Flutter Bootcamp

Widgets

Emanuel López – emanuel.lopez@globant.com

Globant

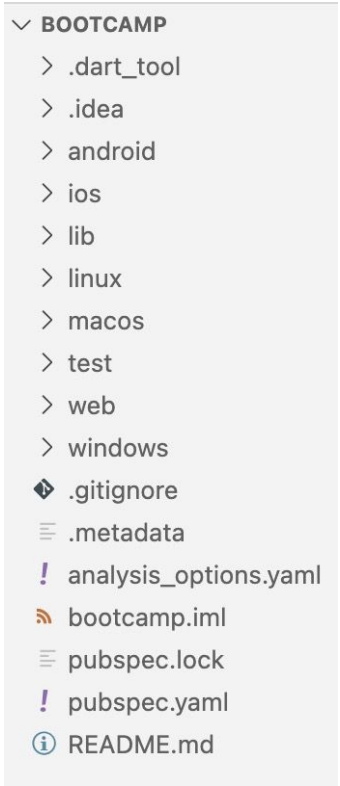
Repaso

- Flutter
  - Que es? Ventajas de programar con Flutter
  - Requerimientos para programar aplicaciones con Flutter
  - Arquitectura: Framework, Engine y Embedder
- Dart:
  - Características
  - Tipo de datos
  - Interpolación de Strings
  - Null Safety. Operadores
  - Funciones. Tipos de parámetros
  - Control de flujo
  - Future, async/await
  - Clases. Herencia
  - Mixins
  - Extensiones

Repositorio del Bootcamp: bit.ly/FlutterBootcampGlobant

Globant

Contenido de un proyecto Flutter



- **android / ios / linux / macos / web / windows:** Contiene el código nativo y configuraciones específicas para cada una de las plataformas.
- **lib:** Contiene el código fuente de la aplicación en Dart.
- **test:** Contiene los tests de la aplicación (unit tests y widget tests).
- **build:** Generado automáticamente. Contiene archivos temporales y binarios de la compilación. Excluido del control de versiones.
- **pubspec.yaml:** Archivo para la gestión de dependencias, assets y configuraciones del proyecto.
- **.gitignore:** Listado de archivos y directorios que deben ser ignorados por el sistema de control de versiones.
- **analysis_options.yaml.** Contiene las reglas para el análisis estático del código. Mas información: dart.dev/tools/analysis (ver ejemplos)
- **README.md:** Información descriptiva sobre el proyecto.

Globant

pubspec.yaml

- **name:** Nombre de la aplicación
- **description:** Descripción de la aplicación
- **version:** versionName+versionNumber (versión+build) ej: `1.1.0+73`
- **environment:** especifica la versión de Dart requerida por el proyecto

```
environment:
  sdk: ">=2.19.0 <3.0.0"
```

compatible con versiones de Dart entre 2.19.0 y 3.0.0.

- **dependencies:** paquetes necesita nuestro proyecto para funcionar

```
dependencies:
  flutter:
    sdk: flutter
  http: ^0.13.3
  provider: ^6.0.0
```

pubspec.yaml

- **dev_dependencies:** paquetes solo requeridas durante el desarrollo de nuestro proyecto
- **flutter:** assets y fuentes requeridas en el proyecto

```
flutter:
  uses-material-design: true
  assets:
    - assets/images/
    - assets/sounds/
  fonts:
    - family: Roboto
      fonts:
        - asset: fonts/Roboto-Regular.ttf
        - asset: fonts/Roboto-Bold.ttf
```

Mas información:

- dart.dev/tools/pub/pubspec
- docs.flutter.dev/tools/pubspec

Globant

Dependencias y caret notation ^

```
dependencies:
  ...
  http: ^0.13.3
  provider: ^1.2.0
```

`^0.13.3` permitirá usar cualquier versión desde 0.13.3 hasta antes de la versión `0.14.0`.

`^1.2.0`, aceptará versiones desde `1.2.0` hasta antes de `2.0.0`.

Beneficio: Permite beneficiarnos automáticamente de cualquier actualización sin que tengamos que modificar el archivo `pubspec.yaml` cada vez que haya una nueva versión de la dependencia.

También nos aseguramos de no usar versiones que podrían romper funcionalidades importantes debido a cambios mayores.

Mas información: dart.dev/tools/pub/dependencies#caret-syntax

Dependencias y caret notation ^

¿Qué pasa si no usas el ^?

Si no incluímos el caracter `^`, Flutter/Dart usará exactamente la versión que especifiquemos. Por ejemplo:

```
dependencies:
  ...
  http: 0.13.3
```

En este caso, se utilizará siempre la versión `0.13.3` de **http**. Si en algún momento aparece una versión `0.13.4` o `0.14.0`, nuestro proyecto no será actualizado automáticamente.

Directorio de paquetes para Dart y Flutter: pub.dev

Para agregar un paquete a nuestro proyecto: `flutter pub add http`

Widgets

Un widget en Flutter es el bloque fundamental de construcción de la interfaz de usuario.

Todo en Flutter, desde elementos visuales como botones, texto e imágenes, hasta la estructura de la interfaz como columnas, filas y contenedores, está compuesto por widgets.

Un widget describe una parte de la interfaz de usuario de una aplicación, define su apariencia, comportamiento y disposición.

Cuando se construye la interfaz de una aplicación, se crea un árbol de widgets donde cada widget es un nodo que puede tener uno o más hijos.

En Flutter, hay 2 tipos principales de widgets: **StatelessWidget** y **StatefulWidget**

StatelessWidget

Es un widget **inmutable**. No puede cambiar una vez que ha sido construido.

No tiene estado interno que pueda modificarse después de su creación.

Si necesita actualizarse, debe construirse nuevamente desde cero.

Los datos que utiliza se proveen de manera externa (ej: por parámetro)

Ejemplos: widgets simples como botones, textos estáticos, iconos, etc

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

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

  @override
  Widget build(BuildContext context) {
    return const Placeholder();
  }
}
```

StatefulWidget

Es un widget **que tiene estado**. Puede cambiar durante el tiempo de vida de la aplicación.

Los cambios en su estado hacen que el widget se vuelva a construir.

Tienen una clase de estado asociada (`State`) donde se define la lógica y las variables que pueden cambiar con el tiempo.

Cuando cambia el estado, se vuelve a ejecutar el método build para reconstruir el widget con su nuevo estado.

Ejemplos: formularios, controles interactivos, animaciones, etc.

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

class MyWidget extends StatefulWidget {
  const MyWidget({super.key});

  @override
  State<MyWidget> createState() => _MyWidgetState();
}

class _MyWidgetState extends State<MyWidget> {
  @override
  Widget build(BuildContext context) {
    return const Placeholder();
  }
}
```

Ciclo de vida de un StatefulWidget

El ciclo de vida de un `StatefulWidget` describe las distintas etapas por las que pasa un widget con estado, desde que se crea hasta que se destruye.

Este ciclo es importante para gestionar correctamente el estado, recursos y lógica de una aplicación.

Las fases más importantes (mas usadas) son:

- `initState()`: Inicializa el estado. Se ejecuta una única vez luego de creado el widget
- `build()`: Construye la interfaz. Se ejecuta cada vez que cambia el estado del widget.
- `setState()`: notifica que el estado del widget ha cambiado y que debe ser reconstruido.
- `dispose()`: Limpia los recursos antes de la destrucción del widget.

BuildContext

El `BuildContext` es un objeto que representa el contexto de un widget en el árbol de widgets. Es esencialmente una referencia al lugar donde se encuentra un widget en la jerarquía de widgets.

Cada widget tiene su propio `BuildContext`, y este objeto proporciona información sobre la ubicación y configuración del widget en el árbol.

El `BuildContext` es inmutable y sólo es válido durante un ciclo de construcción de widgets. Cuando se produce una reconstrucción del árbol de widgets, se crean nuevos contextos para los nuevos widgets, y los contextos anteriores se invalidan y ya no son válidos.

El `BuildContext` es una herramienta esencial para interactuar con el entorno del widget y proporciona información valiosa sobre la configuración y ubicación de los widgets en el árbol de widgets. Es una parte crucial en la construcción y renderizado de la interfaz de usuario en Flutter.

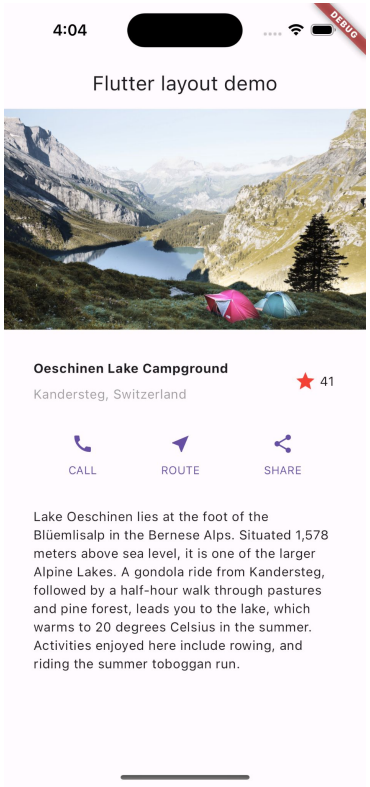
Keys

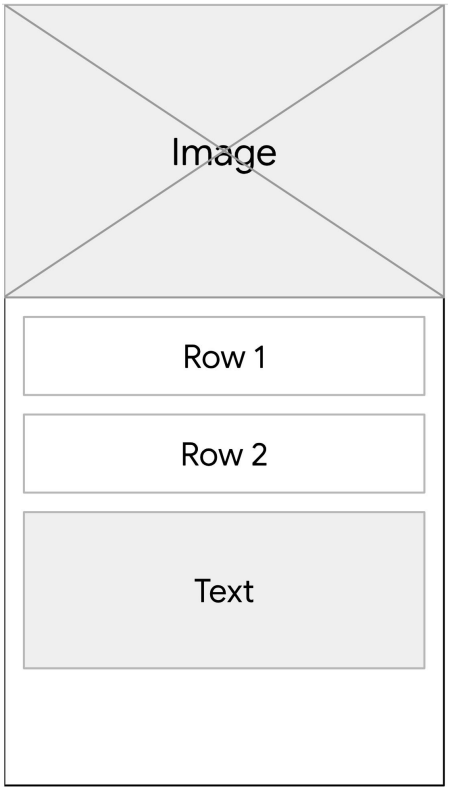
El atributo **key** es un identificador especial que se utiliza para controlar de manera más precisa como los widgets son manejados dentro del árbol de widgets, especialmente cuando Flutter necesita reconstruir la interfaz de usuario.

Esto es útil para garantizar que el estado y las propiedades de los widgets se gestionen correctamente durante los ciclos de reconstrucción y actualización.

Es especialmente útil cuando trabajamos con listas dinámicas, animaciones o interfaces complejas.

Flutter layout codelab





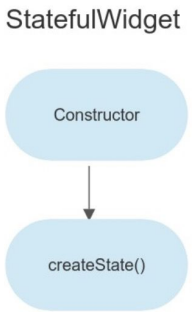
Flutter layout codelab:

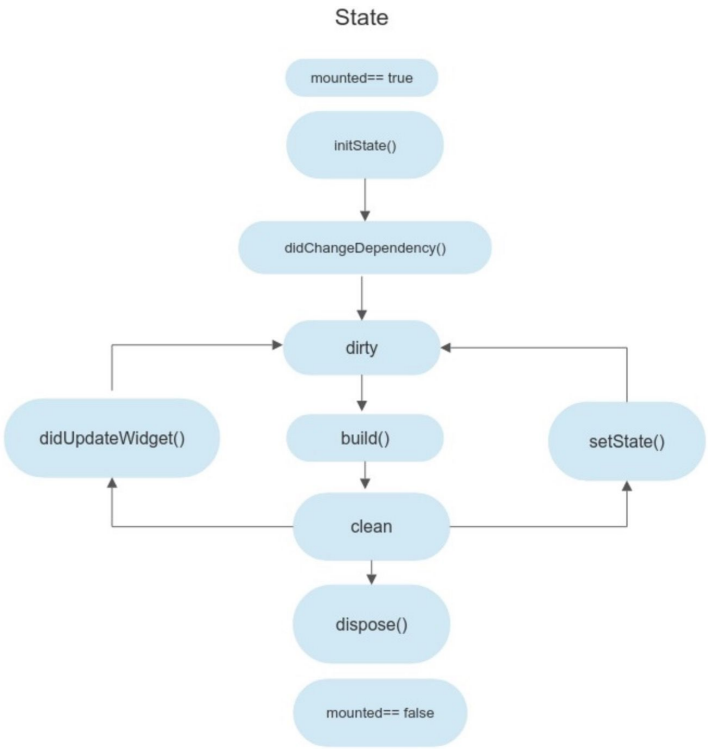
Parte 1: docs.flutter.dev/ui/layout/tutorial

Parte 2: docs.flutter.dev/ui/interactivity

Globant

Ciclo de vida completo de un StatefulWidget





Globant

Ciclo de vida completo de un StatefulWidget

`createState()`

- Es el primer método que se invoca cuando se crea un `StatefulWidget`.
- Flutter llama a este método para crear una instancia de la clase `State` asociada al widget.

InitState()

- Este es el primer método que se invoca dentro de la clase **State**. Es llamado una sola vez cuando se crea la instancia del estado.
- Se usa para inicializar cualquier dato o suscripción que el widget necesite.

`didChangeDependencies()`

- Este método se llama después de **initState()** y cada vez que cambian las dependencias del widget, por ejemplo, si cambia el `InheritedWidget` del cual depende.
- Se suele usar para obtener acceso a dependencias que no estaban disponibles en **initState()**.

Ciclo de vida completo de un StatefulWidget

`build()`

- Es el método que define la interfaz del widget. Se llama cada vez que se reconstruye el widget.
- Este método se puede invocar varias veces durante el ciclo de vida, cada vez que el estado cambia o que el framework requiere una nueva construcción.

setState()

- Llamar a `setState()` notifica a Flutter que el estado del widget ha cambiado y que el widget debe ser reconstruido.
- Este método actualiza la interfaz llamando al método `build()` nuevamente.

`didUpdateWidget()`

- Se llama si el widget padre que contiene este `StatefulWidget` se reconstruye y pasa nuevos datos al widget hijo.
- Útil para actualizar el estado en función de las nuevas propiedades del widget.

Ciclo de vida completo de un StatefulWidget

`deactivate()`

- Se llama cuando el widget se va a eliminar del árbol de widgets pero aún no se ha destruido completamente.
- Este método te permite manejar tareas previas a la eliminación del widget, por ejemplo, eliminar listeners.

dispose()

- Este es el último método que se llama cuando el widget se elimina permanentemente del árbol.
- Aquí es donde debes liberar recursos, cancelar suscripciones, cerrar controladores, etc.

Ciclo de vida completo de un StatefulWidget

mounted : El término `mounted` se refiere a si el widget está actualmente adjunto al árbol de widgets y es parte de la interfaz visible de la aplicación.

- `mounted == true`: El widget está en el árbol de widgets y es parte activa de la interfaz de usuario.
- `mounted == false`: El widget ha sido eliminado del árbol de widgets, pero su estado (`State`) sigue existiendo, probablemente hasta que se llame a `dispose()` para limpiar los recursos.

`mounted` es una propiedad que se puede consultar dentro de la clase `State`. Es útil para verificar si el widget todavía está adjunto al árbol antes de ejecutar operaciones asíncronas o actualizar el estado.

```
if (mounted) {
  setState(() { ... });
}
```

Globant

Ciclo de vida completo de un StatefulWidget

dirty : un widget se considera "`dirty`" cuando su estado ha cambiado y necesita ser reconstruido. Esto generalmente ocurre después de llamar a `setState()`.

- El proceso de marcado como "`dirty`" es lo que le dice al framework que debe ejecutar nuevamente el método `build()` para actualizar la interfaz gráfica.

Flutter optimiza el proceso de renderizado. Solo los widgets marcados como "`dirty`" serán reconstruidos, no todo el árbol de widgets.

```
void _incrementCounter() {
  setState(() {
    _counter++;
  });
}
```

Ahora el widget está "dirty" y Flutter lo reconstruirá en el próximo ciclo de renderizado.

Globant

Ciclo de vida completo de un StatefulWidget

clean : Un widget está en estado "`clean`" cuando no necesita ser reconstruido. Es decir, su estado no ha cambiado y no está marcado para ser actualizado.

- Un widget está en estado "`clean`" cuando su interfaz visual está sincronizada con su estado actual y no necesita ninguna actualización.
- Después de que un widget ha sido reconstruido (es decir, su método `build()` ha sido invocado), pasa del estado `dirty` a `clean`.

Links adicionales

- Flutter samples: flutter.github.io/samples/
- Flutter codelabs: docs.flutter.dev/codelabs
- Flutter cookbook: docs.flutter.dev/cookbook
- Widget catalog: docs.flutter.dev/ui/widgets
- Flutter en YouTube: youtube.com/@flutterdev
    - Widget of the Week: youtube.com/playlist?list=PLjxrf2q8roU23XGwz3Km7sQZFTdB996iG
    - Package of the Week: youtube.com/playlist?list=PLjxrf2q8roU1quF6ny8oFHJ2gBdrYN_AK
    - Observable Flutter: youtube.com/playlist?list=PLjxrf2q8roU1GHtc2FCHoEZr_v-LqnTZX
- Flutter library: flutterlibrary.com

Preguntas?

Globant

Gracias!

Globant