**WORKSHOP – CONCEPTOS BASICOS DE FLUTTER**

**Estructura del proyecto (Entendamos los archivos y carpetas principales).**

La estructura de una aplicación de Flutter puede variar, pero aquí hay una explicación general de las carpetas comunes en un nuevo proyecto de Flutter:

* **lib:** esta carpeta contiene el código fuente principal de la aplicación, incluidos los archivos Dart que definen los widgets, la lógica de la aplicación, etc.
* **test:** esta carpeta contiene pruebas unitarias y de integración para la aplicación.
* **iOS y android:** estas carpetas contienen el código específico de la plataforma para las versiones de iOS y Android de la aplicación, respectivamente.
* **web:** esta carpeta contiene el código para una versión web de la aplicación (consulte Flutter para web).
* **assets:** esta carpeta contiene los archivos de recursos de la aplicación, como imágenes, fuentes, etc.
* **build:** esta carpeta contiene archivos generados por el compilador de Flutter, como archivos .apk y .ipa para la implementación de la aplicación.
* **.metadata:** este archivo contiene metadatos sobre el proyecto Flutter.
* **.packages:** este archivo contiene información sobre los paquetes utilizados en el proyecto.
* **pubspec.yaml:** este archivo define los paquetes de dependencia utilizados en el proyecto y otros metadatos de la aplicación.

**GLOSARIO**

1. VOID: como tipo de devolución se usa para indicar que una función no devuelve nada. Dart permite devolver funciones nulas con tipo de retorno nulo pero también permite usar return; sin especificar ningún valor. Para tener una forma consistente, no debe devolver nulo y solo usar un retorno vacío

2. CLASS: Casi todo el código que escriba en Dart estará contenido en clases. Y una clase es un modelo para un object. Es decir, una clase describe un objeto que puede crear. El objeto en sí es lo que contiene cualquier dato y lógica específicos.

3.EXTENDS: se usa para crear una nueva clase que hereda propiedades y métodos de una clase existente. Esto se utiliza en la programación orientada a objetos para reutilizar y extender el código

4. STATELES WIDGET: es un tipo de widget que no mantiene ningún estado interno. Esto significa que sus propiedades (inputs) no cambian una vez que se crean. Un "StatelessWidget" es utilizado para representar elementos de la interfaz de usuario que no necesitan actualizarse constantemente ni responder a eventos internos. Estos widgets son eficientes en términos de rendimiento ya que no requieren re-renderización cuando los datos cambian, lo que los hace ideales para elementos estáticos en la interfaz de usuario.

5.CONST: es una palabra clave que se utiliza para declarar objetos inmutables en tiempo de compilación. Cuando usas "const" para crear un objeto, estás indicando que sus propiedades y valores no cambiarán después de la compilación. Esto puede ayudar a mejorar el rendimiento y la eficiencia de tu aplicación al evitar reconstrucciones innecesarias de widgets o elementos.

6. RETURN: se utiliza para devolver un valor desde una función. En el contexto de la construcción de widgets, se usa para retornar un widget que se renderizará en la interfaz de usuario. Por ejemplo, al definir una función que construye un widget, puedes utilizar "return" para devolver el widget que deseas mostrar en la pantalla. Esto permite a Flutter comprender qué debe mostrarse en la interfaz de usuario en función del árbol de widgets que estás creando.

7 CHILD: se refiere al widget secundario contenido dentro de otro widget, que actúa como su contenedor o padre. El widget "child" se coloca dentro de otro widget para definir su contenido o contenido interno.

8. SATAFUL WIDGET : es un tipo de widget que puede mantener y actualizar su estado interno. Esto permite que la interfaz de usuario reaccione a cambios en los datos y proporcione una experiencia interactiva. Los "Stateful Widgets" constan de dos clases: una clase que define el widget en sí y otra clase llamada "State" que administra el estado mutable del widget. Cuando los datos cambian, el "State" se actualiza y el widget se reconstruye automáticamente para reflejar esos cambios en la interfaz de usuario.

9. @OVERRIDE: es una anotación que se utiliza para indicar que un método en una subclase está sobrescribiendo un método de la superclase. Esto es especialmente útil en la programación orientada a objetos para asegurarse de que estás reemplazando correctamente el comportamiento de un método heredado.

Cuando utilizas `@override` antes de un método en una subclase, estás indicando explícitamente que estás reemplazando la implementación del método de la superclase. Si la firma del método en la subclase no coincide exactamente con la de la superclase, el compilador generará un error.

10. STATEFUL: se refiere a un tipo de widget que puede mantener y gestionar su propio estado interno a lo largo del tiempo. Los widgets "Stateful" son útiles cuando necesitas que una parte de la interfaz de usuario cambie dinámicamente en respuesta a eventos o datos cambiantes.

Un widget "Stateful" consta de dos clases: una clase que hereda de `StatefulWidget` y otra clase que hereda de `State`. La clase `StatefulWidget` crea una instancia de la clase `State` y delega la gestión del estado a esta última. El estado interno puede cambiar a lo largo del tiempo y, cuando cambia, el widget "Stateful" se reconstruye para reflejar esos cambios en la interfaz de usuario.

11. WIDGET: es un término fundamental que se refiere a los componentes visuales y de diseño que componen la interfaz de usuario de una aplicación. Los widgets son bloques de construcción reutilizables que pueden incluir elementos como botones, texto, imágenes, campos de entrada y mucho más. Los widgets se agrupan en una jerarquía para formar la interfaz de usuario completa de una aplicación Flutter. Pueden ser simples, como un widget de texto, o más complejos, como un widget de lista desplazable que contiene varios otros widgets. La combinación y anidación de widgets permite crear interfaces de usuario ricas y personalizadas en Flutter.

12 WIDGET BUILD: es donde especificas cómo debe lucir y comportarse tu widget en la interfaz de usuario.

13. FINAL : se utiliza para declarar una variable que no puede cambiar su valor una vez que se le ha asignado un valor inicial. En otras palabras, una vez que una variable se declara como "final", no se puede modificar su valor posteriormente en el código. Esto puede ser útil cuando deseas asegurarte de que el valor de una variable no cambie después de su asignación inicial.

14. CHILDREN : se refiere a una propiedad utilizada en muchos widgets para definir una lista de widgets secundarios que serán colocados y renderizados dentro del widget principal. Por ejemplo, en el widget `Column` o `Row`, puedes usar la propiedad "children" para especificar una lista de widgets que se alinearán vertical u horizontalmente, respectivamente. Esto permite crear interfaces de usuario complejas y dinámicas al combinar varios widgets secundarios dentro de un widget principal.