

LABORATORIO	09 – CRONÓMETRO
OBJETIVO GENERAL	Aplicar los StatefulWidget
INSTRUCCIONES	Elabora tu propia versión de un cronómetro.
NOMBRE ESTUDIANTE	Itsai Zempoaltecatl
FECHA	17/marzo/2022
GRUPO	8SA

RECURSOS

Puedes utilizar el recurso que te comparto para la creación de tu cronómetro

- <https://youtu.be/mKYE7wRNa7Y>
- <https://docs.flutter.dev/cookbook/lists/long-lists>

Al finalizar tu aplicación básica deberás ser capaz de colocar diferentes tiempos en un **ListView.builder** [no usar otra estructura, usar obligatoriamente]. Al finalizar tu aplicación deberá tener una apariencia similar a:



Puntos a entregar:

- Entregar video de 5 minutos como máximo con **formato MP4 (si es otro formato no se revisa)**, donde expliques como implementaste tu aplicación **detalladamente**, deberás **mostrar el código durante la explicación así como la aplicación desarrollada**.
- La aplicación deberá correr en **un emulador o dispositivo físico**, la grabación se debe apreciar correctamente
- Entregar reporte escrito en formato PDF

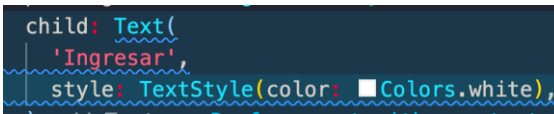
Elementos a calificar

- Reporte 10 pts
- Funcionamiento app 10 pts
- Diseño de la app 10pts

- Explicación del video 10pts, [la explicación deberá ser detallada, en caso de no dominar la explicación, la aplicación no se tomará en cuenta]

Te resta puntos 🖐

- Usar widgets que no explicas en tu video-10pts
- Usar widgets deprecate (desactualizados) -10pts
- Si la aplicación presenta al menos 1 warning -10pts

○ 

```
child: Text(  
  'Ingresar',  
  style: TextStyle(color: Colors.white),  
),
```

Ilustración 1 Ejemplo de warning

INTRODUCCIÓN

El widget `ListView.builder()` nos ayuda a crear widgets bajo demanda o mejor dicho en tiempo de ejecución con un scroll, es útil cuando no sabemos la cantidad de widgets que usaremos o su aumentarán en el momento de que la aplicación este en uso.

En este ejemplo se utilizó con un cronometro, se fue creando un widget `Container` con un `child Row` que a su vez contenía 2 `Text`, como el aumento de los widgets fue dinámico fue necesario usar este widget.

DESARROLLO

```
void iniciarCronometro() {
    if (!estaCorriendo) {
        timer = Timer.periodic(const Duration(milliseconds: 100), (timer) {
            milisegundos += 100;
            setState(() {});
        }); // Timer.periodic
        estaCorriendo = true;
    }
}

String formatearTiempo() {
    Duration duration = Duration(milliseconds: this.milisegundos);
    String dosValores(int valor) {
        return valor >= 10 ? "$valor" : "0$valor";
    }

    String dosValoresMili(int valor) {
        return valor == 0 ? "000" : "$valor";
    }

    String minutos = dosValores(duration.inMinutes.remainder(60));
    String segundos = dosValores(duration.inSeconds.remainder(60));
    String milisegundos =
        dosValoresMili(duration.inMilliseconds.remainder(1000));
    return "$minutos:$segundos:$milisegundos";
}

void detenerCronometro() {
    if (estaCorriendo) {
        timer.cancel();
        estaCorriendo = false;
    }
}

int milisegundos = 0;
late Timer timer;
bool estaCorriendo = false; }
```

Métodos para mostrar el tiempo:

iniciaCronometro(): inicializa un timer con periodos de Duration() de 100 milisegundos, entonces la variable milisegundos aumentara 100 cada 100 milisegundos.

formatearTiempo(): le da un formato al tiempo que ha pasado en base al valor de la variable milisegundos, podemos notar que se va obteniendo el residuo en minutos, segundos y milisegundos, al final se devuelve un string que mostraremos en la aplicación

detenerTiempo(): pregunta a la bandera estaCorriendo si el timer estar activo, en caso de que si lo detiene con timer.cancel()

```
final listVueltas = <Vuelta>[];
int contadorVueltas = 0;

class Vuelta {
    final String idVuelta;
    final String tiempo;
    Vuelta(this.idVuelta, this.tiempo);
}
```

listVueltas: crear una lista de la clase Vuelta (imagen derecha) que contiene 2 propiedades, esto servirá para guardar los valores de las vueltas cada que lo necesitemos

contadorVueltas: será el contador de las vueltas que iremos agregando

```
Container(
    padding: const EdgeInsets.only(top: 200, bottom: 100),
    alignment: Alignment.center,
    child: Text(
        formatearTiempo(),
        style: const TextStyle(fontSize: 100, color: Colors.white),
    ), // Text
), // Container
```

Text toma el valor que está regresando el método de formatearTiempo

```
onPressed: () {
  setState(() {
    if (estaCorriendo) {
      contadorVueltas += 1;
      listVueltas.add(Vuelta(
        'Vuelta $contadorVueltas', formatearTiempo()));
    } else {
      iniciarCronometro();
    }
  });
}

onPressed: () {
  setState(() {
    if (estaCorriendo) {
      detenerCronometro();
    } else {
      milisegundos = 0;
      contadorVueltas = 0;
      listVueltas.clear();
    }
  });
}
```

onPress del botón 1 (izquierda), en base a la bandera estaCorriendo asigna una funcionalidad al botón, en el caso que el cronometro este corriendo agrega una nueva Vuelta a la lista de vueltas, igual aumenta el contador de las vueltas
Si no está corriendo, solo inicia el cronometro

onPress botón 2 (derecha), al igual que el anterior, si el cronometro está corriendo, le da la funcionalidad de parar el boton, en caso contrario reinicia el cronometro, la variable milisegundos, el contador de vueltas y limpia la lista de vueltas.

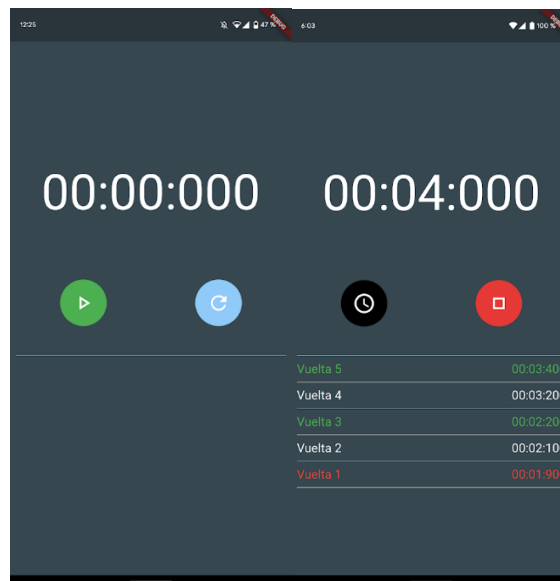
```
itemCount: contadorVueltas,
itemBuilder: (context, int index) {
  return Container(
    padding: const EdgeInsets.only(top: 10, bottom: 10),
    decoration: const BoxDecoration(
      border: Border(
        bottom: BorderSide(width: 0.5, color: Colors.white)),
    ), // BoxDecoration
    child: Row(
      mainAxisAlignment: MainAxisAlignment.spaceBetween,
      children: [
        Text(
          listVueltas[index].idVuelta,
          style: colorText(index + 1),
        ), // Text
        Text(listVueltas[index].tiempo,
          style: colorText(index + 1)) // Text
      ],
    ),
  );
}
```

Para llenar el ListView.builder en el atributo itemBuilder se crea un método con 2 parametros, el segundo (index) nos funciona como un índice que va aumentando de 1 en 1 empezando desde 0, este es útil para ir recorriendo la listas que usaremos, en este caso listVueltas.

El método devuelve los widgets que llenarán el ListView.builder, como se ve en la imagen se crea un Container con un child Row, Row contiene 2 Text, cada uno de ellos tomaran la información de la lista que está siendo recorrida, el primer Text es el idVuelta, el segundo el tiempo y así cada widget se va creando con el mismo formato pero la información cambia

```
TextStyle colorText(int i) {
  Color c = (i == 1)
    ? Colors.red
    : (i % 2 == 0)
    ? Colors.white
    : Colors.green;
  return TextStyle(fontSize: 25, color: c);
}
```

En los Text que se agregan al ListView.builder vemos que el style tienen método para darle estilo, es este método. Va revisando el index en el que se encuentra, si es el primero regresa un estilo con color rojo, si es par blanco y no verde



Aplicación resultado, del lado izquierdo tenemos el cronometro sin iniciar, vemos los botones de iniciar y reiniciar, del otro lado vemos cuando el cronometro ya está iniciado y a su vez cuenta con vueltas agregadas

CONCLUSIONES

El widget `ListView.builder` es demasiado útil para aplicaciones en las que no sabemos cuantos elementos tendremos en la pantalla, podríamos verlo como en una app de comida, tenemos muchos platillos, esa información viene de una base de datos y en base a esa información se llenará la lista, como no sabemos los platillos exactos no podemos usar un `ListView` convencional o un `Column` ya que estos widgets están definidos y no se pueden ir agregando widgets en tiempo de ejecución.

REFERENCIAS

Xenia Padilla. 2021. 24 FLUTTER CRONOMETRO CON STATEFULWIDGETS (Video). YouTube.
<https://www.youtube.com/watch?v=mKYE7wRN7Y>

Flutter. List class, de Flutter Sitio web: <https://api.dart.dev/stable/2.16.1/dart-core/List-class.html>

Flutter.Border class, de Flutter Sitio web:
<https://api.flutter.dev/flutter/painting/Border-class.html>

Flutter. List class, de Flutter Sitio web: <https://api.dart.dev/stable/2.16.1/dart-core/List-class.html>

Flutter. `ListView.builder` constructor, de Flutter Sitio web:
<https://api.flutter.dev/flutter/widgets/ListView/ListView.builder.html>

Desconocido. (2017). Flutter - Bottom Overflowed By 119 Pixels [duplicate]. 2017, de stackoverflow Sitio web: <https://stackoverflow.com/questions/54156420/flutter-bottom-overflowed-by-119-pixels>