Flutter Bootcamp

Introducción a Flutter y Dart

Emanuel López – emanuel.lopez@globant.com

Globant

Que es Flutter?

Flutter es un SDK de código abierto creado por Google para construir **aplicaciones nativas de alta calidad para múltiples plataformas** (iOS, Android, web y escritorio) utilizando un **único código base**.

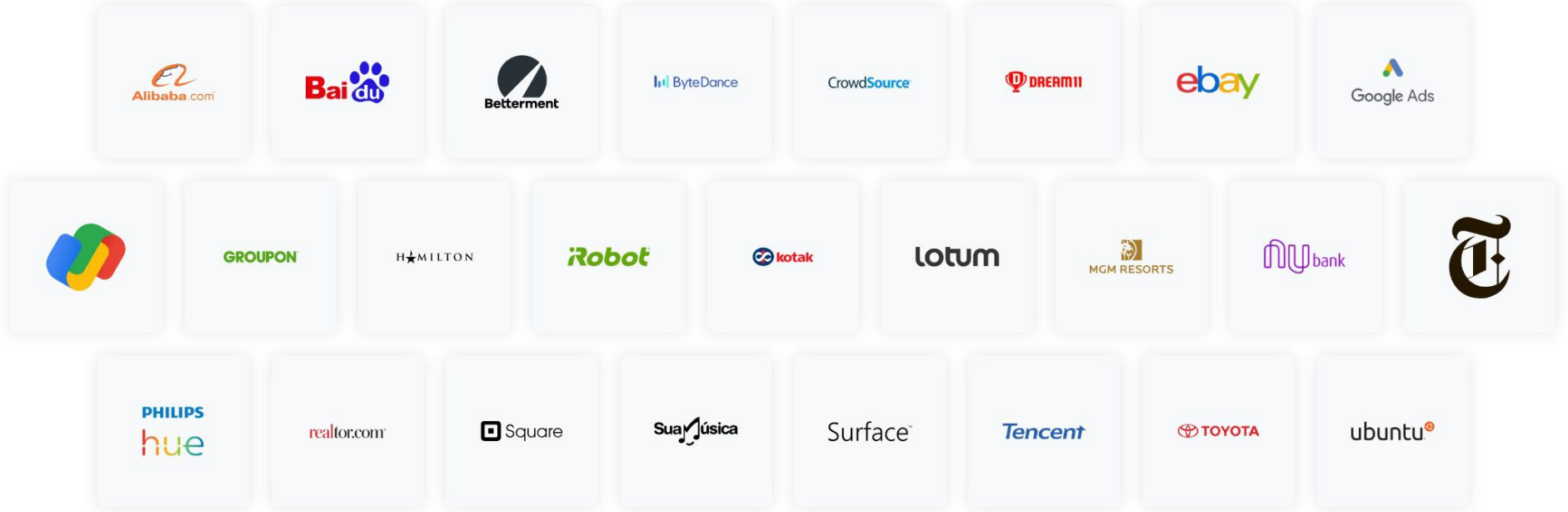
Utiliza el lenguaje de programación Dart y se destaca por su capacidad de renderizar interfaces de usuario de manera rápida y eficiente, lo que permite que las aplicaciones se vean y se sientan nativas en cada plataforma.

El componente básico de un programa en Flutter es el "**widget**", que a su vez puede constar de otros widgets. Un widget describe la lógica, la interacción y el diseño de un elemento de la interfaz de usuario.

Ventajas de desarrollar con Flutter

- **Desarrollo multiplataforma.** Un único código. Múltiples plataformas.
- **Alta velocidad de desarrollo.** Hot reload y Hot restart.
- **Experiencia de usuario consistente.** Widgets personalizables.
- **Alto rendimiento.** Compilación a código nativo.
- **Fuerte soporte de la comunidad y ecosistema.** Amplio ecosistema de plugins. Comunidad activa.
- **Desarrollo para web y escritorio.** Expansión más allá del móvil.
- **Integración con código nativo.** Accesso a APIs nativas
- **Actualizaciones y soporte continuo.** Respaldado por Google.
- **Diseño personalizado.** Independencia de la plataforma.
- **Reducción de costos.** Menor costo de desarrollo
- **Open Source.** Podemos monitorear su evolución.

Quién está usando Flutter?



Fuente: flutter.dev/showcase

Globant

Flutter – SDK y herramientas

Para desarrollar aplicaciones Flutter en Windows necesitamos las siguientes herramientas:

- **Git**
- **Android Studio.** Instalar el plugin **Flutter para IntelliJ**
- **Visual Studio Code.** Instalar la extensión **Flutter** para VSCode
- **Flutter SDK.**
- Opcional: **Bruno** (API client. Alternativa a Postman): usebruno.com

Extensiones VSCode recomendadas:

- Error Lens,
- GitLens – Git supercharged,
- Sort lines

Get Started: docs.flutter.dev/get-started/install/windows/mobile

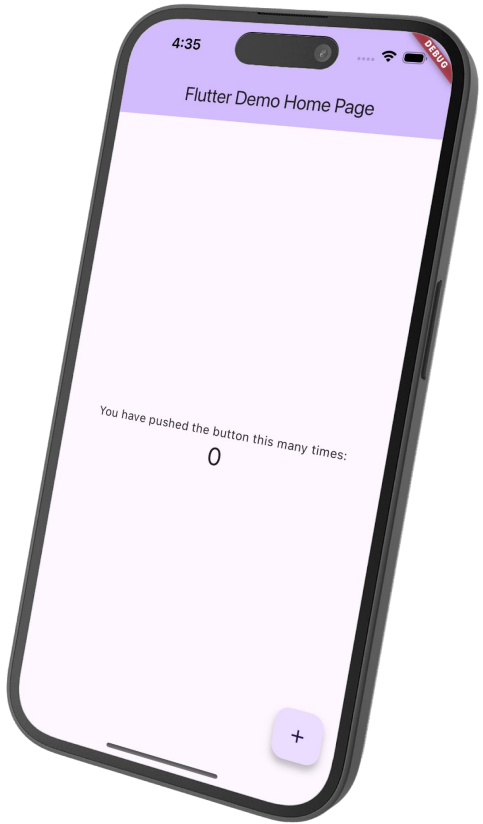
Globant

Creando una app en Flutter

```
$ flutter create bootcamp
$ cd bootcamp
$ flutter run
```

Algunos parámetros opcionales:

```
flutter create my_app  --org com.globant.bootcamp
                       --template {app|module|package|plugin}
                       --platforms=android,ios,web
```



Write your first Flutter app codelab: docs.flutter.dev/get-started/codelab

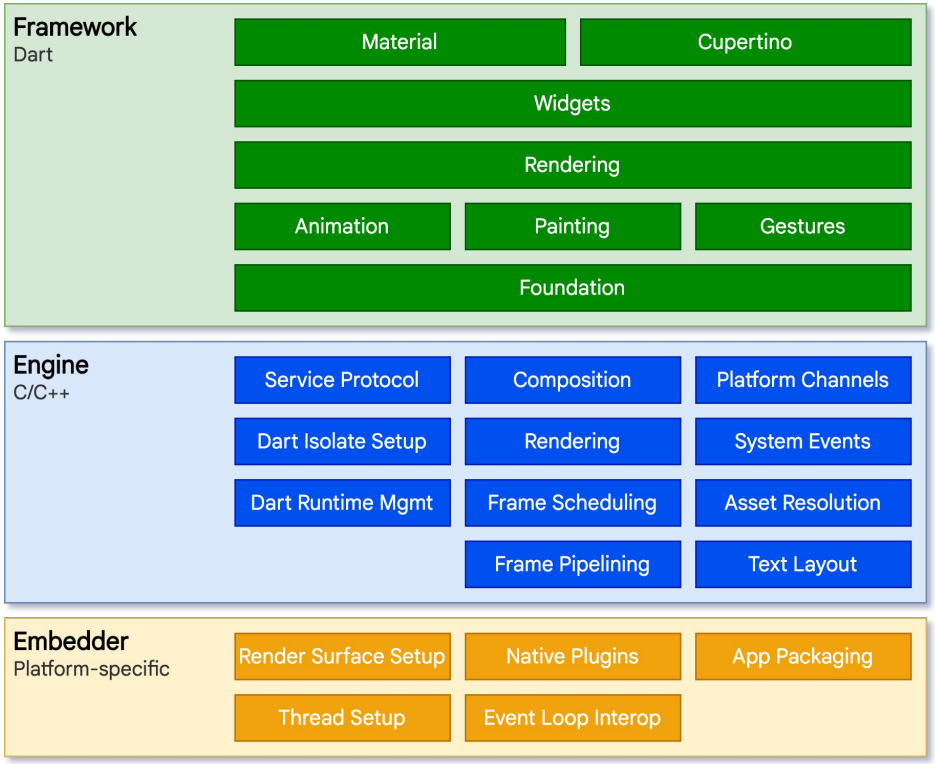
Globant

Arquitectura de Flutter

Flutter tiene una **arquitectura modular y en capas.**

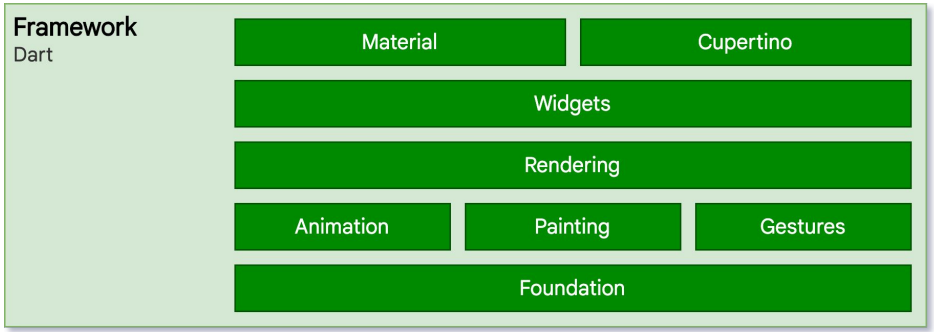
Esto nos permite escribir la lógica de la aplicación una vez y obtener un **comportamiento consistente en todas las plataformas.**

Ninguna capa tiene acceso privilegiado a la capa inferior, y cada parte del nivel del framework está diseñada para ser opcional y reemplazable.



Arquitectura – Framework

Contiene las librerías de alto nivel que los desarrolladores usan directamente para construir aplicaciones.

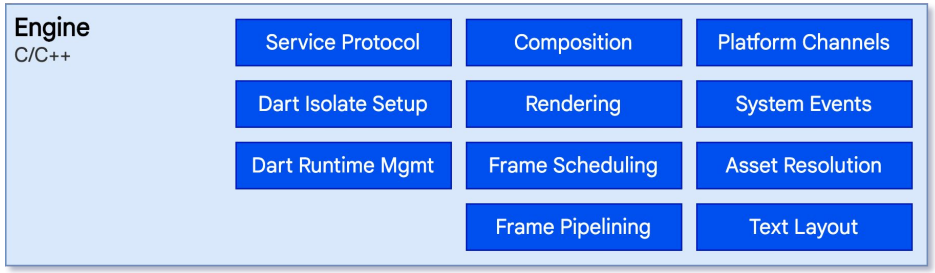


Componentes principales:

- **Widgets:** Componentes para construir interfaces de usuario
- **Gestos y animaciones:** Interacción de usuario
- **Diseño (layout):** Define cómo los widgets se muestran en pantalla.
- **Temas:** Personalización de colores, tipografías, etc de la aplicación.
- **Plugins:** Acceso a características específicas de la plataforma (como acceso a la cámara o geolocalización)

Arquitectura – Engine

Esta capa contiene librerías de bajo nivel responsables de las funciones críticas que hacen que Flutter funcione en diferentes plataformas.



Componentes principales:

- **Gráficos y Renderizado:** El motor dibuja las escenas de la aplicación y las convierte en píxeles que se muestran en la pantalla, utilizando herramientas gráficas como Skia.
- **Entrada/Salida (I/O):** Maneja la entrada y salida de datos, como la lectura de archivos o la comunicación con APIs externas.
- **Disposición de texto y accesibilidad:** Gestiona cómo se muestra el texto y asegura que las aplicaciones sean accesibles para personas con discapacidades.
- **Arquitectura de plugins y runtime de Dart:** Integra los plugins y ejecuta el código Dart, permitiendo que la lógica de la aplicación se ejecute correctamente.

Arquitectura – Embedder

El Embebedor es diferente para cada sistema operativo objetivo (Android, iOS, Windows, macOS, Linux, o web).

Funcionalidad principal:

- **Empaquetado:** Empaqueta la aplicación Flutter como una aplicación independiente o módulo embebido, dependiendo de la plataforma.
- **Interfaz con la plataforma:** Maneja la comunicación entre la aplicación Flutter y los servicios específicos del sistema operativo subyacente, como acceso al hardware o gestión del ciclo de vida de la aplicación.

Mas información en: docs.flutter.dev/resources/architectural-overview

Dart

Dart es el lenguaje de programación que se utiliza para desarrollar aplicaciones en Flutter.

- **Orientado a objetos.** Con soporte para programación funcional.
- **Compilación fléxible.** *Just-in-Time* durante el desarrollo y *Ahead-of-Time* para la versión final en nativo.
- **Sintáxis clara y familiar.** Similar a Javascript, Java y C#. Facilita su aprendizaje.
- **Soporte para tipado estático y dinámico.**
- **Null safety.** Para evitar `NullPointerExceptions`

Dart – Tipo de datos

- Numbers (`int, double`)
- Strings (`String`)
- Booleans (`bool`)
- Records (`(value1, value2)`)
- Lists (`List`, also known as `arrays`)
- Sets (`Set`)
- Maps (`Map`)
- Runes (Runes; often replaced by the characters API)
- Symbols (Symbol)
- The value `null` (`Null`)

Dart – Hello World!

```
/// Creación de un proyecto dart
$ dart create dart_basics


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

bit.ly/DartBasics

Globant

Dart – Sintáxis

```
final name = 'Bob';
final String name = 'Bob';
var name = 'Bob';
Object name = 'Bob';
String name = 'Bob';
var year = 1977;
var pi = 3.14159265359;
```

```
var planets = [
    'Jupiter','Saturn',
    'Uranus','Neptune'
];
var colors = {
    'red', 'green',
    'blue', 'Neptune'
};
var image = {
    'tags': ['saturn'],
    'url': '//path/to/saturn.jpg'
};
```

Globant

Dart – Interpolación de Strings

La interpolación de strings en Dart te permite insertar variables o expresiones dentro de cadenas de texto de manera sencilla y legible.

En lugar de concatenar cadenas manualmente usando el operador +, la interpolación te ofrece una forma más limpia de hacerlo utilizando el símbolo de dólar ($).

Ventajas de la interpolación de strings en Dart

- **Legibilidad**: Hace que el código sea más fácil de leer y entender, ya que elimina la necesidad de concatenaciones con +.
- **Flexibilidad**: Permite insertar tanto variables simples como expresiones complejas dentro de una cadena de texto.
- **Menos errores**: Al evitar la concatenación manual, reduces el riesgo de cometer errores tipográficos o de concatenación incorrecta.

Dart – Null safety

Null Safety es una característica que previene errores causados por el acceso a variables que tienen un valor `null` cuando no deberían. Estos errores, en otros lenguajes, son comunes y difíciles de detectar en desarrollo.

Beneficios:

1. **Prevención de errores en tiempo de ejecución.**
    - Sin null safety, un error común es que una variable que esperas que tenga un valor tenga en cambio `null`. Esto puede provocar errores en tiempo de ejecución (runtime errors) como el famoso `"NullPointerException"`.
    - Con null safety, Dart evita estos errores a través de análisis estáticos en tiempo de compilación. Si una variable puede ser nula, Dart te obligará a manejar ese caso de manera explícita.

Dart – Null safety

Beneficios (cont.)

2. **Mejor rendimiento y optimizaciones**:
    - Al saber que una variable nunca será `null`, el compilador de Dart puede optimizar el código para hacerlo más eficiente, generando binarios más pequeños y mejorando el rendimiento.

```
int? nullableVar;           // Puede ser nulo
nullableVar = null;         // Esto es válido

int nonNullableVar = 42;    // No puede ser nulo
nonNullableVar = null;      // Error en tiempo de compilación
```

Dart – Null safety. Operadores

Dart nos provee varios operadores que te ayudan a manejar valores nulos de manera segura y eficiente:

- Operador de acceso condicional (`?.`)
- Operador de asignación condicional (`??=`)
- Operador de fusión con null (`??`)
- Operador de null assertion (`!`)
- Operador de cascada condicional (`?..`)
- Operador de indexado seguro (`?[ ]`)
- Operador de propagación condicional (`...?`)

Dart – Funciones

Dart es un lenguaje orientado a objetos, eso significa que hasta las funciones son objetos y tienen un tipo. Por lo tanto, las funciones pueden ser asignadas a variables o pasadas como argumento a otras funciones.

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

Arrow syntax:

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

Dart – Funciones. Parámetros

Una función puede tener cualquier cantidad de parámetros posicionales obligatorios. Estos pueden ir seguidos de parámetros nombrados (`named parameters`) o parámetros posicionales opcionales (`optional positional parameters`), pero no ambos a la vez.

```
void enableFlags(bool bold, bool hidden) {...}

void enableFlags(bool bold, [bool? hidden]) {...}

void enableFlags(bool bold, {bool? hidden}) {...}

void enableFlags(bool bold, {required bool hidden}) {...}

void enableFlags(bool bold, {bool hidden = true}) {...}
```

Dart – Control de flujo

En Dart existen varias estructuras de control de flujo para controlar el comportamiento de nuestro programa:

- Estructurra `if` y `else`
- Estructura `else if`
- Estructura `switch`
- Bucles `for`
- Bucle `while`
- Bucle `do-while`
- Estructura `break`
- Estructura `continue`
- Estructura `try-catch-finally`
- Expresión `assert`
- Expresión `throw`

Dart – Future, async, await

Future: Un futuro representa el resultado de una operación asincrónica que eventualmente se completará con un valor o un error. En este ejemplo, el tipo de retorno de `Future<String>` representa una promesa de proporcionar eventualmente un `String` (o un error).

```
Future<String> readFileContent(String filename) {...}

readFileContent('file.txt').then((content){print(content);});
```

async – await: Las palabras clave async y await proporcionan una forma declarativa de definir funciones asincrónicas y utilizar sus resultados.A continuación, la ejecución se bloquea mientras espera la entrada/salida de un archivo:

```
final content = await readFileContent('file.txt');

print(content);
```

Globant

Dart – Clases y Herencia

```
class Person {
  Person({
    required this.firstName,
    required this.lastName,
    this.age = 0,
  });

  final String firstName;
  final String lastName;
  final int age;
  ...

}
```

```
class Employee extends Person {
  Employee({
    required super.firstName,
    required super.lastName,
    super.age,
    required this.salary,
  });

  final double salary;
  ...

}
```

Dart – Mixins

Los `mixins` en Dart son una forma de compartir funcionalidad entre varias clases sin utilizar herencia directa. Un mixin es un conjunto de métodos y propiedades que puedes agregar a una clase.

A diferencia de la herencia, donde una clase puede extender solo de una clase padre, con los mixins una clase puede "mezclar" múltiples mixins y heredar funcionalidades de ellos.

Cuándo usar mixins?

Usamos mixins cuando varias clases necesitan la misma funcionalidad, pero no tienen una relación de herencia. Los mixins te permiten evitar la duplicación de código sin forzar una relación jerárquica entre las clases.

Dart – Extensiones

Una extensión es una forma de añadir nuevas funcionalidades a una clase existente. Podemos definir métodos adicionales, getters y setters en una extensión, y estos métodos estarán disponibles en las instancias de la clase a la que extienden.

```
extension PalindromeExtension on String {
  bool get isPalindrome {
    String reversed = this.split('').reversed.join('');
    return this == reversed;
  }
}

print('radar'.isPalindrome); // Prints: true
print('hello'.isPalindrome); // Prints: false
```

Links adicionales

- Get Started: docs.flutter.dev/get-started/install/windows/mobile
- Introduction to Dart: dart.dev/language
- Ejemplos Dart Basics: bit.ly/DartBasics
- Repositorio del Bootcamp: bit.ly/FlutterBootcampGlobant
- Ejercicios Dart:
  - exercism.org/tracks/dart/exercises
  - github.com/bizz84/dart-course-materials
  - hackmd.io/@kuzmapetrovich/S1x90jWGP

Globant

Questions?

Globant

Thanks!

Globant