MINISTERIO DE TECNOLOGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES

Unidad 2

Interfáces Gráficas de Usuario

Universidad Tecnológica de Pereira

MisionTIC 2022

MINISTERIO DE TECNOLOGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES

El Lenguaje Dart

El lenguaje Dart

Dart es un lenguaje de tipo estático, y tiene un tipo de sistema “sonoro”, el cual garantiza que una expresión de un tipo determinado, no pueda producir un valor de otro tipo, por lo tanto, no hay sorpresas.

El lenguaje de programación Dart nació en 2011 con el modesto objetivo de arreglar la web. En otras palabras, intentar reemplazar a JavaScript en el navegador.

```
// Define a function.
void _printInteger(int aNumber) {
  print('The number is $aNumber.');
}

// This is where the app starts executing.
void main() {
  var number = 42;
  _printInteger(number);
}
```

El lenguaje Dart

Variables

```
var name = 'Bob';
String name = 'Bob';
```

Valor por defecto y valores nulos

```
int? lineCount; // null
int lineCount = 0;
```

Variables de inicialización tardía `late`

```
late String description;
late String temperature = readThermometer();
```

Variables constantes

```
final name = 'Bob';
final String nickname = 'Bobby';
const bar = 1000000;
const double atm = 1.01325 * bar;
```

Tipos de datos

- Numbers (**int**, **double**)
- Strings (**String**)
- Booleans (**bool**)
- Lists (**List**, also known as arrays)
- Sets (**Set**)
- Maps (**Map**)
- Runes (**Runes**; often replaced by the characters API)
- Symbols (**Symbol**)
- The value null (**Null**)

El lenguaje Dart

Cadenas

```
var s1 = 'Single quotes work well for string
literals.';
var s2 = "Double quotes work just as well.";
var s3 = 'It\'s easy to escape the string
delimiter.';
var s4 = "It's even easier to use the other
delimiter.";

var s1 = 'String '
    'concatenation'
    " works even over line breaks.";

var s1 = '''
You can create
multi-line strings like this one.
''';
```

Booleanos

```
// Check for an empty string.
var fullName = '';
assert(fullName.isEmpty);

// Check for zero.
var hitPoints = 0;
assert(hitPoints <= 0);

// Check for null.
var unicorn;
assert(unicorn == null);

// Check for NaN.
var iMeantToDoThis = 0 / 0;
assert(iMeantToDoThis.isNaN);
```

El lenguaje Dart

Listas

```
var list = [1, 2, 3];

var list = [
  'Car',
  'Boat',
  'Plane',
];

var constantList = const [1, 2, 3];

var list = [1, 2, 3];
var list2 = [0, ...]?list];
```

```
var nav = ['Home', 'Furniture', 'Plants',
           if (promoActive) 'Outlet'];

var listOfInts = [1, 2, 3];
var listOfStrings = ['#0',
                     for (var i in listOfInts) '#$i'];
```

Conjuntos

```
var halogens = {'fluorine', 'chlorine', 'bromine',
'iodine', 'astatine'};

var names = <String>{};
```

El lenguaje Dart

Mapas

```
var gifts = {'first': 'partridge'};
gifts['fourth'] = 'calling birds';
```

```
var gifts = {'first': 'partridge'};
gifts['first'] == 'partridge';
```

```
final constantMap = const {
  2: 'helium',
  10: 'neon',
  18: 'argon',
};
```

Funciones

```
bool isNoble(int atomicNumber) {
  return _nobleGases[atomicNumber] != null;
}
```

```
bool isNoble(int atomicNumber) =>
        _nobleGases[atomicNumber] != null;
```

Parámetros nombrados

```
void enableFlags({bool? bold, bool? hidden})
{...}
enableFlags(bold: true, hidden: false);
```

El lenguaje Dart

Funciones

Parámetros posicionales opcionales

```
String say(String from, String msg, [String?
device]) {
  var result = '$from says $msg';
  if (device != null) {
    result = '$result with a $device';
  }

  return result;
}

say('Bob', 'Howdy')

say('Bob', 'Howdy', 'smoke signal')
```

Funciones

Parámetros posicionales opcionales

```
void doStuff(
    {List<int> list = const [1, 2, 3],
     Map<String, String> gifts = const {
       'first': 'paper',
       'second': 'cotton',
       'third': 'leather'
     }}) {
  print('list:  $list');
  print('gifts: $gifts');
}
```

El lenguaje Dart

Función main()

```
void main() {
  print('Hello, World!');
}

void main(List<String> arguments) {
  print(arguments);
}
```

Funciones anónimas

```
const list = ['apples', 'bananas', 'oranges'];
list.forEach((item) {
  print('${list.indexOf(item)}: $item');
});
```

Funciones como objetos de primera clase

```
void printElement(int element) {
  print(element);
}

var list = [1, 2, 3];
list.forEach(printElement);

var loudify = (msg) =>
              '!!! ${msg.toUpperCase()} !!!';
```

Closures

```
Function makeAdder(int addBy) {
  return (int i) => addBy + i;
}
```

El lenguaje Dart

Operadores

```
a++
a + b
a = b
a == b
c ? a : b
a is T
```

```
(employee as Person).firstName = 'Bob';
```

```
if (employee is Person) {
  // Type check
  employee.firstName = 'Bob';
}
```

```
var a = 2;
a *= 3;
```

```
if (!done && (col == 0 || col == 3)) {
  // ...Do something...
}
```

```
var visibility = isPublic ? 'public' :
'private';
```

Para más información revisar el Tour del lenguaje

El lenguaje Dart

Sentencias de control de flujo

```
if (isRaining()) {
  you.bringRainCoat();
} else if (isSnowing()) {
  you.wearJacket();
} else {
  car.putTopDown();
}

var message = StringBuffer('Dart is fun');
for (var i = 0; i < 5; i++) {
  message.write('!');
}
```

```
for (final candidate in candidates) {
  candidate.interview();
}

var collection = [1, 2, 3];
collection.forEach(print);

while (!isDone()) {
  doSomething();
}

do {
  printLine();
} while (!atEndOfPage());
```

El lenguaje Dart

Sentencias de control de flujo

```
var command = 'OPEN';
switch (command) {
  case 'CLOSED':
    executeClosed();
    break;
  case 'PENDING':
    executePending();
    break;
  case 'APPROVED':
    executeApproved();
    break;
  case 'DENIED':
    executeDenied();
    break;
  case 'OPEN':
    executeOpen();
    break;
  default:
    executeUnknown();
}
```

Excepciones

```
throw FormatException('Expected at least 1 section');

throw 'Out of llamas!';

try {
  breedMoreLlamas();
} on OutOfLlamasException {
  // A specific exception
  buyMoreLlamas();
} on Exception catch (e) {
  // Anything else that is an exception
  print('Unknown exception: $e');
} catch (e) {
  // No specified type, handles all
  print('Something really unknown: $e');
}
```

El lenguaje Dart

Clases

```
class Point {
  final double x;
  final double y;

  Point(this.x, this.y);

  Point.named({required this.x, required
this.y});

  double distanceTo(Point other) {
    var dx = x - other.x;
    var dy = y - other.y;
    return sqrt(dx * dx + dy * dy);
  }

}

var p1 = Point(2, 2);
var p2 = Point.fromJson({'x': 1, 'y': 2});
```

```
abstract class Doer {
  void doSomething();
}

class EffectiveDoer extends Doer {
  @override
  void doSomething() {
  }
}
```

Nota:

```
enum Color { red, green, blue }
```

El lenguaje Dart

Usar bibliotecas

```
import 'package:lib1/lib1.dart';
import 'package:lib2/lib2.dart' as lib2;

// Uses Element from lib1.
Element element1 = Element();

// Uses Element from lib2.
lib2.Element element2 = lib2.Element();
```

```
// Import only foo.
import 'package:lib1/lib1.dart' show
foo;

// Import all names EXCEPT foo.
import 'package:lib2/lib2.dart' hide
foo;
```

Carga diferida de una biblioteca

```
import 'package:greetings/hello.dart' deferred as hello;

Future<void> greet() async {
  await hello.loadLibrary();
  hello.printGreeting();
}
```

Plantillas (Generics)

```
var names = <String>['Seth', 'Kathy', 'Lars'];
var uniqueNames = <String>{'Seth', 'Kathy', 'Lars'};
var pages = <String, String>{
  'index.html': 'Homepage',
  'robots.txt': 'Hints for web robots',
  'humans.txt': 'We are people, not machines'
};
```

El lenguaje Dart

Funciones asíncronas

```
Future<void> checkVersion() async {
  var version = await lookUpVersion();
  // Do something with version
}

void main() async {
  checkVersion();
  print('In main: version is ${await
lookUpVersion()}');
}

void main() async {
  await for (final request in requestServer) {
    handleRequest(request);
  }
}
```

Generadores

```
Iterable<int> naturalsTo(int n) sync* {
  int k = 0;
  while (k < n) yield k++;
}

Stream<int> asynchronousNaturalsTo(int n) async* {
  int k = 0;
  while (k < n) yield k++;
}
```

Comentarios

```
// TODO: refactor into an AbstractLlamaGreetingFactory?

/*
 * This is a lot of work. Consider raising chickens.
 */

/// A domesticated South American camelid (Lama glama).
```

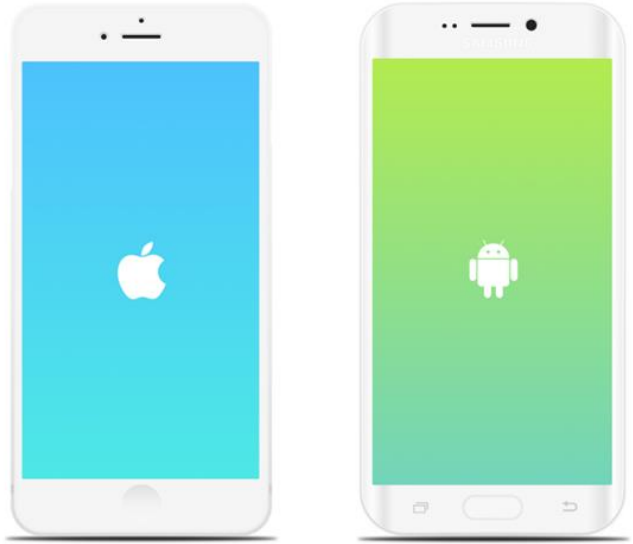
Universidad Tecnológica de Pereira

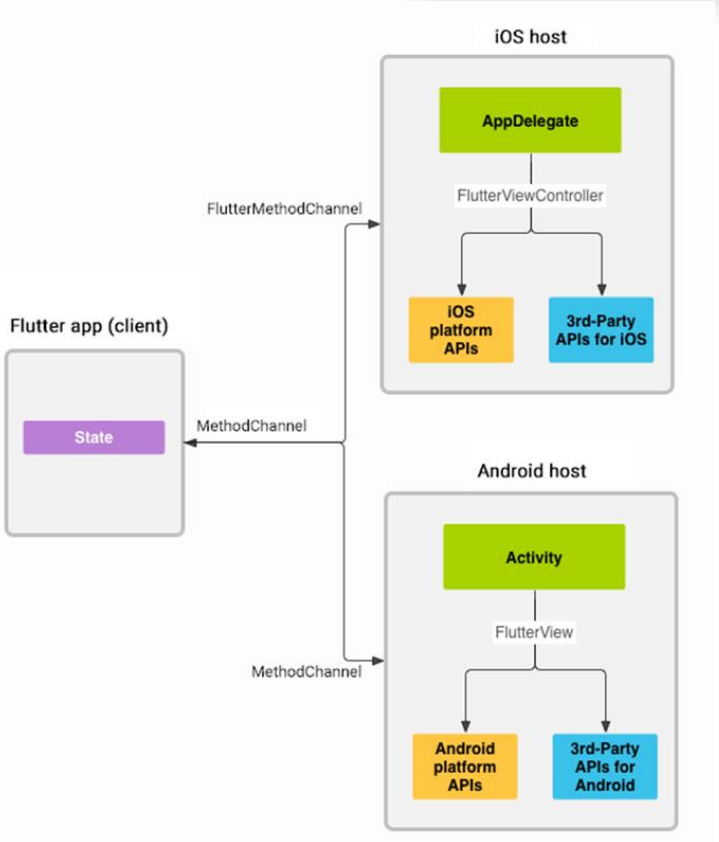
MisionTIC2022

MINISTERIO DE TECNOLOGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES

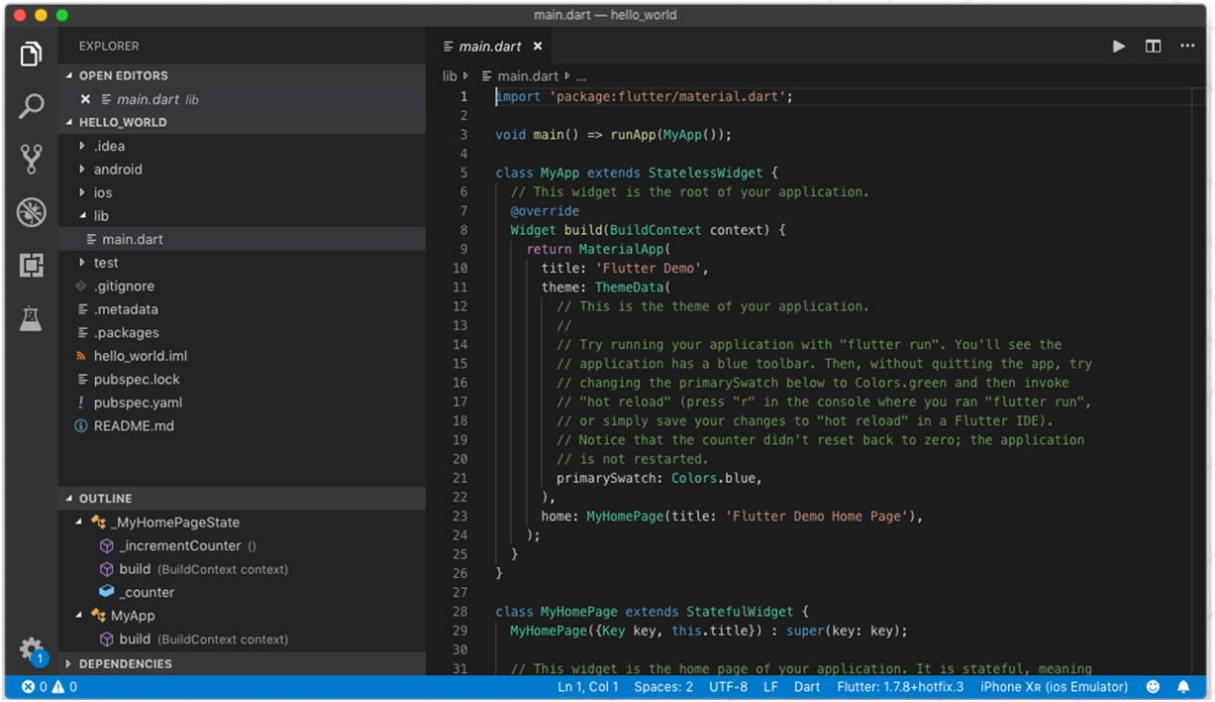
Flutter: Introducción a los Widgets

Flutter para Android y iOS





Composición de un proyecto en Flutter



Elementos de interés:

- Archivo **lib/main.dart**
- Carpeta **ios**
- Carpeta **android**
- Archivo **pubspec.yaml**

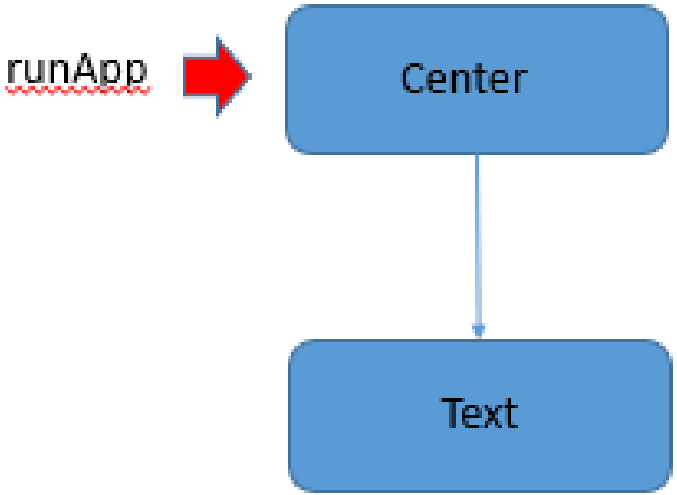
Hello World!

main.dart ● ! pubspec.yaml

lib > main.dart > ...

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

Run | Debug
void main() {
  runApp(
    Center(
      child: Text(
        'Hello, world!',
        textDirection: TextDirection.ltr,
      ), // Text
    ), // Center
  );
}
```

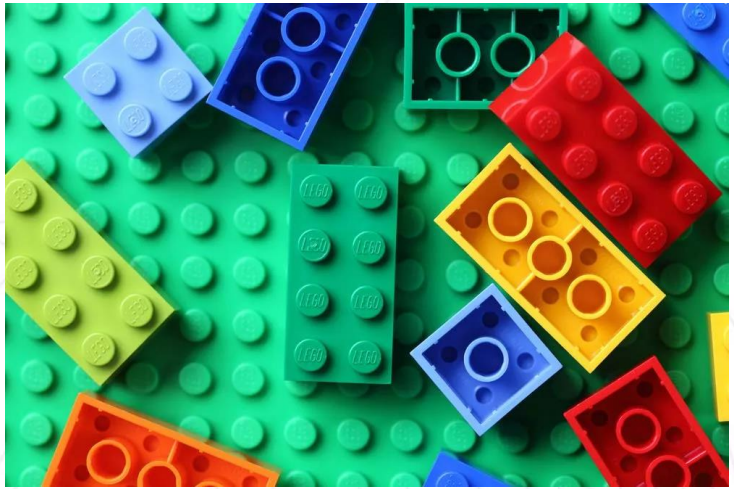


¿Qué son los Widgets?

Los widgets en flutter con los componentes estructurales de una aplicación móvil.

Pueden ser referenciados como si fueran piezas de Lego. Hay diferentes tipos de widgets que pueden ser puestos sobre otros legos más grandes o contenedores.

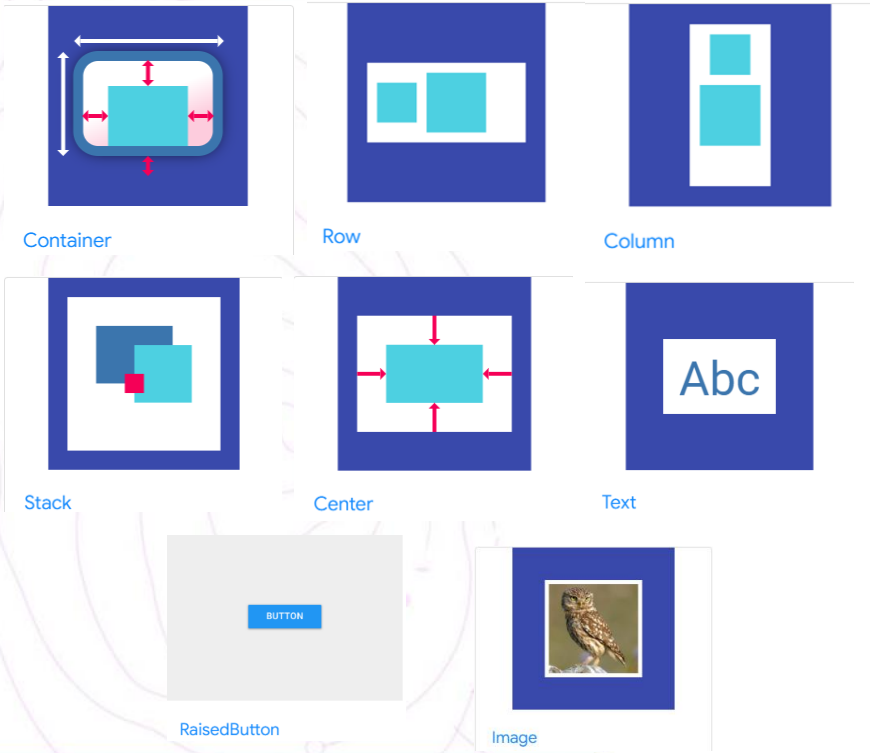
Una aplicación Flutter se construye ensamblando widgets hasta obtener el producto terminado



Widgets básicos

Flutter viene con un conjunto de potentes widgets básicos, de los cuales los siguientes son de uso muy común:

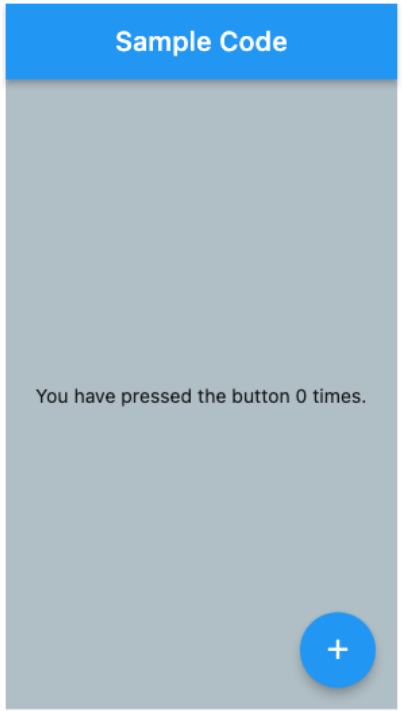
- **Text**: Permite crear una cadena de texto con estilo.
- **Row**, **Column**: Permiten crear layouts tanto en la dirección horizontal (Row) como en la vertical (Column).
- **Stack**: Permite apilar los widgets uno encima del otro en el orden como se pintan.
- **Center**: Widget que centra a su hijo en sí mismo.
- **Container**: Widget que te permite crear un elemento visual rectangular.
- **Image**: Muestra una imagen.
- **RaisedButton**: Un botón elevado de Material Design.

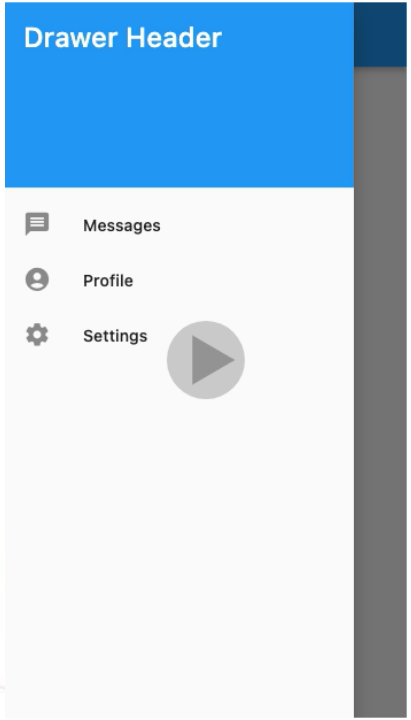


https://flutter-es.firebaseapp.com/docs/development/ui/widgets

Widgets de Material Design

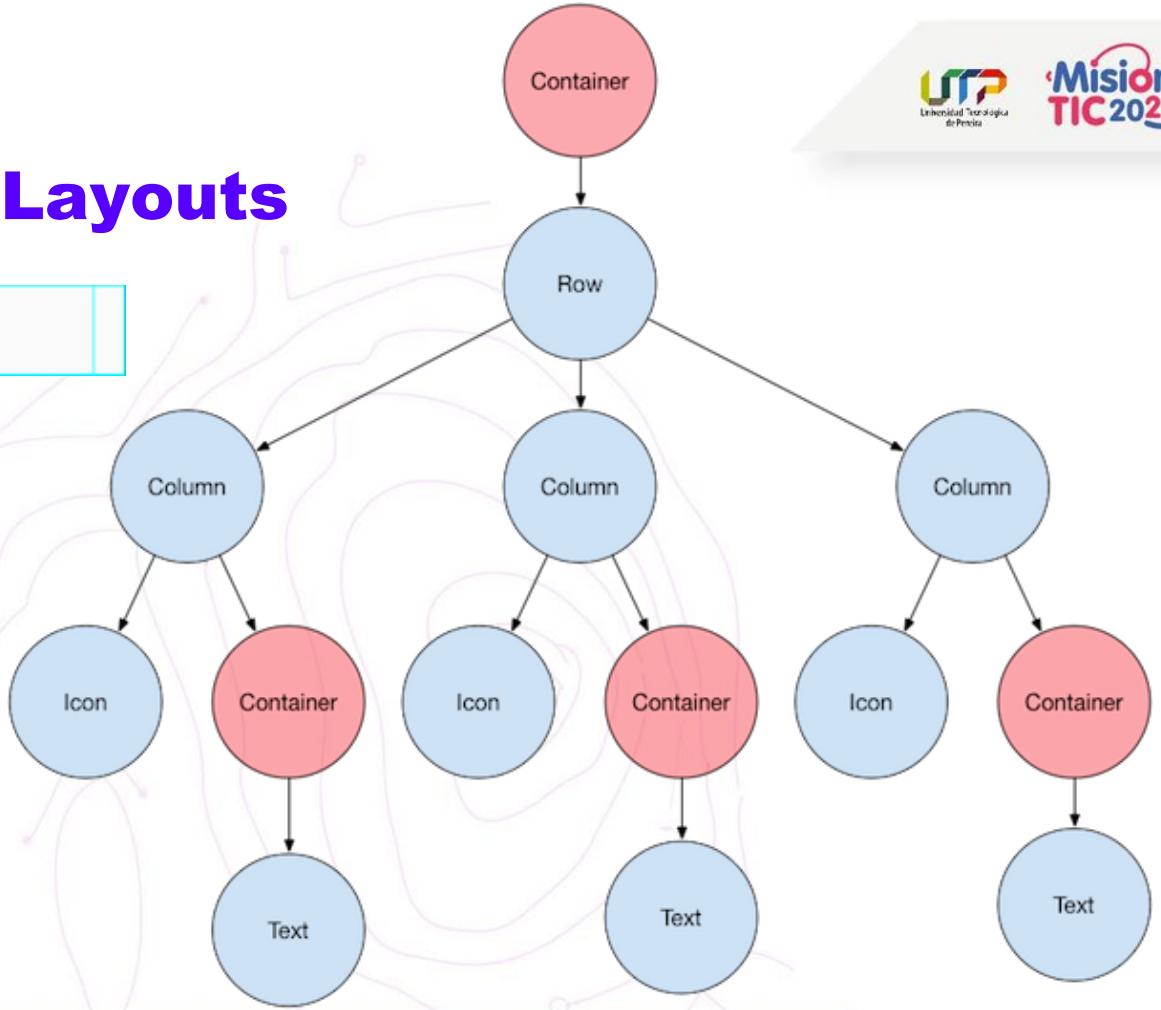
- **MaterialApp**: Un widget de conveniencia que envuelve una cantidad de widgets que comúnmente se requieren para las aplicaciones.
- **Scaffold**: Estructura de diseño visual básica de Material Design.
- **Appbar**: Barra de herramientas y potencialmente otros widgets.
- **BottomNavigationBar**: Facilitan explorar y cambiar entre las vistas de nivel superior en un solo toque.
- **TabBar**: Muestra una fila horizontal de pestañas.
- **TabBarView**: Una vista de página que muestra el widget que corresponde a la pestaña seleccionada actualmente.
- **Drawer**: Mostrar enlaces de navegación en una aplicación.



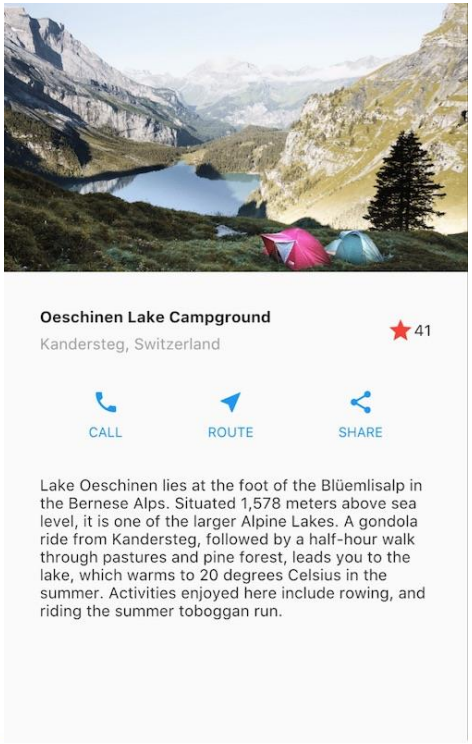


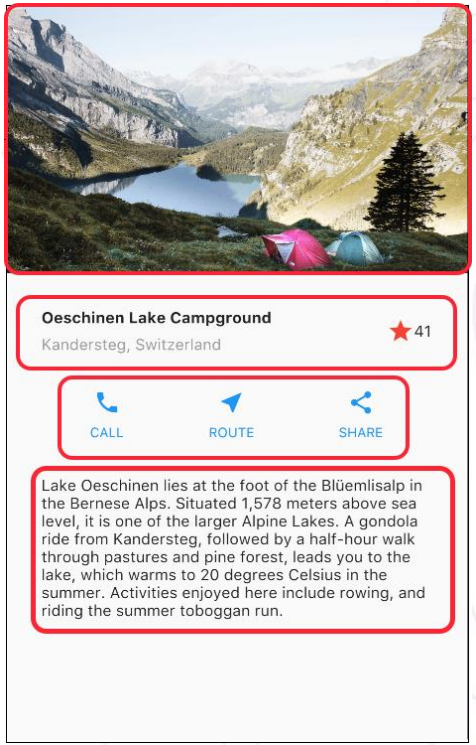
https://flutter-es.firebaseapp.com/docs/development/ui/widgets/material

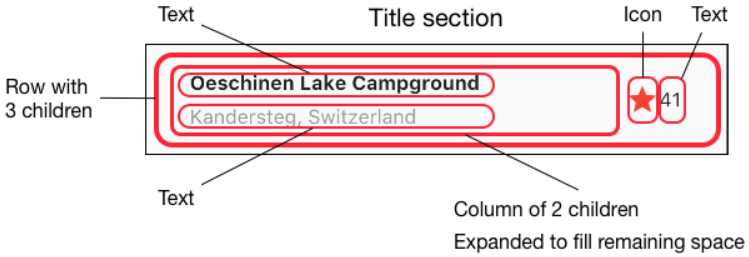
Construcción de Layouts

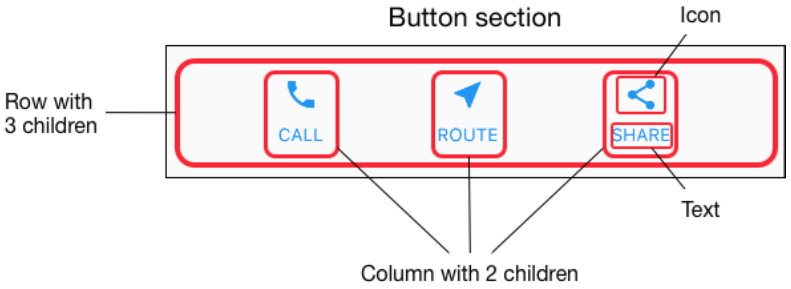


Construyendo una vista

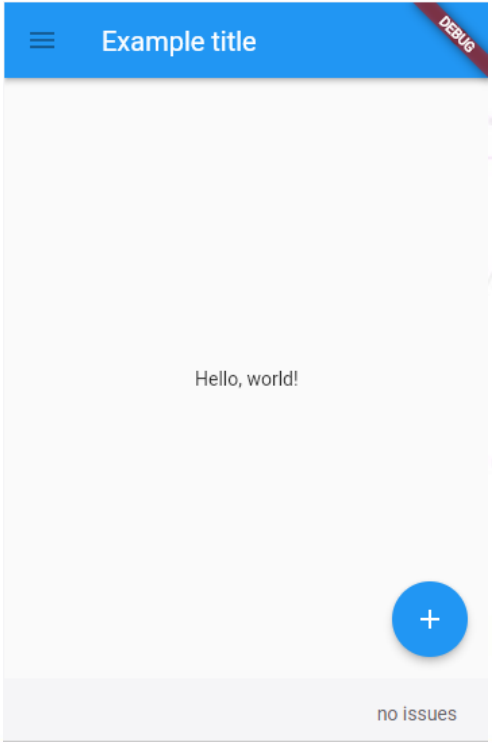








Mi primer App

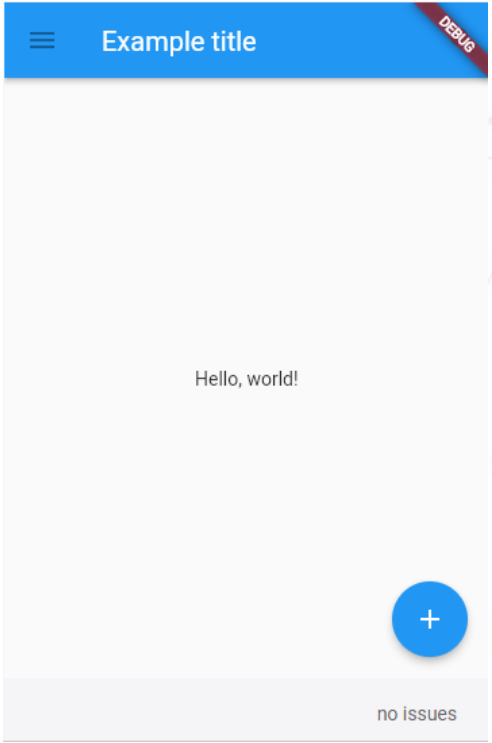



```
import 'package:flutter/material.dart';

void main() {
  runApp(
    const MaterialApp(
      title: 'Flutter Tutorial',
      home: TutorialHome(),
    ),
  );
}

. . .
```

Mi primer App




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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        leading: const IconButton(
          icon: Icon(Icons.menu),
          tooltip: 'Navigation menu',
          onPressed: null,
        ),
        title: const Text('Example title'),
        actions: const [
          IconButton(
            icon: Icon(Icons.search),
            tooltip: 'Search',
            onPressed: null,
          ),
        ],
      ),
      // body is the majority of the screen.
      body: const Center(
        child: Text('Hello, world!'),
      ),
      floatingActionButton: const FloatingActionButton(
        tooltip: 'Add', // used by assistive technologies
        onPressed: null,
        child: Icon(Icons.add),
      ),
    );
  }
}
```

Widgets Stateful y stateless

Un widget puede ser **stateful** o **stateless**.

Si un widget cambia (por ejemplo, cuando el usuario interactúa con él) es **stateful**.

Icon, **IconButton**, y **Text** son ejemplos de widgets stateless. Los widgets stateless heredan de la clase **StatelessWidget**.

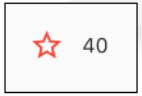
Un widget stateful es dinámico: por ejemplo, este puede cambiar su apariencia en respuesta a eventos desencadenados por la interacción del usuario o cuando recibe datos.

Checkbox, **Radio**, **Slider**, **InkWell**, **Form** y **TextField** son ejemplos de widgets **stateful**. Los widgets stateful heredan de la clase **StatefulWidget**.

El estado de un widget se almacena en un objeto **State**, separando el estado del widget de su apariencia. El estado de un widget consiste en valores que pueden cambiar, como el valor actual de un slider o cuando un checkbox está marcado como checked. Cuando el estado de un widget cambia, el objeto state llama a **setState(),** diciendo al framework que repinte el widget.

Widgets con estado

Favorited



Not favorited

```
class FavoriteWidget extends StatefulWidget {
  @override
  _FavoriteWidgetState createState() =>
_FavoriteWidgetState();
}

class _FavoriteWidgetState extends State<FavoriteWidget>
  bool _isFavorited = true;
  int _favoriteCount = 41;

  void _toggleFavorite() {
    setState(() {
      _favoriteCount = _isFavorited ?
                  _favoriteCount - 1 : _favoriteCount + 1;
      _isFavorited = !_isFavorited;
    });
  }
  . . .
```

```
@override
Widget build(BuildContext context) {
  return Row(
    mainAxisSize: MainAxisSize.min,
    children: [
      Container(
        padding: EdgeInsets.all(0),
        child: IconButton(
          icon: (_isFavorited ? Icon(Icons.star) : Icon(Icons.star_border)),
          color: Colors.red[500],
          onPressed: _toggleFavorite,
        ),
      ),
      SizedBox(
        width: 18,
        child: Container(
          child: Text('$_favoriteCount'),
        ),
      ),
    ],
  );
}
}
```