FLUTTER / DART

PARADIGMA REACTIVO

Equipo 4
Padilla Valencia Emanuel
Martinez Valdez Clara Andrea
Armas Diaz Erick Hidekio

FUNDAMENTOS

Dart

Dart es un lenguaje de programación de propósito general, de código abierto y orientado a objetos (maneja clases, objetos y herencia), desarrollado por Google, optimizado para crear aplicaciones rápidas en multiples plataformas (móviles, web, escritorio).

Flutter

Flutter es un framework de Google que permite crear aplicaciones multiplataforma **usando el lenguaje Dart** y un motor gráfico propio apoyándose en el SDK de Flutter, los SDK de cada plataforma y librerías adicionales para desarrollar apps de forma rápida y eficiente.

PARADIGMA REACTIVO

La programación reactiva es un paradigma de programación declarativa que se basa en la idea de procesamiento asincrónico de eventos y flujos de datos.

En la Programación Reactiva, el sistema reacciona automáticamente a los cambios de datos/estado
Principio fundamental:
Cuando los DATOS cambian → La UI se ACTUALIZA automáticamente

PARADIGMA REACTIVO EN FLUTTER

- Streams de eventos (Flujos): Secuencias de datos que llegan a lo largo del tiempo.
- Propagación automática: Cuando el dato cambia, todos los elementos que lo "escuchan" se actualizan solos.
- Observabilidad: Los componentes están en constante espera de nuevos eventos o estados.

El Hot Reload es su mayor fuerte; permite ver cambios en el código en menos de un segundo sin perder el estado de la app.

CARACTERÍSTICAS GENERALES

DART - Lenguaje:

- Orientado a objetos e imperativo
- Tipado estático fuerte
- Null Safety (evita errores de valores nulos)
- Programación asíncrona (async/await)
- Compilación JIT (desarrollo) y AOT (producción)
- Función main() como punto de entrada

FLUTTER - Framework:

- UI Declarativa
- Todo es un widget
- Multiplataforma
- Motor de renderizado propio (Skia/Impeller)
- Conceptos Combinados:
- Paradigma Mixto: Dart imperativo + Flutter declarativo
- Asincronía: Procesos en segundo plano sin bloquear UI

DART

```dart
void incrementar() {
  setState(() {
    contador++;
    if (contador > 10) {
      contador = 0;
    }
  });
}
```

FLUTTER (usando DART)

```dart
Widget build(BuildContext context) {
  return Column(
    children: [
      Text('Contador: $contador'),
      ElevatedButton(
        onPressed: incrementar,
        child: Text('Incrementar'),
      ),
    ],
  );
}
```

DISCPLINA DE TIPOS

Tipado Estático

Dart verifica los tipos durante la compilación. Si se declara un int de 64 bits, el compilador no permitirá asignarle un String

Tipado Fuerte

El lenguaje es estricto con las reglas de tipos. Por ejemplo:
Un int siempre ocupará sus 8 bytes y no puede ser usado donde se espera un bool (Booleano).
No puedes realizar operaciones matemáticas entre tipos incompatibles (como sumar un número a un texto)

Inferencia de Tipos

Dart puede inferir qué tipo ded variable es, por ejemplo:
var edad = 25;
Dart infiere automáticamente que es un int de 64 bits.

LANZAMIENTO

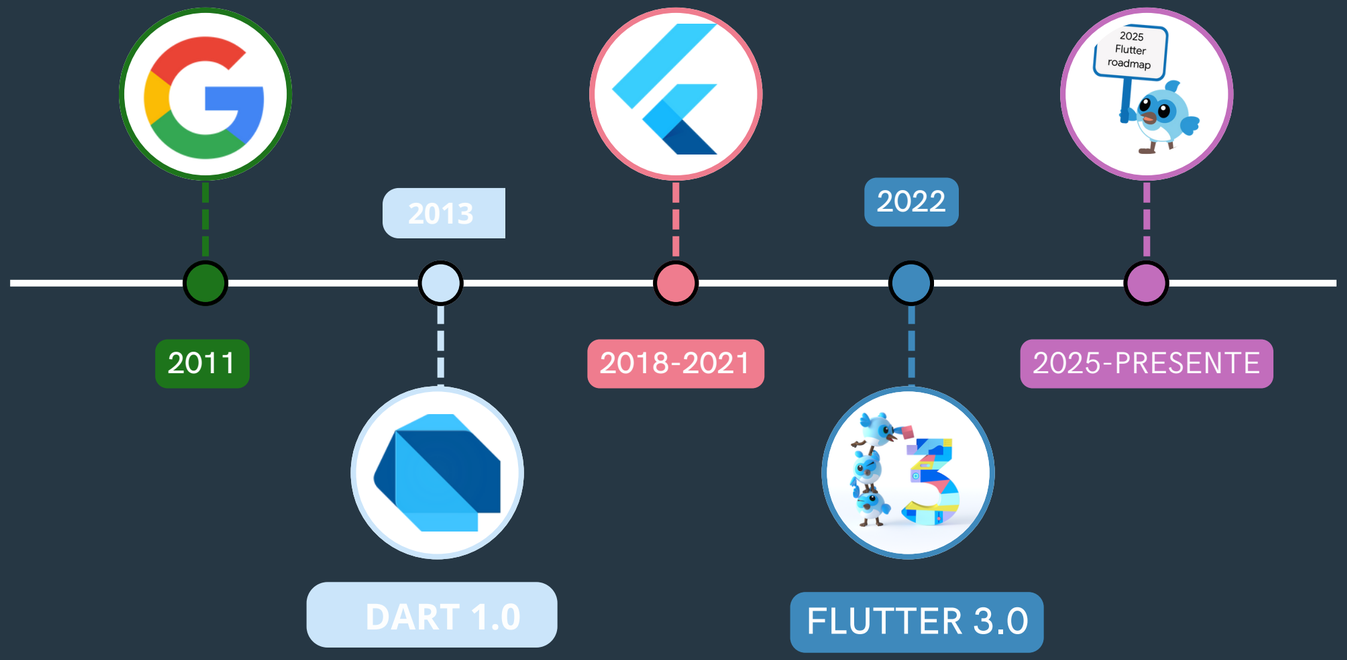
Google presenta Dart, con el propósito de reemplazar a JavaScript

CRECIMIENTO

Se lanza Flutter 1.0. Comienza el soporte para Android y iOS. Incorpora el tipado fuerte y el null safety, crece en popularidad alzanzando 500,000 apps publicadas y 4 millones de desarrolladores

CONSOLIDACIÓN

Flutter es un líder indiscutible en desarrollo multiplataforma Competencia con React Native, .NET MAUI, Kotlin Multiplatform



Se lanza oficialmente la primera versión, pero JavaScript sigue siendo la opción popular

Alcanza soporte estable para las 6 plataformas: iOS, Android, Web, Windows, macOS, Linux. Consigue 5 millones de desarrolladores en Git

TIPOS DE DATOS EN DART

Objeto

num (Supertipo Numérico)
Descripción: Tipo padre de int y double
Uso: Cuando una variable puede ser entero o decimal
Tamaño: 64 bits

bool (Booleano)
Descripción: Valor lógico verdadero o falso
Tamaño: 1 bit de información (pero ocupa más en memoria por alineación)
Valores: Solo true o false

String (Cadena de texto)
Descripción: Secuencia de caracteres
Codificación: UTF-16
Tamaño: Variable (depende de la longitud)
Memoria: ~2 bytes por carácter + overhead

int (Entero)
Descripción: Números enteros sin decimales
Tamaño: 64 bits (8 bytes)
Rango: -2^{63} a $2^{63}-1$
Mínimo: -9,223,372,036,854,775,808
Máximo: 9,223,372,036,854,775,807

double (Punto Flotante)
Descripción: Números con decimales
Tamaño: 64 bits (8 bytes)
Estándar: IEEE 7
precisión: ~15-17 dígitos decimales
Rango: Aproximadamente $\pm 1.7976931348623157 \times 10^{308}$

int (num)	42
double (num)	3.14159
bool	true/false
String	‘Hello’ + “World”

Tipos Compuestos (Colecciones)

1. List (Listas)
Es la colección más común. Es un grupo ordenado de objetos donde cada elemento tiene un índice (empezando desde 0).

```
List<String> frutas = ['Manzana', 'Pera', 'Manzana'];
```

2. Set (Conjuntos)
Es una colección de elementos donde cada objeto debe ser único. No permite duplicados.

- Uso: Cuando necesitas asegurar que no haya elementos repetidos

```
Set<String> invitados = {'Juan', 'Maria', 'Pedro', 'Juan'};
```

3. Map (Mapas / Diccionarios)
Es una colección de pares llave-valor. Cada llave es única y se usa para acceder a su valor correspondiente.

- Uso: Para representar datos estructurados, como la información de un usuario.

```
Map<String, dynamic> usuario = {
  'nombre': 'Juan',
  'edad': 25,
  'esPremium': true
};
```

4.Tipos Especiales
dynamic (Tipo Dinámico)
Descripción: Variable que puede ser de cualquier tipo
Type checking: Deshabilitado en tiempo de compilación
Verificación: En tiempo de ejecución
Cuándo usar: Cuando el tipo es realmente desconocido (raro)
Riesgo: Errores en tiempo de ejecución

```
dynamic variableLoca = 'Hola';
variableLoca = 10;     // No hay error
variableLoca = true;   // Sigue sin haber error
```

5.Null (Ausencia de valor)
Descripción: Representa ausencia de valor
Null Safety: Sistema para prevenir errores de null
Tipos Nullable: Se marcan con ?
Tipos Non-nullable: Por defecto, no aceptan null
Ejemplos:

```
String nombre = "Gemini"; // No puede ser null
String? apellido;         // SÍ puede ser null (empieza siendo null)
```

Estructuras de Control

CICLOS

For

```
for (int i = 0; i < 5; i++) {
  print(i);
}
```

For-in

```
for (var item in lista) {
  print(item);
}
```

While

```
while (condicion) {
  // código
}
```

Do-While

```
do {
  // código
} while (condicion);
```

CONDICIONALES

if

```
if (condicion) {
 // código si es verdadero
}
else if (otraCondicion) {
// código si la segunda es
verdadera
}
else {
// código si ninguna se
cumple
}
```

Switch Case

```
switch (variable) {
  case valor1:
    // código
    break;
  default:
    // código por defecto
}
```

VENTAJAS

- Multiplataforma Real: Un solo código para iOS, Android, Web y Escritorio.
- UI Consistente: Al usar su propio motor (Impeller es ahora el estándar estable), las animaciones son fluidas y la app se ve idéntica en cualquier dispositivo.
- Respaldo y Actualizaciones: Google mantiene un ciclo de actualizaciones muy activo.
- Curva de Aprendizaje: Si sabes Java, C# o JavaScript, aprender Dart es muy rápido

DESVENTAJAS

- Peso de la App: Las apps de Flutter son 4-7MB más pesadas que las nativas puras.
- Consumo de Memoria: Al incluir su propio motor gráfico y entorno de ejecución (runtime), el consumo de RAM es más elevado que en apps nativas.
- Ecosistema muy específico: Aunque pub.dev es muy bueno y organizado, se queda corto frente al ecosistema de JavaScript
- Aislamiento de Dart: Fuera de Flutter, Dart tiene muy poco uso profesional. No es un lenguaje “general” como JavaScript o Python.

Uso ideal y limitaciones

- Desarrollo multiplataforma con una sola base de código para Android, iOS, Web y Escritorio
- Proyectos con poco tiempo o presupuesto que necesitan salir rápido al mercado
- Creación de MVPs para validar una idea en varias plataformas al mismo tiempo
- Equipos que buscan reducir costos de desarrollo y mantenimiento
- Mantiene consistencia visual y funcional entre plataformas
- Acelera el desarrollo y simplifica el mantenimiento

Usos principales

- Aplicaciones multiplataforma
- Interfaces dinámicas con cambios frecuentes como redes sociales o chats
- Dashboards para visualización de métricas en tiempo real (ejemplo: pantalla de terminales de aeropuertos)

Limitaciones de Dart / Flutter

- Ecosistema más limitado frente a Java, Python o C++
- Menor disponibilidad de librerías y herramientas especializadas
- Integración más compleja con sistemas legacy
- No ideal para aplicaciones de una sola plataforma
- No recomendado para alto rendimiento crítico como juegos 3D complejos o procesamiento intensivo en tiempo real

Característica	Flutter / Dart (Reactivo)	Java (Orientado a Objetos)	Python (Funcional/Scripting)
Paradigma Principal	**Reactivo / Declarativo**: La UI reacciona a cambios de estado.	**Orientado a Objetos (POO)**: Basado estrictamente en clases y objetos.	**Multiparadigma (énfasis Funcional)**: Destaca por su flexibilidad y funciones.
Unidad de Código	**Widget**: Todo es un componente visual que se redibuja.	**Clase**: El plano que define datos (atributos) y acciones (métodos).	**Función / Script**: Bloques que transforman datos sin estados complejos.
Manejo de Datos	**Flujos (Streams)**: Datos que llegan de forma asíncrona y continua.	**Estado del Objeto**: Datos encapsulados que mutan dentro de la instancia.	**Inmutabilidad**: Preferencia por no cambiar datos (List Comprehensions, Map, Filter).
Sintaxis / Estilo	Declarativo (Dices *qué* quieres ver en pantalla).	Imperativo (Dices *cómo* realizar cada paso secuencial).	Conciso y legible (Casi pseudocódigo matemático).
Uso Ideal	Interfaces de usuario fluidas y aplicaciones móviles.	Sistemas empresariales masivos y aplicaciones Android nativas.	Ciencia de datos, IA, scripts rápidos y automatización.


Flutter / Dart
y el Paradigma Reactivo
Definiciones Principales
Flutter: Framework de Google que usa Dart y un motor gráfico propio para desarrollar apps rápidas y eficientes
Dart: Lenguaje de propósito general, orientado a objetos, desarrollado por Google y optimizado para aplicaciones multiplataforma.
Paradigma Reactivo: Programación declarativa basada en flujos de datos y procesamiento asincrónico de eventos
Estructuras de control
Ciclos
For In
While
For
Do-While
Condicionales
If
Swicth Case
Else If
Componentes del Paradigma (Cómo funciona)
Streams (Flujos): Secuencias de datos que llegan a lo largo del tiempo.
Principio fundamental: Cuando los DATOS cambian, la UI se ACTUALIZA automáticamente.
Hot Reload: Permite ver cambios en el código en menos de un segundo sin perder el estado.
Observabilidad: Componentes en espera constante de nuevos eventos o estados.
Propagación automática: Los elementos que "escuchan" el dato se actualizan solos al detectar cambios
Características Dart
Disciplina de Tipos:
Inferencia de Tipos: Puede deducirlo automaticamente
Tipado Estatico: Verifica tipos durante la compilación.
Tipado Fuerte: Las operaciones entre tipos incompatibles NO son posibles
Tipos de Datos:
Numericos
Enteros
Double
Lógico yTexto
booleano
string
Uso Ideal y Limitaciones
Caso de uso
Aplicciones de Escritorio, Web y Multiplataforma
Interfaces dinámicas con cambios frecuentes. Ejemplo: pantallas con horarios de salida en aeropuertos y centrales de autobuses
Desventajas
Una app pequeña puede tener mayor peso.
Alto consumo de RAM.
Ventajas
Respaldado por Google
Multiplataforma Real: Un solo código para TODO.
UI Consistente y fluida al tener su propio motor gráfico (Skia/Impeller)
Curva de aprendizaje: rápido aprendizaje si se viene lenguajes como Java, c# o JS.
Línea de Tiempo y Evolución:
Presentado en 2011 y lanzado en 2013 con el propóstito de reemplazar a JavaScript, sin mucho exito hasta el 2017 cuando se incorpora con Flutter.
Al 2025, es de los lenguajes de programación para multiples plataformas más populares (del paradigma reactivo), con millones de desarrolladores.

GRACIAS

POR SU ATENCIÓN