botón
    // print("Presionaste el botón");
    Navigator.push( → ponemos una página en el array MaterialPageRoute con Navigator
      context,
      MaterialPageRoute(builder: (context)=>const Pagina02()),
    );
  },
  child: const Text("Ir a la otra página"),
)
```

pagina02.dart

import 'package:flutter/material.dart';

class Pagina02 extends StatelessWidget { → sólo se necesita un stateless widget
const Pagina02({super.key});

@override

```
Widget build(BuildContext context) {
  return Scaffold( → y su scaffold
    appBar: AppBar(
      title: const Text("Términos y condiciones"),
    ),
    body: Padding(
      padding: const EdgeInsets.all(12.0),
      child: Column(
        children: [
          const Text("Mis términos y condiciones",
```



```
style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold, color: Colors.indigo)
),
```

```
const SizedBox(height: 15,),
```

..... → puede haber más contenido en medio.

```
SizedBox(
```

```
width: MediaQuery.of(context).size.width/1.5, → se puede cambiar el ancho de un container o sizedbox
```

```
child: ElevatedButton( → estilo para un elevated button
```

```
style: ElevatedButton.styleFrom(
  backgroundColor: Colors.black,
  foregroundColor: Colors.white,
  padding: const EdgeInsetsDirectional.all(15.0),
```

```
),
```

```
onPressed: (){
```

```
  // print("Volver");
```

```
  // Se podrían guardar los registros en una BD.
```

Navigator.pop(context); → al presionar vuelve a la página principal del contexto.

```
},
```

```
child: const Row( → el elevatedbutton consta de más de un hijo, icono y texto.
```

```
  mainAxisAlignment: MainAxisAlignment.center,
```

```
  children: [
```

```
    Text("Pulsa para volver", style: TextStyle(fontSize: 15,)),
```

```
    SizedBox(width: 10,),
```

```
    Icon(Icons.arrow_back_outlined),
```

```
  ],
```

```
),
```

```
), ), ], ), ), ); } }
```

Vídeo 7. Alert Dialog

Después del import, defino una variable booleana que servirá ¿¿para todo el programa?? No tengo muy claro ésto de las variables “globales”

```
bool _subscrito = false; → fuera de todas las clases
void main() => runApp(const MyApp());
...
```

Antes de cerrar la clase _InicioState , coloco la función mostrarAlerta

```
class _InicioState extends State<Inicio> {
  Widget build(BuildContext context) {
    ....
    ....
    ....
  } // fin del Widget
  /* ===== */
  /* funcion dentro de la clase */
  /* ===== */
  void _mostrarAlerta(BuildContext contexto) {
    ....
  } // fin de mostrar alerta
} // fin de la clase _InicioState
```

body

```
body: Center(
  child: Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      ElevatedButton( → la columna tiene un botón, que llamará a la función mostrar alerta...
        onPressed: (){
          _mostrarAlerta(context); → a la función mostrar alerta se le pasará el contexto
        },
```

```

style: ElevatedButton.styleFrom(
  foregroundColor: const Color.fromARGB(158, 237, 221, 221),
  backgroundColor: Colors.red,
),
child: Text( → el texto del botón depende de lo que valga la variable suscrito.
  _suscrito ? "¿Quieres dejar de pertenecer al canal?" : "¿Quieres inscribirte en el canal?",
  style: const TextStyle(fontSize: 16.0),
),
const SizedBox(height: 100,),
Text( → el texto de la etiqueta informativa tb dependerá de lo que valga esa variable.
  _suscrito ? "Subscrito" : "No suscrito",
  style: const TextStyle(fontSize: 20),
)
],
),
), // Widget que se llama center

```

mostrarAlerta

void _mostrarAlerta(BuildContext contexto) { → se define como void, sin retornar nada y se le pasa el contexto

showDialog(→ ventana de diálogo

// builder: (contexto) => **const** AlertDialog(

barrierDismissible: **false**, // No se puede quitar la alerta → alerta no desaparece al hacer clic fuera.

builder: (_) => **AlertDialog**(→ el diálogo es de tipo alerta.

title: **const** Text("Suscribirse"),

content: **Text**(→ texto que cambia según suscrito.

_suscrito ? "¿Estás seguro que quieres desuscribirte?" : "¿Estás seguro de suscribirte?"

),

actions: [→ botones de acción. Más de uno, es un array.

ElevatedButton(

onPressed: () {

print("Si");

setState() { → función que recarga el estado del Widget build BuildContent context

// Para ello la función tiene que estar dentro de la clase Inicio extends.

```

// Ver la última llave al final.
    _subscrito = !_subscrito; → recarga el estado, y cambia el valor de la variable.
  });
  Navigator.pop(contexto); → la alerta es una ventana más, vuelve a la pantalla inicial
},
child: const Text("Sí, seguro"),
),
ElevatedButton(
  onPressed: () { → Cancelar es no hacer nada.
    print("No");
    Navigator.pop(contexto); //vuelve a la pantalla
  },
  child: const Text("No, cancela")
),
],
), // fin de AlertDialog
context: contexto,
); // fin de ShowDialog, que es una función
} // fin de mostrar alerta

```

Vídeo 8. Consumiendo una API REST >> clases

| Código de la clase | Llamada |
|---|--|
| <pre> class Empresa { // definición de una clase } </pre> | <pre> class _InicioState extends State<Inicio> { Empresa _facebook = new Empresa(); // creo una variable local, una instancia, de esa clase @override Widget build(BuildContext context) { </pre> |

Utiliza además este código. Supongo que para recargar cada vez que se inicia ¿¿??

@override → el método initState ya está en la clase State. Lo que hace este método es sobrescribirlo.

```

void initState() { super.initState(); print(_facebook); }

```

Queda así

| Código de la clase | Llamada |
|--|---|
| <pre>class Empresa { // definición de una clase late String nombre; late String propietario; late int ingresoAnual; // constructor Empresa(String nm, String pr, int ing) { nombre = nm; // inicializa nombre propietario = pr; // inicializa propietario ingresoAnual = ing; // inicializa ingreso Anual } }</pre> | <pre>final Empresa _facebook = new Empresa("Facebook","Mack Zukerberg",1000); // creo una variable local de esa clase pasando los parámetros al constructor.</pre> |

Usando la instancia

```
child: Text(
  "${_facebook.nombre}- ${_facebook.propietario} - ${_facebook.ingresoAnual}",
  style: const TextStyle( fontSize: 14.0, color: Colors.blue),
  textAlign: TextAlign.justify,
```

Se concatena de esa manera los datos.

Vídeo 9. Consumiendo API REST >> Listas

Primero, creamos una clase. Al final del programa, fuera de todo Widget

```
/* Clases */
class Persona { → almacena datos de nombre, apellidos y teléfono de una persona
    late String nm;
    late String lastname;
    late String phone;

    Persona(nombre,apellidos,tfno) { → constructor
        nm = nombre;
        lastname = apellidos;
        phone = tfno;
    }
}
```

Segundo, creamos un array (se llaman List) de objetos de la clase Persona.

```
class _MyWidgetState extends State<MyWidget> { → dentro de la clase y antes del build
BuildContext

    final List<Persona> _personas=[ → el array o List se llama _personas
        Persona("Luis","Suárez","003562589"),
        Persona("Manuel","Laínez","0032443"),
        Persona("Alberto","García","00311131"),
        Persona("Ana","González","14324324"),
        Persona("Mariela","Gómez","435435345"),
        Persona("Humberto","Rodríguez","9987667"),
    ];

    @override
    Widget build(BuildContext context) {
        return MaterialApp(
        .....
    }
```

Tercero, dentro del body creamos un objeto ListView con builder.

```
body:ListView.builder( → Widget ListView
  itemCount: _personas.length, → necesita saber cuántos items construirá de antemano.
  itemBuilder: (context,index) { → construye cada ítem de la lista
    // return Text(_personas[index].nm);
    return ListTile( → Cada ítem será un TILE
      title: Text("${_personas[index].nm} ${_personas[index].lastname}"), → contiene título (texto principal)
      subtitle: Text(_personas[index].phone), → subtítulos
      leading: CircleAvatar( → circulito al inicio, inicial.
        child: Text(_personas[index].nm.substring(0,1)),
      ),
      trailing: const Icon(Icons.arrow_forward_ios_rounded), → icono de flecha al final
      onPressed: () { → Al hacer una pulsación larga llama a la función _borrarpersona
        // onPressed: () { // pulsación corta
          _borrarpersona(context,_personas[index]); → pasamos el contexto y un elemento del array
        },
      ); // ListTile
    }, // Fin del itemBuilder
  ), // ListView builder
```

Cuarto, por fin la función _borrarpersona

```
Widget build(BuildContext context) {
  ....
  ....
  ....

  _borrarpersona (context, Persona persona) {
    showDialog( → abro un diálogo, que necesita...
      context: context, → el contexto
      builder: () => AlertDialog( → construyo una alerta en ese contexto ( _ )
        title: const Text("Borrado"), → con un título
        content: Text("${_¿Quieres borrar a ${persona.nm}} de la lista?"), → Y un contenido dinámico
        actions: [
          TextButton(onPressed: () { → acción de cancelar. Sólo cierra la alerta.
            Navigator.pop(context);
          },
          child: const Text("Cancelar"),
        ),
        TextButton(onPressed: () { → acción de borrar
          print("${persona.nm} ${persona.lastname}"); → muestra en la consola quien borrará
```

```

setState() { → IMPORTANTE. Lo que sigue se necesita que renueve el estado, refresque la pantalla.
  _personas.remove(persona); → quitar el objeto del array o lista
});
Navigator.pop(context); → Quitar la alerta. Navegar entre pantallas.
},
child: const Text("Borrar", style: TextStyle(color: Colors.red)),
),
],

), // AlertDialog
); // ShowDialog
}

} // Fin del widget principal build(BuildContext context)

```

Vídeo 10 y 11. API REST >> componente http

Traer información desde APIs de Internet.

Instalación

- Visitar página <https://pub.dev/packages/http/install>
- Utilizo el método “dart pub add http”. Esto instala la dependencia “http: ^1.2.1” en el fichero pubspec.yaml. Desde el terminal.
- Instala los paquetes necesarios. He tenido que actualizar usando “dart pub outdated” y “dart pub upgrade --major-versions”.

Registro en GIPHY development

- Creo una api llamada “mi_ejemplo” con clave tNZCn64z2rLXBWv0oAFw1DeSdt2EeU4f
- Obtengo una dirección URI con el acceso a datos JSON. Esa dirección hay que copiarla.

Crear una estructura que almacenará los datos

- En una carpeta aparte “models”, dentro de “libs”, fichero **Gif.dart**. Creo la clase **Gif**

```

class Gif { → al llamarla dentro de main.dart debe crearse la línea “import 'package:j_api_http/models/Gif.dart';”
  late String name;
  late String url;
  Gif(nombre,direccion) {
    name = nombre;
    url=direccion;
  }
}

```


Crear un objeto Gif, con la condición Future

class _MyAppState extends State<MyApp> { → **empezamos dentro de la clase State**

late Future<List<Gif>> _listadoGifs; → **Crear una variable tipo lista de instancias de la clase Gif**

→ **IMPORTANTE: condición Future. No podrá hacerlo hasta no recibir cierta información de internet, no podrá hacerlo inmediatamente.**

Future<List<Gif>> _getGifs() async { → método *asíncrono* que aplicamos a las listas tipo Gif

final response = await

http.get(Uri.parse("https://api.giphy.com/v1/gifs/trending?api_key=tNZCn64z2rLXBWv0oAFw1DeSdt2EeU4f&limit=10&offset=0&rating=g&bundle=messaging_non_clips")); → **la respuesta viene desde nuestra Uri. Se almacena en response y tenemos que precederla de await, o sea, esperar hasta que nos llegue de internet.**

List<Gif> gifs = []; → **creamos lista vacía de la clase Gif, que es lo que tiene que retornar. Importante porque tiene que retornar algo.**

if (response.statusCode==200) { → **código 200, si hay respuesta**

String body = utf8.decode(response.bodyBytes); → **asegurarme que todo está en utf8**

final jsonData = jsonDecode(body); → **convierte información en json válido**

// print(jsonData["data"][0]["images"]); // devuelve el objeto data del json, elemento 0

for (var item in jsonData["data"]) { → **por cada valor en cada data de json**

gifs.add(→ **añado a la lista gifs un objeto tipo Gif, con título y url del gif**

Gif(item["title"], item["images"]["fixed_height"]["url"])

);

}

} else { → **y si no, que arroje una excepción**

throw Exception("Falló la excepción");

}

return gifs; → **importante, debe retornar algo, aunque sea la lista vacía**

}

@override → **superponer un método de clase que ya existe**

void initState() { → **se ejecuta al inicializar la clase. Hay que recargarla desde inicio.**

super.initState(); → **ejecuta el método que ya existe de initState**

_listadoGifs = _getGifs(); → **añade la definición de lista _listadoGifs a lo que me retorna.**

print(_listadoGifs); // Me quedo aquí. Consigo descargar de internet un montón de información de imágenes.

}

.....

Estructura del body. El widget FutureBuilder

body: FutureBuilder(→ **objeto FutureBuilder, que se construirá cuando reciba los datos..**

future: _listadoGifs, → **lista anterior de Gifs. La que se recibe en “un futuro”**

builder: (context, snapshot) { → **¿Cómo se construye? en un contexto, con unos datos que recibe “snapshot”.**

String devolver = "";

if(snapshot.hasData) { → **si tengo datos**

// print(snapshot.data);

// devolver = "Hola, con datos";

// return ListView(→ **para probar un list view**

return GridView.count(→ **widget definitivo. En formato filas y columnas.**

crossAxisCount: 2,

children: **_misGifs(snapshot.data),** → **los obtengo de una función**

);

} else if (snapshot.hasError) { → **si tiene errores.**

print(snapshot.error);

devolver = "Hola, sin datos";

return Text(devolver);

}

return const Center(→ **centrado, mientras no carga los datos, sale un símbolo de “cargando”.**

child: CircularProgressIndicator(

color: Color.fromARGB(255, 91, 80, 162),

),

); // Fin del center

}, // opción builder

), // Fin del FutureBuilder

Por fin, llamada a la función ***_misGifs***

List<Widget> _misGifs(List<Gif>? data) { → **retorna una lista de widgets y acepta una lista de objetos de la clase Gif que podrían ser nulos. NOTA: esto hay que estudiarlo.**

List<Widget> gifs = []; → **creo una lista de gifs**

for (var i in data!) { → **por cada dato de los recibidos, si no son nulos.**

```
  gifs.add( → Imágenes que se ajustan a cuadros expandidos. Eso es lo que devuelve. Una por cada imagen recuperada de internet
    Card(
      child: Column(
        children: [      Expanded(child: Image.network(i.url, fit:
                          BoxFit.fill,)),
                          // Image.network(i.url),
                          /* Padding(
                          padding: const EdgeInsets.all(8.0),
                          child: Text(i.name),
                          ),*/
                        ],
      )) // fin del Card y de Column
    ); // fin de gifs add
  } // fin del for
  return gifs;
}
```

Vídeo 12. Barra de navegación inferior

Quitar etiqueta de desarrollo

```
return MaterialApp(
  debugShowCheckedModeBanner: false, // para que me quite la etiqueta de desarrollo
  ....
```

Parte del Scaffold, el bottomNavigationBar,

Justo antes de "Widget build(BuildContext context) {" se declara la variable **int _paginaActual = 0;**

```

home: Scaffold(
  appBar: ....
  body: ....
  bottomNavigationBar: BottomNavigationBar
    → apartado especial del Material app, con un widget especial BottomNavigationBar
  onTap: (index){ → al pulsar sobre la pantalla
    setState() { → Necesita recargar el estado
      _paginaActual = index; → Señala el botón que he pulsado
    });
  },
  currentIndex: _paginaActual, → Índice por defecto
  items: const [ → que recibe elementos items y da error si no pongo al menos dos, y si no le pongo labels.
    BottomNavigationBarItem(icon: Icon(Icons.home), label:"casa"),
    → que son elementos de la clase BottomNavigationBarItem
    BottomNavigationBarItem(icon: Icon(Icons.supervised_user_circle_outlined), label:
"coche"),
    ],),

  ), → Fin del Scaffold

```

Dos contenidos distintos, del tipo statelesswidget

Aunque están en el mismo fichero, deberían ir en ficheros distintos.

```

class PaginaHome extends StatelessWidget {
  const PaginaHome({super.key});
  @override
  Widget build(BuildContext context) {
    return const Center(
      child: Text("Mi página home", style: TextStyle(fontSize: 20.0, color: Colors.indigo)),
    );
  }
}

```

```

class PaginaUsers extends StatelessWidget {
  const PaginaUsers({super.key});
  @override
  Widget build(BuildContext context) {
    return const Center(
      child: Text("Mi página users", style: TextStyle(fontSize: 20.0, color: Color.fromARGB(255,
14, 99, 23))),
    );
  }
}

```

Selección del contenido

A) Si hay sólo dos widgets

En el body del SCAFFOLD

```
body: _paginaActual==0 ? const PaginaHome() : const PaginaUsers(),
```

B) Si hay más de dos widgets

Creamos un array o lista de widgets

```
class _MyAppState extends State<MyApp> {  
  
  int _paginaActual = 0;  
  
  final List<Widget> _mispaginas =[  
    const PaginaHome(),  
    const PaginaUsers(),  
  ];  
  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      ....  
    );  
  }  
}
```

Y en el body llamamos a esa lista con la variable `_paginaActual`

```
body: _mispaginas[_paginaActual],
```

NOTA: lo mejor es crear dentro del “lib” una carpeta llamada “pages”, y dentro meter un fichero .dart por cada widget. Todo por separado. Así queda más claro. En el fichero principal debe llamarse con import a ambas páginas dart.

Vídeo 13. Animar contenedores

Por defecto, los contenedores *no tienen ni ancho ni alto, y son transparentes*. Si acaso tienen el tamaño de los hijos que contiene.

Partimos de un dibujo de un container

```
body: Center(
  child: Container (
    width: 100.0,
    height: 100.0,
    // color: Colors.orange, --> No se puede poner con decoration
    decoration: BoxDecoration(
      borderRadius: BorderRadius.circular(10.0),
      color: Colors.orange,
    ),
  ),
),
```

Pero creamos variables para poder hacer una animación al pulsar un botón. Y una función que se llamará al apretar un botón

```
class _MyAppState extends State<MyApp> {

  double _width = 100.0; → declarar variables
  double _height = 100.0;
  Color _color = Colors.blue;
  BorderRadius _borderRadius = BorderRadius.circular(10.0);

  void _cambiarContainer(){ → llamamos a la función random. Debe importar paquete matemático.
    final random = Random();
    _width = random.nextDouble()*250.0+100.0; // entre 100 y 350
    _height = random.nextDouble()*250.0+100.0;
    _color = Color.fromRGBO( → colores por RGB
      random.nextInt(256),
      random.nextInt(256),
      random.nextInt(256),
      1); // colores opacos
    // random.nextDouble());
    _borderRadius = BorderRadius.circular(random.nextInt(15)+3);
    setState(() {}); → importante, hay que cambiar el estado del widget
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Material App',
      home: Scaffold(
```

```

appBar: AppBar(
  title: const Text('Contenedor Animado'),
),
body: Column(
  // mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  // mainAxisAlignment: MainAxisAlignment.max,
  children: [
    Expanded( → el elemento Center y el Container se expanden todo lo que pueden a lo largo de la columna.
    child: Center(
      child: AnimatedContainer ( → Container que se anima.
        width: _width, → ancho
        height: _height, → alto
        // color: Colors.orange, --> No se puede poner con decoration
        decoration: BoxDecoration(
          borderRadius: _borderRadius, → tipo de radio de borde
          color: _color, → ancho
        ),
        duration: const Duration(milliseconds: 500,), → lo que dura la animación
        curve: Curves.elasticIn, → tipo de efecto. Muy chulo.
      ),
    ),
    ),
    ),
    ElevatedButton( → botón que llama a la función y realiza el cambio.
      onPressed: _cambiarContainer,
      child: const Text("Cambiar Container")),
    ],
  ),
);
}
}

```

Si envuelvo el AnimatedContainer en un InkWell, puedo asignarle la acción onTap. Al pulsar sobre el cuadro, se cambia.

Vídeo 14. Carrusel de imágenes o swiper.

Imágenes de izquierda a derecha.

Descargamos un complemento externo llamado flutter_swiper de https://pub.dev/packages/flutter_swiper/versions

Hay que seguir las indicaciones de esta página, para instalar carousel_slider, que es el complemento actualizado: https://pub.dev/packages/carousel_slider/install , y seguir un poco las indicaciones para actualizar dependencias.

En el pubspec.yaml, en **dependencies**, aparece **carousel_slider: ^4.2.1**

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

```
void main() => runApp(MyApp());
```

```
class MyApp extends StatelessWidget {
  MyApp({super.key});
```

```
    final List<String> imgList = [ → imágenes cargadas de internet
```

```
'https://images.unsplash.com/photo-1520342868574-5fa3804e551c?ixlib=rb-0.3.5&ixid=eyJhcnBfaWQiOjE5MDd9&s=6ff92caffcdd63681a35134a6770ed3b&auto=format&fit=crop&w=1951&q=80',
'https://images.unsplash.com/photo-1522205408450-add114ad53fe?ixlib=rb-0.3.5&ixid=eyJhcnBfaWQiOjE5MDd9&s=368f45b0888aeb0b7b08e3a1084d3ede&auto=format&fit=crop&w=1950&q=80',
'https://images.unsplash.com/photo-1519125323398-675f0ddb6308?ixlib=rb-0.3.5&ixid=eyJhcnBfaWQiOjE5MDd9&s=94a1e718d89ca60a6337a6008341ca50&auto=format&fit=crop&w=1950&q=80',
'https://images.unsplash.com/photo-1523205771623-e0faa4d2813d?ixlib=rb-0.3.5&ixid=eyJhcnBfaWQiOjE5MDd9&s=89719a0d55dd05e2deae4120227e6efc&auto=format&fit=crop&w=1953&q=80',
'https://images.unsplash.com/photo-1508704019882-f9cf40e475b4?ixlib=rb-0.3.5&ixid=eyJhcnBfaWQiOjE5MDd9&s=8c6e5e3aba713b17aa1fe71ab4f0ae5b&auto=format&fit=crop&w=1352&q=80',
'https://images.unsplash.com/photo-1519985176271-adb1088fa94c?ixlib=rb-0.3.5&ixid=eyJhcnBfaWQiOjE5MDd9&s=a0c8d632e977f94e5d312d9893258f59&auto=format&fit=crop&w=1355&q=80'
];
```

```
@override
```

```
Widget build(BuildContext context) {
```

```
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Material App',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Mi carrusel'),
        ),
        body: Center(
          child: Container(
            child: _swiper(), → carrusel dentro de un contenedor
          ),
        ),
      ),
    );
```



```

    ),
    ),
    );
}

```

```

Widget _swiper(){
  return CarouselSlider(
    items: _misWidgets(), → obtiene la lista de una función
    options: CarouselOptions(
      aspectRatio: 16/9,
      // height: 800,
      viewportFraction: 0.9, // fracción del Carrusel que ocupa la pantalla?
      initialPage: 0, // Qué contenedor es el que inicia el carrousel. 0, 1, 2...
      enableInfiniteScroll: true, // loop infinito. Del 3er pasa al 1er.
      reverse: false, // si es true, orden de los items al revés
      autoPlay: true,
      autoPlayInterval: const Duration(seconds: 2),
      autoPlayAnimationDuration: const Duration(milliseconds: 500),
      autoPlayCurve: Curves.linear,
      enlargeCenterPage: true, // si true, pone más grande el central y escondidos los otros.
      scrollDirection: Axis.horizontal, // eje en el que se cambian o rotan
    ),
  );
}

```

List<Widget> _misWidgets() { → **función que crea la lista**

```

List<Widget> lista = [];
for (var cada in imgList) {
  lista.add (
    Card(
      child: Expanded(
        child: Image.network(
          cada,
          fit: BoxFit.fill,),
      ),
    ),
  );
}
return lista;
}
}

```

Vídeo 15. Persistencia de datos

Hay que instalar el paquete de datos https://pub.dev/packages/shared_preferences , que va por la versión 2.2.3

Desde el terminal instalar con **flutter pub add shared_preferences** e ir resolviendo los problemas de referencias siguiendo las instrucciones.

Importante: para ver que funciona hacer un HARD RESET. Al inicio.

```
import 'package:flutter/material.dart';
import 'package:shared_preferences/shared_preferences.dart'; → importar paquete de
shared_preferences
```

```
void main() => runApp(const MyApp());
```

```
class MyApp extends StatefulWidget {
  const MyApp({super.key});
  @override
  State<MyApp> createState() => _MyAppState();
}
```

```
class _MyAppState extends State<MyApp> {
```

```
  int value = 0; → valor que irá cambiando. Se inicia en cero.
```

```
  _changeValue() async { → función asíncrona
```

```
    SharedPreferences preferencias = await SharedPreferences.getInstance();
```

```
    → obtener instancia de SharedPreferences con await.
```

```
    setState(() { → cambia el estado, necesito añadir uno a value. Y guardarlo en las preferencias.
```

```
      value ++;
```

```
      preferencias.setInt("valor",value);
```

```
    });
```

```
  }
```

```
  _cargarPreferencias() async { → función asíncrona
```

```
    SharedPreferences preferencias = await SharedPreferences.getInstance(); → obtiene instancia
```

```
    setState(() {
```

```
      value = preferencias.getInt("valor") ?? 0; → obtiene el valor y si no, es CERO.
```

```
    });
```

```
  }
```

```

@override
void initState() { → initState no puede ser asíncrono, pero sí llamar a una función asíncrona _cargarPreferencias()
    // TODO: implement initState
    super.initState();
    _cargarPreferencias();
}

@override
Widget build(BuildContext context) {
    return MaterialApp(
        debugShowCheckedModeBanner: false,
        title: 'Preferencias',
        home: Scaffold(
            appBar: AppBar(
                title: const Text('Preferencias'),
            ),
            body: Center(
                child: Column(
                    mainAxisAlignment: MainAxisAlignment.center,
                    children: [
                        Text(
                            value.toString(),
                            style: const TextStyle(fontSize: 80.0, color: Colors.purple)),
                        TextButton(
                            onPressed: _changeValue, → cambiar valor y guardarlo.
                            child: const Text("Aumentar Valor"))
                    ],
                ),
            ),
        );
}
}

```

Vídeo 16. Pasar una información de un widget a otro (página a otra).

Página principal

```
import 'package:flutter/material.dart';
import 'package:o_traspasar_informacion/pages/sfull.dart';
import 'package:o_traspasar_informacion/pages/sless.dart';
```

```
void main() => runApp(const MyApp());
```

```
class MyApp extends StatelessWidget {
  const MyApp({super.key});
```

```
  @override
```

```
  Widget build(BuildContext context) {
    return const MaterialApp(
      title: 'Material App',
      home: Home(), → extiende a Home
    );
```

```
  }
}
```

```
class Home extends StatelessWidget { → página extendida a Home
```

```
  const Home({
    super.key,
  });
```

```
  @override
```

```
  Widget build(BuildContext context) {
```

```
    TextEditingController miTexto = TextEditingController(text: "");
```

→ **mitexto es una variable del tipo TextEditingController, capaz de almacenar un valor de un campo de texto.**

```
    return Scaffold(
      appBar: AppBar(
        title: const Text('Página Principal'),
      ),
      body: Column(
        children: [
          Container(
```

```

padding: const EdgeInsets.all(20.0),
child: TextField(
  controller: miTexto, → campo de texto, variable que puede traspasar páginas
  style: const TextStyle(color: Colors.white,) ,
  decoration: const InputDecoration( → decoración del TextField
    /* focusColor: Colors.white10, */
    fillColor: Color.fromARGB(255, 108, 106, 106),
    filled: true,
    border: InputBorder.none,
    hintText: "Información a mandar",
  ),
),
),
ElevatedButton(
  onPressed: (){
    Navigator.push(context,
      MaterialPageRoute(builder: (context)=> PStless(miTexto.text)),); → se le pasa el String
      → Pasar a una página stateless
    /* print(miTexto.text); */
  },
  child: const Text("Enviar a Stateless"),
),
const SizedBox(height: 20.0,),
ElevatedButton(
  onPressed: (){
    Navigator.push(context, → idéntico al pasarlo a un stateful
      MaterialPageRoute(builder: (context)=> PStfull(miTexto.text)),);
  },
  child: const Text("Enviar a Statefull"),
),
],
),
);
}
}

```

Página de tipo stateless

```
import 'package:flutter/material.dart';
class PStless extends StatelessWidget {
  final String texto; → variable de texto final que recibe la página stateless
  const PStless(this.texto, {super.key}); → constructor con la variable.
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text("Stateless Página"),
      ),
      body: Center(
        child: Text(texto), → uso de la variable
      ),
    );
  }
}
```

Página de tipo statefull

```
import 'package:flutter/material.dart';
class PStfull extends StatefulWidget {
  final String texto; → variable de texto final que recibe la página statefull
  const PStfull(this.texto, {super.key}); → constructor con la variable.
  @override
  State<PStfull> createState() => __PStfullState();
}
class __PStfullState extends State<PStfull> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Statefull Página',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Página StateFull'),
        ),
        body: Center(
          child: Text(widget.texto), → ¡¡Ojo!!, cambia la llamada al objeto.
        ),
      ),
    );
  }
}
```

Vídeo 18. Firebase

Conexión FIREBASE - Flutter

<https://www.youtube.com/watch?v=nqAlz7Pkbp0>

- **Crear un materialapp desde cero.**
 - Cambiaremos la id del build.gradle y resto del documento
- **Instalar el paquete change_app_package_name**

Instalo desde la terminal con:

 - flutter pub add -d change_app_package_name (con -d se instala en dependencias de desarrollo)
 - Tengo que hacer un flutter pub outdated
 - flutter pub get y flutter pub outdated para actualizar dependencias
 - dart run change_app_package_name:main **com.agrgal.testfirebase**
 - Y así le cambio el nombre a la aplicación. Ver más en [change_name](#)

- **Comprobar**

Comprobar que en android>>app>>src>>kotlin>>MainActivity.kt se ha cambiado el nombre, por ejemplo. La versión mínima es flutter.minSdkVersion he averiguado que es al 21, que la pone él manualmente. No lo cambio porque es esa. 21 se necesita para FireBase y para Messages la 19.

En el proyecto de FireBase le doy a

- Agregar tu proyecto a la App . Botón FLUTTER.
- Instalar FireBase CLI, según las instrucciones de <https://firebase.google.com/docs/cli?hl=es&authuser=1#mac-linux-auto-script>
 - curl -sL <https://firebase.tools> | bash
- Hacer **firebase login** desde el terminal. Se abre una página en el navegador y te autentificas con la cuenta de google. Puedes hacer **firebase projects:list** para comprobar y ver tus proyectos.
- **Instalar firebase_core**
 - Primero hacer dart pub global activate flutterfire_cli
 - Variable export --> export PATH="\$PATH":"\$HOME/.pub-cache/bin"
 - Hacemos un "flutterfire configure" (dentro de la carpeta del proyecto y siempre que se cambie el ID)
 - Elegimos el proyecto (flechas arriba y abajo)
 - Seleccionamos y deseccionamos plataformas (espacio, y flechas).
 - ¡¡Dios !! ALT+Z en el terminal para poder alargar el ancho de la ventana del terminal...
 - Escribo el nombre que seleccioné antes para el ID: **com.agrgal.testfirebase**
- **Crear el fichero firebase_options**

- Instalamos **flutter pub add firebase_core** y actualizamos con flutter pub outdated
- Se puede comprobar que se quitan los errores.
- Da un error al hacer la implementación en el móvil ANDROID. Dentro de la carpeta android buscar **settings.gradle** y cambiar la versión del kotlin a 1.8.0 id **"org.jetbrains.kotlin.android" version "1.8.0" apply false**

• Creando Firestore Database en FireBase

https://youtu.be/3IJI0tF6Fg?si=N6i2P8_E0bV9hRHU

Creo una base de datos en Firestore Database

- La ID la hace automáticamente. Escojo un servidor multirregional (euro3 en mi caso).
- Voy a empezar en modo de pruebas. Tiene un tiempo limitado, en el que cualquiera puede modificar los datos. El otro modo, modo de producción, solo tú tienes acceso a los datos y a quienes se registren con el modo de registro que indiques. Lo que sí es que le daré un año, a esas pruebas.
- No es una base de datos relacional. Se trabaja con colecciones.
 - Creo una colección llamado **usuarios**.
 - Dentro de esta colección creo un campo llamado email y otro llamado password. Ambos en un **documento** con un ID

• Instalar la librería cloud_firestore

- flutter pub add cloud_firestore y después actualizar a flutter pub outdated

• Código para usar la base de datos

Dentro de void main. La primera línea asegura la actualización de dependencias y la segunda que se inicia Firebase

```
void main(){
  WidgetsFlutterBinding.ensureInitialized();
  Firebase.initializeApp();
  runApp(const MyApp());
}
```

•

Y dentro del botón he puesto, que cuando se presione, se añada a la colección un dato

```
FirebaseFirestore.instance.collection("usuarios").add({
  'email':'micorreo@otro.com',
  'password':'12312312 21312 12 '
});
```


Vídeo 21. Splashscreen (android)

- Navegamos hasta la carpeta android >> app >> src >> main >> res y aquí hay dos carpetas drawable y drawable-v21 con exactamente el mismo fichero, launch_background.xml. Las modificaciones se hacen en el mismo fichero.
- En la raíz, **res**, en la subcarpeta **values**. Voy a crear un fichero **color.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="primary">#4455F3</color>
</resources>
```

1. En la carpeta drawable, creo un fichero llamado **bg_color.xml**
 - a. `<?xml version="1.0" encoding="utf-8"?>`
 - b. `<shape xmlns:android="http://schemas.android.com/apk/res/android">`
 - i. `<solid android:color="@color/primary"/>`
 - c. `</shape>`
 2. En el fichero launch_background.xml cambio el item **<item android:drawable="@android:color/white" />** por **<item android:drawable="@drawable/bg_color" />**
- **Y repetimos el proceso 1 y 2 en la carpeta drawable-v21**

Para insertar una imagen

- Primero, en la carpeta **mipmap** pongo una imagen con transparencia de tipo png. En todas y cada una de las carpetas mipmap.
- Segundo, en el fichero launch_background.xml activo un item tipo bitmap. Cuidado, en los dos launch_background.xml

```
<item>
    <bitmap
        android:gravity="center"
        android:src="@mipmap/joda" /> → ¡Ojo! no se le pone png
</item>
```

Vídeo 22. Splashscreen con IOS → No lo hago

Vídeo 23. Cambiar el logo de tu APP (Android) y 24. Como un pro

Cambiar icono de forma manual

- Navegamos hasta la carpeta android >> app >> src >> main >> res y en cada carpeta midmap está el icono ic_launcher.png
- Pues es tan sencillo como copiar aquí mi icono, y renombrarlo a ic_launcher.png . Hay que hacerlo en las cinco carpetas midmap.
- El problema de cambiarlo así es que hay que respetar los tamaños.
- Esto **NO LO HAGO**.

Cambiar icono con un paquete

- Es mejor hacerlo de esta forma.
- En la carpeta de proyecto, creo una carpeta llamada **assets** y una subcarpeta llamada **icon**. Y en esa carpeta copio mi icono en formato png. Mejor la renombro y le pongo icon.png
- En **pubspec.yaml**, en el apartado **dev_dependencies** copio **flutter_launcher_icons: ^0.13.1** (ver web https://pub.dev/packages/flutter_launcher_icons) ¿o ver cómo se instala?
- En el mismo fichero **pubspec.yaml** voy a pegar, al final, indentado a la izquierda,

```
flutter_launcher_icons:  
  android: "launcher_icon"  
  # ios: true → lo comento porque no me hace falta.  
  image_path: "assets/icon/icon.png"  
  min_sdk_android: 21 # android min sdk min:16, default 21
```
- Observar que en el ejemplo de la web, también viene como ponerlo para aplicaciones linux, web, etc.
- Hacemos **flutter pub get**, y lo que implica de actualización.
- Creamos los iconos con **flutter pub run flutter_launcher_icons:main** (me da un aviso que mejor **dart run...** , pero lo instala).

Cambiar icono de forma profesional

- Prepara el icono. Él lo hace 500x500 con formato png transparente.
- Entramos en la web <https://romannurik.github.io/AndroidAssetStudio/index.html> y en el apartado **launcher icon generator**. Pero... me manda a esta nueva página: <https://icon.kitchen>. En imagen, introducimos nuestra imagen. En este caso mi JODA.
- Descargamos la carpeta que nos genera. Una es de Android.
- En la ruta android >> app >> src >> main >> res, borramos todas las carpetas midmap y copiamos las carpetas generadas con la aplicación.
- Y ejecutamos la aplicación... ¡¡YA!!
- Supongo que algo parecido se hará con IOS , web, etc.

Video 25. Drawer

- Es un componente del Scaffold: drawer: Drawer();

```
home: Scaffold(  
  drawer: Drawer( → Dentro del Scaffold añado un objeto Drawer  
    // width: 400,  
    child: Container( → Y ya los contenedores  
      color: Color.fromARGB(255, 205, 205, 161),  
      child: Column(  
        children: [  
          Container(  
            width: 100,  
            height: 60,  
            margin: const EdgeInsets.only(top: 50.0,bottom: 20.0,),  
            child:  
Image.network("https://e7.pngegg.com/pngimages/959/75/png-clipart-blue-flat-car-car-fl  
at-car-thumbnail.png"),  
          ),
```

- double.infinity → **tomar todo el ancho disponible.**

Vídeo 26 y 27. TextFormField personalizados

- Nuestro formulario form se hace en página aparte, y se llama en el body de main.dart

Un campo sería...

```
// Método _inputName
Container _inputName() { → lo extraemos, y le ponemos un nombre
  return Container(
    decoration: BoxDecoration( → decoración de la caja
      borderRadius: BorderRadius.circular(5),
      // color: const Color.fromARGB(255, 139, 147, 187),
      border: Border.all(color: Colors.grey),
    ),
    padding: const EdgeInsets.symmetric(horizontal: 15.0),
    margin: const EdgeInsets.symmetric(horizontal: 15.0),
    child: TextFormField(
      style: const TextStyle(fontSize: 20.0),
      decoration: const InputDecoration( → decoración del campo en sí
        border: InputBorder.none,
      ),
    ),
  );
}
```

- Se repite cuatro veces, cada una con un nombre.
- Nuestro form contiene un Row. ¡Cuidado! Los containers internos a este Row deben tener el ancho definido. Con **width: MediaQuery.of(context).size.width*0.5** o bien **width: MediaQuery.sizeOf(context).width*0.5**.
- Para que no haya overflow vertical, el objeto Column puede wrapearse dentro de otro llamado **SingleChildScrollView**.
- Para que un TextFormField tenga formato numérico, introducir el modificador **keyboardType: TextInputType.number**.
- Buscar en internet e instalar **mask_text_input_formatter** (flutter pub add mask_text_input_formatter)
- Ahora puedo establecer una variable con la máscara para la tarjeta

```
var maskCode = MaskTextInputFormatter(
  mask: "####-####-####-####",
  filter: {"#":RegExp(r'[0-9]')}
);
```

- Y llamarla dentro del textField con **inputFormatters: [maskCode]**. Ojo, las llaves.

- Queda pendiente la cuestión de validar un valor. Por ejemplo, la fecha del formato "mm/yy" ¿Cómo se hace?

=====

- Se puede hacer poniendo una variable global que compruebe si se está autovalidando

bool _autovalidate = false;

- Y después poner esto en el TextFormField (acompañado por cambio de color por variable en el TextStyle).

```
autovalidateMode: AutovalidateMode.onUserInteraction, → método de autovalidación
onChanged: (value){
  setState(() {
    _autovalidate = true; → variable global autovalidada
  });
},
validator: (value) {
  if (value!.isEmpty) { return "00/00";} else {
    final components = value.split("/");
    if (components.length == 2) {
      final month = int.tryParse(components[0]);
      final year = int.tryParse(components[1]);
      if ( month!>=0 && month<12 && year!>=0 && year<99) {
        colorValidacion = Colors.black;
        return "Fecha correcta";
      }
    }
  }
  colorValidacion = Colors.red;
  return "Fecha incorrecta";
},
```

Vídeo 28. Notificaciones locales

- Para crear el proyecto con un nombre, lo hacemos desde la consola
 - Navegamos hasta la carpeta del proyecto 28_notificaciones_locales
 - Comando: flutter create --org com.agrgal[espacio]notificaciones_locales
 - Se le pone espacio y ya en el ID se pone **com.agrgal.notificaciones_locales**
- También se le puede cambiar el nombre: [ver marcador](#)

- Extraemos el body como widget, y le ponemos el nombre de la página principal.
- Creamos un ElevatedButton con una función vacía. Al dar clic se mostrará nuestra notificación.
- En el pubspec.yaml agregamos el paquete **flutter pub add flutter_local_notifications**
- En el minuto 5:40 te indica que hay que hacer un cambio en el build.gradle de android>>app pero a mí no me lo marca.
- Dentro de la carpeta **lib**, creamos la subcarpeta **services** y dentro de ella el fichero **notification_service.dart**. Trabajamos en este fichero.
 - importamos el paquete **flutter_local_notifications** con import:
 - Construimos el servicio

```
import 'package:flutter_local_notifications/flutter_local_notifications.dart';
```

// Instancia de nuestro paquete del tipo FlutterLocalNotificationPlugin

```
final FlutterLocalNotificationsPlugin fln = FlutterLocalNotificationsPlugin(); → objeto fln
```

// función que inicia las notificaciones

```
Future<void> initNotifications() async { // Inicio algunas constantes
```

// Constante de inicialización en Android. Entre paréntesis el nombre del icono

```
const AndroidInitializationSettings initializationSettingsAndroid =
AndroidInitializationSettings('app_icon');
```

// En la carpeta android>>app>>src>>main>>res>>drawable debe haber un icono que se llame app_icon. Importante, también en la carpeta drawable-v21

// ¿También en la carpeta drawable-v21? Lo voy a copiar.

// Constante de inicialización en IOS

```
const DarwinInitializationSettings initializationSettingsIOS = DarwinInitializationSettings();
```

// Crear objeto de inicialización como tal

```
const InitializationSettings initializationSettings = InitializationSettings(
  android: initializationSettingsAndroid,
  iOS: initializationSettingsIOS,
);
```

// Activar la inicialización

```
await fln.initialize(initializationSettings);
```

```
}
```

```
=====
```

- Volvemos a main.dart
- Modificamos main() de forma que sea asíncrona, y que inicialice el servicio

```
void main() async { // es una función asíncrona
  WidgetsFlutterBinding.ensureInitialized(); // Se asegura de ejecutar todas las inicializaciones antes de ejecutar runApp
  await initNotifications(); // ejecuta el método de inicialización que hicimos del servicio.
  runApp(const MyApp());
}
```

=====

- Volvemos a la carpeta **lib**, creamos la subcarpeta **services** y dentro de ella el fichero **notification_service.dart**. Creamos la llamada al objeto.

// Función que muestra la notificación

```
Future<void> mostrarNotificacion() async {
  const AndroidNotificationDetails and = AndroidNotificationDetails(
    "0", "mi_canal" ) // Detalles de nuestra notificación ANDROID
  // Así sería en IOS
  const DarwinNotificationDetails dnd = DarwinNotificationDetails();
  const NotificationDetails nd = NotificationDetails( → objeto nd detalles de la notificación
    android: and,
    iOS: dnd,
  );
  await fln.show(1, // una ID, por ejemplo 1. fln, objeto plugin
    "Mi notificacion", // Título de la notificación
    "Esta es una notificación", // Cuerpo de la notificación
    nd, // Objeto detalles de la notificación
  );
}
```

=====

- Y desde el botón, onPressed, llamamos al método **mostrarNotificacion()**;
- Si queremos un comportamiento más prioritario, podemos hacerlo en detalles

```
Future<void> mostrarNotificacion() async {
  const AndroidNotificationDetails and = AndroidNotificationDetails(
    "0", "mi_canal",
    importance: Importance.max,
    priority: Priority.high); // Detalles de nuestra notificación ANDROID
```

- No parece funcionar en mi app. No sé por qué.

Vídeo 29. Sistemas de rutas

- Recordar crear un proyecto desde la línea de comandos con: **flutter create --org com.agrgal[espacio]sistema_rutas, para tenerlo en el ID.**
- Dentro de la carpeta **lib**, creamos la carpeta **pages**.
- Dentro de **pages**, vamos a crear un **home_page.dart**. Será un widget **Stateful** y retornará un **Scaffold** con su **appbar** y un texto que diga **home_page**
- Ahora bien, desde nuestro MaterialApp, en vez de hacer home: Scaffold(..), llamamos a HomePage(),
- El sistema de rutas **es para decirle a Flutter a qué páginas puede navegar.**

1ª forma)

import 'package:sistema_rutas/pages/otra_page.dart'; → import la otra página

.... En el onpress

```
// Forma antigua
// Creo una ruta del tipo MaterialPageRoute a la otra página
MaterialPageRoute route = MaterialPageRoute(builder: (context) => const OtraPage());
// Usando Navigator.push
Navigator.push(context, route);
```

2ª forma)

- En el main.dart, quito home y pongo routes

```
return MaterialApp(
  title: 'Material App',
  initialRoute: "/",
  routes: {
    "/": (context) => const HomePage(), → Una por cada página.
    "/otra": (context) => const OtraPage(), → Le añadido otra página
  },
);
```

- Se me tienen que ir haciendo los import en el main
- En el onPressed del botón de home_page añadido: **Navigator.pushNamed(context, '/otra');** y debe funcionar igual,

Por si hay errores en la llamada

- A veces una página puede tener un llamado dinámico, y puede estar equivocado. ¿Qué ocurre si apuntamos a una página errónea? Se puede hacer una página de error. Por ejemplo, hacemos una página llamada **error404.dart**. No la pongo aquí, pero esa página da información de error y tiene un botón para volver al inicio.
- En la página **main.dart**, la incluyo en la ruta con **"/error": (context) => const Error404()**

Y también en el **main.dart**, para que salga por defecto si hay errores, se incluye

```
onGenerateRoute: (settings) {  
    return MaterialPageRoute(builder: (context) => const Error404(),)  
},
```

Por si necesito que haya una página en nivel superior sin la ←

- Por ejemplo, hago una página de LOGIN, que irá a la de HOME, y esa es la página a la que retornará. Pero la página de LOGIN y HOME están al mismo nivel superior (sin la flecha para retornar).
- Usamos la siguiente orden (dentro de **boton.dart** y se activa con una variable a false).

Navigator.pushReplacementNamed(contexto, pagina);

- Esa orden reemplaza el nivel en el que está la pantalla por otro al mismo nivel. Como el LOGIN está en el nivel superior, HOME se pone en el nivel superior.

NOTA: se borra el stack con estas líneas:

```
Navigator.pushAndRemoveUntil(  
    contexto,  
    MaterialPageRoute(builder: (context)=> const HomePage()),  
    (Route<dynamic> route) => false,);
```

O si se quiere hacer con el nombre de la clave de la página... (ejemplo, '/home')

```
Navigator.pushNamedAndRemoveUntil (  
    contexto,  
    pagina, → variable que almacena la ruta.  
    (Route<dynamic> route) => false,  
);
```

Export páginas

- Puedo hacer un fichero de índice **index.dart** e incluir la exportación de todas las páginas

```
export 'package:sistema_rutas/pages/error404.dart';  
export 'package:sistema_rutas/pages/home_page.dart';  
export 'package:sistema_rutas/pages/login.dart';  
export 'package:sistema_rutas/pages/otra_page.dart';  
export 'package:sistema_rutas/pages/tercera_page.dart';
```

- Y en el main.dart simplemente importo el fichero **index.dart**

```
import 'pages/index.dart';
```

Video 30-34. Firebase CRUD

En documento aparte:
<https://docs.google.com/document/d/1Huh72h3aUEtiCATytqfZTaEiGOONSFgidFOw3z4acJg/edit?usp=sharing>

Video 35. Mini Galería

- Empiezo haciendo la estructura. Una página main, una página **home_page.dart**. En la página home por ahora ponemos un body con un center y un texto.
- Vamos a usar el widget **GridView** (en el body de home_page.dart), que tiene cuatro modos de usarse: con builder, count, custom (más profesional) y extent (más sencillo). En el modo **extent** indico el ancho del máximo elemento.

Primer ejemplo:

```
body: GridView.extent(  
  maxCrossAxisExtent: 150.0, // ancho máximo  
  children:const [  
    Text("Hola"),  
    Text("Hola"),  
    Text("Hola"),  
    Text("Hola"),  
    Text("Hola"),  
    Text("Hola"),  
  ],
```



)

- Bueno, extraigo un método desde el children (que es una lista de Widgets) e incorporo en la carpeta data un list con las imágenes. Simplemente un array con direcciones de imágenes.

```
// Método extraído de children
List<Widget> get _imagesList {
```

```
  List<Widget> listImages = [];
  for (var cadaImagen in images) {
    listImages.add( → añado a la lista
      Image.network( → una imagen
        cadaImagen,
        fit: BoxFit.cover,
        errorBuilder: (context, error, stackTrace) { → si da un error
          return const Icon(Icons.error, size: 140,);
        },
      ));
  }

  return listImages;
}
```



- Y además, añado propiedades al GridView...

```
body: GridView.extent(
  maxCrossAxisExtent: 150.0, // ancho máximo
  padding: const EdgeInsets.all(5.0), // separación del borde del gridView alrededor
  mainAxisSpacing: 5.0, // separación entre filas de elementos
  crossAxisSpacing: 5.0, // en cada fila una separación de 5
  children: _imagesList,
)
```

=====

Para hacer clic en una imagen y que se abra ampliada, utilizo otra página, mandando por parámetro la url. Esta página la llamaré **images_page.dart**

```
import 'package:flutter/material.dart';
class ImagePage extends StatelessWidget {
```

```
  final String url; → creo una variable url
```

```
  const ImagePage({super.key, required this.url}); → solicito que al llamar a ImagePage se pase la url
```

```

@override
Widget build(BuildContext context) {
  return MaterialApp( → Sin AppBar
    debugShowCheckedModeBanner: false,
    title: 'Material App',
    home: Scaffold(
      body: Image.network(url) → muestra la imagen.
    ),
  );
}

```

=====

- Al método donde se cargan los widgets de imágenes le añado un GestureDetector, lo que permite hacer un clic sobre el mismo.

```

// Método extraído de children
List<Widget> get _imagesList {
  List<Widget> listImages = [];
  for (var cadaImagen in images) {
    listImages.add(
      GestureDetector( → me permite añadir un onTap, y navego hasta la página donde se muestra la imagen
        onTap: () {
          Navigator.push(context, MaterialPageRoute(builder: (context)=>ImagePage(url: cadaImagen)));
        },
        child: Image.network(
          cadaImagen,
          fit: BoxFit.cover,
          errorBuilder: (context, error, stackTrace) {
            return const Icon(Icons.error, size: 140,);
          },
        ),
      ));
  }
  return listImages;
}

```

- Y en la página donde se muestran las imágenes:

```

return MaterialApp(
  debugShowCheckedModeBanner: false,
  title: 'Material App',
  home: SafeArea( → SafeArea, permite que no se solape con la barra de tareas del móvil
    child: Scaffold(
      backgroundColor: Colors.black,
      body: Column( → Una columna, permitiendo varios elementos en vertical
        crossAxisAlignment: CrossAxisAlignment.start, → alineación horizontal a la izquierda
        mainAxisAlignment: MainAxisAlignment.spaceBetween, → espacio igualado.
        children: [
          IconButton( → botón para regresar, ya que he quitado el AppBar
            onPressed: () {
              Navigator.pop(context);
            },
          ),
        ],
      ),
    ),
  ),
)

```

```

    },
    icon: const Icon(Icons.arrow_back_ios_sharp, color: Color.fromARGB(255, 202, 186, 138)),
  ),
  Image.network(url),
  const Divider(), → pequeño contenedor que no tiene nada para que la imagen quede centrada
],      ),      ),      ); } → Fin del MaterialApp

```

=====

- Por fin uso el widget **hero** para crear un efecto bonito. Se lo aplico a Image.network tanto en el método de carga de imágenes en **home_page** como en **images_page** (digamos que van por pares). Necesita un tag, que va a ser nuestra uri, en ambos casos.

| home_page | images_page |
|--|--|
| child: Hero (tag: cadaimagen , child: Image.network(cadaimagen, | Hero(tag: url , child: Image.network(url)), |

Ahora, ¿cómo evitar que me cargue imágenes en el grid si no existen? Esa es una buena pregunta.

Voy a crear una página **home_page_safe.dart** para trabajar en esto. Conectaré main a esta página, y trabajaré en ella.

- Ver la asistencia de chatGPT.
- Cargar la biblioteca http. Actualizar paquetes.
- Necesito hacer el import **import 'package:http/http.dart' as http;**
- Declaramos la función que devuelve true o false si existe o no...

// Métodos, funciones

```

Future<bool> existe(String uri) async {
  try {
    final response = await http.head(Uri.parse(uri));
    return response.statusCode == 200;
  } catch (e) {
    return false;
  }
}

```

- Y ahora el método necesitamos que retorne un Future.

```
Future<List<Widget>> get _imagesList async { → lista de widgets future, y es asíncrona
  List<Widget> listImages = [];
  for (var cadalmagen in images) {
    if (await existe(cadalmagen)) { → necesitamos un await. Llamamos a la función.
      listImages.add(
        GestureDetector(
          onTap: () {
            Navigator.push(context, MaterialPageRoute(builder: (context)=>ImagePage(url: cadalmagen)));
          },
          child: Hero(
            tag: cadalmagen,
            child: Image.network(
              cadalmagen,
              fit: BoxFit.cover,
              errorBuilder: (context, error, stackTrace) {
                return const Icon(Icons.error, size: 140,);
              },
            ),
          ),
        ); // Fin de list add
      } // fin del if
    }
  }
  return listImages;
}
```

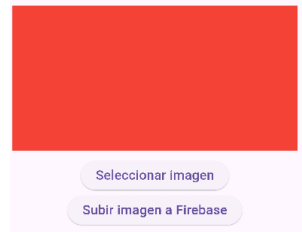
- Y el gridview debe estar wrapeado por un Futurebuilder, que accede a la función _imagesList

```
body: FutureBuilder( → objeto Futurebuilder
  future: _imagesList, → lista de widgets que retorna
  builder: (context, snapshot) {
    if (snapshot.hasData) {
      return GridView.extent (
        maxCrossAxisExtent: 150.0, // ancho máximo
        padding: const EdgeInsets.all(5.0), // separación del borde del gridview alrededor
        mainAxisSpacing: 5.0, // separación entre filas de elementos
        crossAxisSpacing: 5.0, // en cada fila una separación de 5
        children: snapshot.data ?? [], → si es nulo, que devuelva []
      );
    } else {
      return const Center(child: CircularProgressIndicator());
    }
  } )
```

Vídeo 36 - 37. FireBase Storage

- Como usaremos Firebase debemos crear el proyecto con el id. Empezamos a hacerlo desde la terminal con **flutter create --org com.agrgal uploads_firebase**.
- Creamos un `home_page` en página aparte. Tendrá un container (rectángulo rojo) y dos botones (elevated button), uno para seleccionar imagen y otro para subirla a firebase.
- Con **flutter pub add image_picker** instalar esta biblioteca para tomar fotos y guardar imágenes.
- Construimos una carpeta **services** y dentro una página `select_image.dart`

Material App Bar



`import 'package:image_picker/image_picker.dart';` → **importante el import**

```
Future<XFile?> getImage() async { → función que retorna un XFile. Tipo Future, asíncrona
  final ImagePicker picker = ImagePicker(); → instancia de Image Picker
  final XFile? image = await picker.pickImage( → escoger imagen de galería. Con espera.
    source: ImageSource.gallery, → esto escoge la galería del móvil como lugar de donde escoger la imagen
    → También se puede usar ImageSource.camera, para hacer una foto.
  );
  return image; → retorna una imagen, que puede ser null
}
```

Y en `home_page.dart`, en nuestro **ElevatedButton** de “Seleccionar imagen”

- Primero , convierto a `Stateful`
- Añadir una variable del tipo `File? = milimagenASubir;` .Probablemente tendré que importar **`import 'dart:io';`**
- Y entonces, el `onPressed` del `ElevatedButton` queda:

```
onPressed: () async {
  final XFile? imagen = await getImage(); → seleccionar imagen de galería
  setState() { → recargar estado.
    if (imagen?.path != null ) { → hago una doble comprobación de que no es nulo
      milimagenASubir = File(imagen!.path);
    }
  }); },
```

- ¿Qué queda? Bueno, mostrará el container rojo si no tiene nada, pero una imagen si la hemos seleccionado. Por lo tanto, podemos hacer un condicional que muestre uno u otro.


```
body: Column(
  children: [ → si la variable es no nula que muestre la fotografía, si no el container.
    milimagenASubir!=null ? Image.file(milimagenASubir!) : Container (
      width: double.infinity, height: 200.0, margin: const EdgeInsets.all(10),.....
```

Segunda parte, subir a Firebase

- Primero, lugar donde se coloca la imagen “más arreglado”

```
milimagenASubir!=null
  ? SizedBox(
    width: double.infinity,
    height: 400.0,
    child: Image.file(
      milimagenASubir!,
      fit: BoxFit.contain, → con un fit, wrapedo en un SizedBox....
    ),
  )
```

1. Instalamos el core de firebase: desde terminal **dart pub add firebase_core**. Actualizamos.
 2. Instalamos el paquete **dart pub global activate flutterfire_cli**
 3. Configuramos con **flutterfire configure**. Escoger opciones y configurar el id del proyecto. NOTA: CTRL + “-” si es necesario escribir más en el terminal.
- En la consola de Firebase, en COMPILACIÓN, ahora escojo STORAGE. Comienzo, escojo modo de pruebas (pero cambio a un año el período, entrando en Reglas) y el servidor eur3.
 - Instalamos en flutter el paquete **flutter pub add firebase_storage**
 - En **main.dart** conecto con firebase

```
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:uploads_firebase/firebase_options.dart';
// páginas
import 'pages/home_page.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized(); → aseguro que los Widgets se inicializan
  await Firebase.initializeApp( → Inicializo Firebase con las opciones de firebase_options.dart
    options: DefaultFirebaseOptions.currentPlatform
  );
  runApp(const MyApp()); → Después, inicio mi aplicación
}
```

- Creo el servicio que conecta con STORAGE. Fichero **upload_image.dart**

import 'dart:io'; → **importar entrada salida, para permitir tipo File**

import 'package:firebase_storage/firebase_storage.dart'; → **importar paquete firebase_storage**

final FirebaseStorage almacen = FirebaseStorage.instance; → **crear una instancia de FirebaseStorage**

Future<bool> **subirImagen**(File imagen) async { → **función que devuelve true si sube la imagen. Asíncrona**

print(imagen.path); → **Necesito primero el path de la imagen, para conocer su nombre.**

final String nombreFichero = imagen.path.split("/").last; → **obtengo su nombre, desechando el resto de la ruta**

Reference refUpload = almacen.ref().child("misImagenes").child(nombreFichero);

→ **creo una referencia en STORAGE donde se almacenará la imagen. Los "child" puede ser unas carpetas o subcarpetas. Acaba en el nombre del fichero.**

final UploadTask uploadTask = refUpload.putFile(imagen); → **Hago una tarea de subida. Subo la imagen**

print(uploadTask); → **Imprimo esta tarea.**

final TaskSnapshot snapshot = await uploadTask.whenComplete(()=>true);

→ **obtengo datos de esa tarea (snapshot). Cuando se complete, devolverá true.**

print(snapshot); → **Imprimo estos datos.**

final String url = await snapshot.ref.getDownloadURL(); → **obtengo la url de dónde se almacenó esa imagen**

print(url); → **imprimo esa url.**

if (snapshot.state == TaskState.success) { return true; } else { return false; }

→ **Por fin, devuelve el estado de esos datos. true or false.**

} // Fin de la función

=====

- En home_page.dart, en el otro botón

ElevatedButton(

onPressed: () async { → **se convierte en una función asíncrona**

if (milmagenASubir==null) { return; } → **si no hay imagen, acaba**

final uploaded = await subirImagen(milmagenASubir!); → **llamo al servicio subirImagen y entrego la imagen**

if (uploaded) { → **Si es verdad que la he subido...**

ScaffoldMessenger.of(context).showSnackBar(→ **mensajito de que está subida**

const SnackBar(content: Text("Imagen subida correctamente"))

); } else {

ScaffoldMessenger.of(context).showSnackBar(→ **mensajito de que no se ha podido subir.**

const SnackBar(content: Text("Error al subir la imagen"))

);

} },

child: const Text("Subir imagen a Firebase")), → **texto del botón.**

NOTA: en **ScaffoldMessenger.of(context).showSnackBar** me da unos avisos. Como que el context no funciona en una función asíncrona. En el fichero **analysis_options.yaml** se crea una instancia del tipo:

analyzer:

errors:

use_build_context_synchronously: ignore

Vídeo 38. Provider (estado general del programa)

- Vamos a hacer un contador provider, para almacenar el estado general.
- Se crea un main.dart, un home_page.dart con un Scaffold stateful, y en **home_page.dart** se pone un bottomNavigationBar con tres botones:

```
bottomNavigationBar: BottomNavigationBar(  
  items: const [  
    BottomNavigationBarItem(  
      icon: Icon(Icons.home),  
      label: "Page 1",  
    ),  
    BottomNavigationBarItem(  
      icon: Icon(Icons.business),  
      label: "Page 2",  
    ),  
    BottomNavigationBarItem(  
      icon: Icon(Icons.school),  
      label: "Page 3",  
    ),  
  ],  
)
```

- Creamos una página 1, **page1.dart** con un statefullwidget, con un center y un texto de página 1. Y copiamos a **page2.dart** (cambiamos los datos para la 2) y asimismo para **page3.dart**.

En nuestro **home_page.dart**

```
class _HomeScreenState extends State<HomeScreen> {  
  int _selectedIndex = 0; → crear variable que almacena el index  
  final List<Widget> _pages = <Widget> [ → una lista con las tres páginas  
    const Page1(), const Page2(), const Page3(), ];  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: const Text('Contador de Provider'),  
      ),  
      body: _pages[_selectedIndex], → página que se muestra en el body.  
      bottomNavigationBar: BottomNavigationBar( → Navigation Bar  
        onTap: (int index){ → al hacer clic, le manda el índice del BottomNavigationBar  
          setState(() {  
            _selectedIndex = index; → el índice seleccionado es el que recarga la página.  
          });  
        },  
        currentIndex: _selectedIndex, → Y al recargar, muestra el icono que está seleccionado  
        items: const [  
          BottomNavigationBarItem(  
            icon: Icon(Icons.home),  
            label: "Page 1",  
          ),  
          BottomNavigationBarItem(  
            icon: Icon(Icons.business),  
            label: "Page 2",  
          ),  
          BottomNavigationBarItem(  
            icon: Icon(Icons.school),  
            label: "Page 3",  
          ),  
        ],  
      ),  
    );  
  }  
}
```

En la página **page1.dart**

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

```
.....
```

```
class _Page1State extends State<Page1> {
```

```
  int _counter = 0; → hago un contador a 0
```

```
  @override
```

```
  Widget build(BuildContext context) {
```

```
    return Center(
```

```
      child: Column(
```

```
        children: [
```

```
          Text(_counter.toString(), style: const TextStyle(fontSize: 50.0,)), → lo muestro en pantalla
```

```
          ElevatedButton(
```

```
            onPressed: () {
```

```
              setState(() { → cambio su estado, sumándole uno.
```

```
                _counter++;
```

```
              });
```

```
            },
```

```
            child: const Text("Sumar"),
```

```
          ), ), ); } }
```

- **Pero esto sólo funciona en page1. Si accedo a page2, el estado se elimina, y al volver a page1 sigue contando desde cero.**
- Para manejar estados globales, instalamos el paquete provider: **flutter pub add provider**
- Creamos una carpeta llamada **provider** y un fichero llamado **counter_provider.dart**

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

```
class CounterProvider with ChangeNotifier { → clase provider con changenotifier
```

```
  int _counter = 0; → nuestro contador
```

```
  // Método getter. Porque debo acceder así a una variable local
```

```
  int get counter => _counter; // método para poder obtener la variable local
```

```
  // Método setter. Así modifico la variable local
```

```
  void increment() {
```

```
    _counter++;
```

```
    notifyListeners(); → importante: así notifico a los widgets cualquier cambio, los que están obteniendo el valor con un get
```

```
  } }
```

Ahora, necesito activarlo para toda la aplicación. Se hace desde **main.dart** cambiando el main a:

```
void main()
{
  runApp(
    ChangeNotifierProvider( → método usado
      create: (_)=>CounterProvider(),
      child: const MyApp(), → después ejecuta la aplicación
    );
}
```

- Volvemos a **page1.dart**. Retiramos la definición de `_counter` y cambiamos por...

```
return Center(
  child: Column(
    children: [
      Text(
        context.watch<CounterProvider>().counter.toString(), → observar la variable counter del provider
        style: const TextStyle(fontSize: 50.0,)
      ),
      ElevatedButton(
        onPressed: (){
          // setState() { // cambio el estado. → no hace falta el setState
          // _counter++;
          context.read<CounterProvider>().increment(); → llamada al método increment
          // }); → no hace falta, porque el notifyListeners recargará ese dato donde haga falta.
        },
        child: const Text("Sumar"),
      ),
    ],
  );
```

- Jugando un poco con los temas: librería GoogleFonts **flutter pub add google_fonts**

| | |
|---|--|
| theme: ThemeData(useMaterial3: true, // Define the default brightness and colors. colorScheme: ColorScheme.fromSeed(seedColor: Colors.blue, brightness: Brightness.light,), // Define the default `TextTheme`. Use this to specify the default // text styling for headlines, titles, bodies of text, and more. textTheme: TextTheme(displayLarge: const TextStyle(fontSize: 52, fontWeight: FontWeight.bold,), titleLarge: GoogleFonts.oswald(fontSize: 30, fontStyle: FontStyle.italic,), bodyMedium: GoogleFonts.merriweather(), displaySmall: GoogleFonts.pacifico(),),), | |
|---|--|

Vídeo 39. Navegación tabs superior

- Creamos nuestro proyecto y lo llamamos **navegacion_tab_superior**.
- En el Scaffold de HomePage wrapeamos con DewfaultTanController y en la AppBar añadimos bottom, con los controles TabBar y Tab

```
return DefaultTabController(  
  length: 3, → necesito cuántos son  
  child: Scaffold(  
    appBar: AppBar(  
      title: const Text('Tab application'),  
      bottom: const TabBar( → tarBar  
        tabs: [  
          Tab(icon: Icon(Icons.abc)), → Uno, dos, tres iconos  
          Tab(icon: Icon(Icons.abc)),  
          Tab(icon: Icon(Icons.abc)),  
        ],  
      ),  
    ),  
    body: const Center(  
      child: Text('Hello World'),  
    ),  
  ), ); }
```

- Y en el body podemos poner tres tabs, con TabBarView. Lo cual queda muy chulo...

```
body: const TabBarView(  
  children: [  
    Center(child: Text('Bicicleta'),),  
    Center(child: Text('Barco'),),  
    Center(child: Text('Coche'),),  
  ],  
),
```

- Poco más. Podemos poner los children del tabView como widgets Stateless aparte, en ficheros aparte, y llamarlos así.

Vídeo 40. Badges

- Pequeños dibujos que aparecen acompañando a los iconos, como importante, globito con números, etc.
- Empezamos nuestro proyecto con create en el terminal: **flutter create --org com.agrgal badges_ejemplo**
- Simplemente envolviendo un widget con **Badge**, ya aparece un punto en su parte superior derecha.
- **BottomNavigationBar**, iconos con Badges

```
bottomNavigationBar: BottomNavigationBar(  
  items: const [  
    BottomNavigationBarItem(  
      icon: Badge(label: Text('5'), child: Icon(Icons.home)),  
      label: "Casa"  
    ),  
    BottomNavigationBarItem(  
      icon: Badge( → envuelvo con Badge  
        label: Text("10"), → necesita una etiqueta  
        backgroundColor: Colors.amberAccent,  
        child: Icon(Icons.car_rental)  
      ),  
      label: "Coche"  
    ),  
    BottomNavigationBarItem(  
      icon: Icon(Icons.stairs_rounded),  
      label: "Escaleras"  
    ),  
  ],  
),
```

- También tiene otros modificadores como `isLabelVisible`, que poniéndola a `true` o `false` se ve o no el badge. Se puede poner una variable booleana, y por ejemplo con un botón, en `onPressed`, cambiar el estado del widget cambiando la variable. Para ello debe ser un `Stateful`. Recuerdo que las variables van entre **Class Extend** y **Widget Builder**.

A este mensaje, le voy a poner un badge cuando haga `onTab`

```
Container(  
  width: MediaQuery.of(context).size.width*0.7,  
  margin: const EdgeInsets.all(5.0),  
  padding: const EdgeInsets.all(10.0),  
  alignment: Alignment.center,  
  color: Colors.cyan,  
  child: const Text(  
    textAlign: TextAlign.center,  
    "Este es un mensaje que me enviaron a través de un chat",  
    style: TextStyle(color: Colors.white,)  
  ),  
),
```



Este es un mensaje que me
enviaron a través de un chat

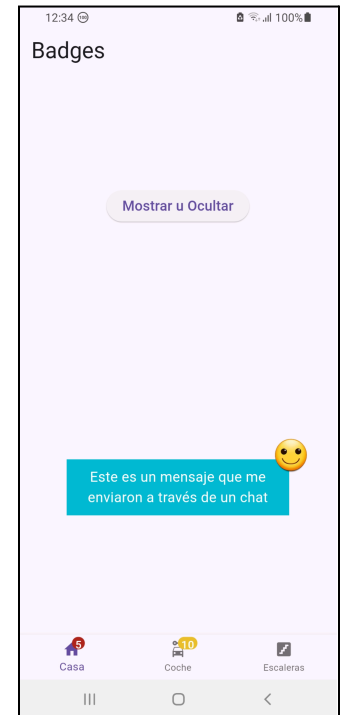
- Vamos a envolver el container en un **inkwell**, de manera que podamos agregar funcionalidad, como el onLongPressed. Y de paso, ya pongo algunas de las opciones de Badge

InkWell(

```

onLongPress: (){
  setState() {
    _isEmojiVisible = !_isEmojiVisible;
  });
},
child: Badge(
  offset: const Offset(30, -30),
  backgroundColor: Colors.transparent,
  isLabelVisible: _isEmojiVisible,
  smallSize: 30.0,
  largeSize: 60.0,
  label: const Text("😊", style: TextStyle(fontSize: 30.0,)), //:smile
  child: Container(
    width: MediaQuery.of(context).size.width*0.7,
    margin: const EdgeInsets.all(5.0),
    padding: const EdgeInsets.all(10.0),
    alignment: Alignment.center,
    color: Colors.cyan,
    child: const Text(
      textAlign: TextAlign.center,
      "Este es un mensaje que me enviaron a través de un chat",
      style: TextStyle(color: Colors.white,)
    ),
  ),
),
),
),
),

```



Vídeo 41. Panel desplegable (SlidingUpPanel).

- Creamos lo de siempre. Un home_page, lo saco de main, etc.
- Utilizaremos el paquete **sliding_up_panel**, en la web https://pub.dev/documentation/sliding_up_panel/latest/
- En este caso tenemos que ir al pubspec.yaml y en dependencias (dependencies:) escribir **sliding_up_panel: ^2.0.0+1**. Vamos, lo que viene en la página. CTRL+s para guardar y que se cargue el paquete.
- Funcionamiento sencillo con slidinguppanel. En mi Scaffold...

```
return Scaffold(  
  appBar: AppBar(  
    title: const Text('Panel Deslizable'),  
    backgroundColor: Colors.yellow[50],  
  ),  
  body: SlidingUpPanel(  
    panel: const Text("Soy el panel"), → lo que es el panel  
    body: const Center( child: Text('Soy el body')), → lo que es el body  
  ),  
);
```

| SlidingUpPanel | Panel_Up |
|---|--|
| <pre>body: SlidingUpPanel(color: Colors.transparent, minHeight: 30, maxHeight: MediaQuery.of(context).size.height*0.8,</pre> | <pre>return Container(decoration: BoxDecoration(color: Colors.red, borderRadius: BorderRadius.circular(15.0)), child: const Column(children:[Icon(Icons.drag_handle_rounded), Image.asset("assets/images/map.jpg"),],),</pre> |
| Con estos dos códigos, consigo las esquinas redondeadas del panel deslizante, lo que le da más presencia visual. | |

- Importante: ahora voy a cargar algunas imágenes. Creo una carpeta que se llama assets/images y dentro meto una imagen que se llamará **map.jpg**
- En el pubspec.yaml y en la sección **flutter:** descomentar otra llamada **assets**, donde introduciré la ruta a mi imagen.

```
# The following section is specific  
flutter:  
  
# The following line ensures that  
# included with your application,  
# the material Icons class.  
uses-material-design: true  
  
# To add assets to your application  
assets:  
  - assets/images/map.jpg  
  - images/a_dot_ham.jpeg
```

- Para introducir la imagen `Image.asset("assets/images/map.jpg")`, en el `panel_up` (ver arriba el código en rojo).
- Aprovechamos para dibujar en el `body` una lista. Lo repasamos...

```
return Container(  
  child: ListView.builder(  
    itemCount: persons.length,  
    itemBuilder: (context, index) {  
      return ListTile(  
        title: Text(persons[index]),  
        trailing: const Icon(Icons.arrow_forward_ios_outlined),  
        leading: CircleAvatar(  
          child: Text(persons[index].substring(0,1))  
        ),  
      );  
    },  
  ),  
);
```

Vídeo 42. Validar formularios en Flutter

- Creamos nuestro proyecto, separamos el home y creamos un **formulario.dart**.
- En éste creamos un stateful con el nombre de **Formulario**.
- En el body de **home_page.dart**, llamamos al formulario.
- El widget que vamos a usar es **Form**. Dentro tendrá un **Column** con varios **inputs**.
- Los inputs, mejor que con **TextField**, trabajaremos con **TextFormField**. Este último se relaciona directamente con el formulario creado.

Nuestro formulario queda como algo así

```
class _FormularioState extends State<Formulario> {

  // Variable creada de tipo GlobalKey asociada al estado de un formulario
  final GlobalKey<FormState> _formularioEstado = GlobalKey<FormState>();
  → definición de una variable global tipo estado de formulario.

  @override
  Widget build(BuildContext context) {
    return Container(
      padding: const EdgeInsets.all(20),
      decoration: BoxDecoration(
        color: Colors.yellow[50],
      ),
      child: Form(
        key: _formularioEstado, → asocio esa clave a este formulario.
        child: Column(
          children: [
            Container(
              decoration: BoxDecoration(
                borderRadius: BorderRadius.circular(10),
                border: Border.all(color: Colors.grey),
              ),
              child: TextFormField(
                validator: (value){ → agrego un validador según un valor
                // si retorna algo nulo es que salió bien, y si una cadena es que salió mal.
                if (!value!.contains("@")) { // si el valor NO contiene una arroba
                return "Correo no válido"; → si hay errores.
                }
                return null; → todo perfecto
                },
                decoration: const InputDecoration(
                  hintText: "correo electrónico", → hay que poner una pista
                  border: InputBorder.none,
                  fillColor: Colors.amber,
                  filled: true,
                ),
              ),
            ),
          ],
        ),
      ),
    );
  }
}
```

```

borderRadius: BorderRadius.circular(10),
border: Border.all(color: Colors.grey),
),
child: TextFormField(
  obscureText: true, → si necesito que sea una contraseña.
  validator: (value){
    // si retorna algo nulo es que salió bien, y si una cadena es que salió mal.
    if (value!.length<8) { // si el valor NO contiene una arroba
      return "Contraseña muy corta";
    }
    return null;
  }, → igual que antes, comprueba en este caso que su longitud es ocho o mayor.
  decoration: const InputDecoration(
    hintText: "password",
    border: InputBorder.none,
    fillColor: Colors.amber,
    filled: true,
  ),
),
Expanded(
  child: Container(),
),
Container(
  width: double.infinity,
  child: ElevatedButton(
    onPressed: (){
      if(_formularioEstado.currentState!.validate()){ → validando el proceso en el botón de enviar.
        // realizamos el resto del proceso
        print("Éxito");
      } else {
        print("Hay errores");
      }
    },
    child: const Text("Enviar"),)
  ),
),
);
}
}

```