| | |
|---|---|
| **LABORATORIO** | 11 – ASYNC, AWAIT Y FUTURES |
| **OBJETIVO GENERAL** | Elabora una aplicación libre donde explique el uso de Async, await y future |
| **INSTRUCCIONES** | Elabora una aplicación libre donde apliques los conceptos de await, async y future |
| **NOMBRE ESTUDIANTE** | Itsai Zempoaltecatl Mejia |
| **FECHA** | 26/marzo/2022 |
| **GRUPO** | 8SA |

RECURSOS

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

- https://youtu.be/AoAlp7V7nA0
- https://youtu.be/lJJuZAcDkz0

**Puedes consultar más contenido digital y colocar las referencias.**

Al finalizar tu aplicación deberás explicar perfectamente el uso de async, await y de future; deberás demostrar los conceptos de forma aplicada dentro de tu aplicación

**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.**
- **Cuidado de copiar y pegar el código que ves en los videos, eso se considerará plagio.**
  **[ya encontré varios trabajos así y quedan anulados]**
- La aplicación deberá correr en **un emulador o dispositivo físico**, la grabación se debe apreciar correctamente en un solo video.
- Entregar reporte escrito en formato PDF

Elementos a calificar

- Reporte 20 pts
- Funcionamiento app 40 pts
- Diseño de la app 20pts
- Explicación del video 20pts, [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
- **Si el código que presentas es copiado y pegado (actividad anulada)**
  - 


*Ilustración 1 Ejemplo de warning*

---

INTRODUCCIÓN

Los métodos asíncronos permiten que el programa complete su ejecución mientras espera el resultado de una operación.

async – await: Proporcionan una manera de definir funciones asíncronas y usar sus resultados. Se agrega 'async' antes del cuerpo de la función para definir una función asíncrona, 'await' solo funciona en funciones asíncronas.

En la forma de () async { await }, await representa una espera al resultado de una operación, como podría ser una consulta a una base de datos (http-get) y hasta que esta sea resuelta todo lo que venga después de la sentencia donde se encuentra await será ejecutado.

Por otro lado, tenemos a Future, una función de este tipo se ejecuta asíncronamente al programa en el momento que llamemos al método.

Cuando una consulta puede presentar un retraso usamos Future, ya que nos permite ejecutar y mostrar la interfaz, mientras a su vez ejecuta la operación que se asignemos a Future. Igual podemos usar async-await en el método Future

```
String createOrderMessage() {
  var order = fetchUserOrder();
  return 'Your order is: $order';
}

Future<String> fetchUserOrder() =>
    // Imagine that this function is
    // more complex and slow.
    Future.delayed(
      const Duration(seconds: 2),
      () => 'Large Latte',
    );

void main() {
  print('Fetching user order...');
  print(createOrderMessage());
}
```

Ejemplo de Future

```dart
Future<String> createOrderMessage() async {
  var order = await fetchUserOrder();
  return 'Your order is: $order';
}

Future<String> fetchUserOrder() =>
    // Imagine that this function is
    // more complex and slow.
    Future.delayed(
      const Duration(seconds: 2),
      () => 'Large Latte',
    );

Future<void> main() async {
  print('Fetching user order...');
  print(await createOrderMessage());
}
```

Future con async y await

La siguiente aplicación implementa este concepto de los métodos asíncronos, tanto Future como async-await. Consta de 2 pantallas, en la primera mostrar la información obtenida de la segunda.

DESARROLLO

```dart
ElevatedButton(
    onPressed: () async {
      final data = await Navigator.push(context,
          MaterialPageRoute(builder: (__) => SecondPage()));
      setState(() {
        name = data.name;
        secondName = data.secondName;
        email = data.email;
      });
      getImage().then((value) {
        setState(() {
          image = value;
        });
      });
    },
    child: Row(
      mainAxisSize: MainAxisSize.min,
      children: const [
        Text("Acceder"),
        Icon(Icons.arrow_forward_ios),
      ],
    )) // Row // ElevatedButton
```

La primer pantalla cuenta con un botón que nos dirige a la segunda pantalla, en el momento que la segunda pantalla ejecute Navigator.pop() este puede enviar datos y son obtenidos en Navigator.push(). Uso await para que la app espere a obtener la información de la segunda pantalla para guardarla en una variable ‘data’ y después usar esta información.

```dart
appBar: AppBar(
  title: const Text('Segunda pantalla'),
  leading: IconButton(
    icon: const Icon(Icons.arrow_back_outlined),
    onPressed: () {
      if (Navigator.canPop(context)) {
        Navigator.pop(context, Information(name, secondName, email));
      }
    },
```

En la segunda pantalla en el leading de nuestro appBar, tenemos un IconButton que nos servira para ejecutar Navigator.pop, el cual devolvera a la pantalla principal la clase Information con los parametros que tomaron el valor de los TextField

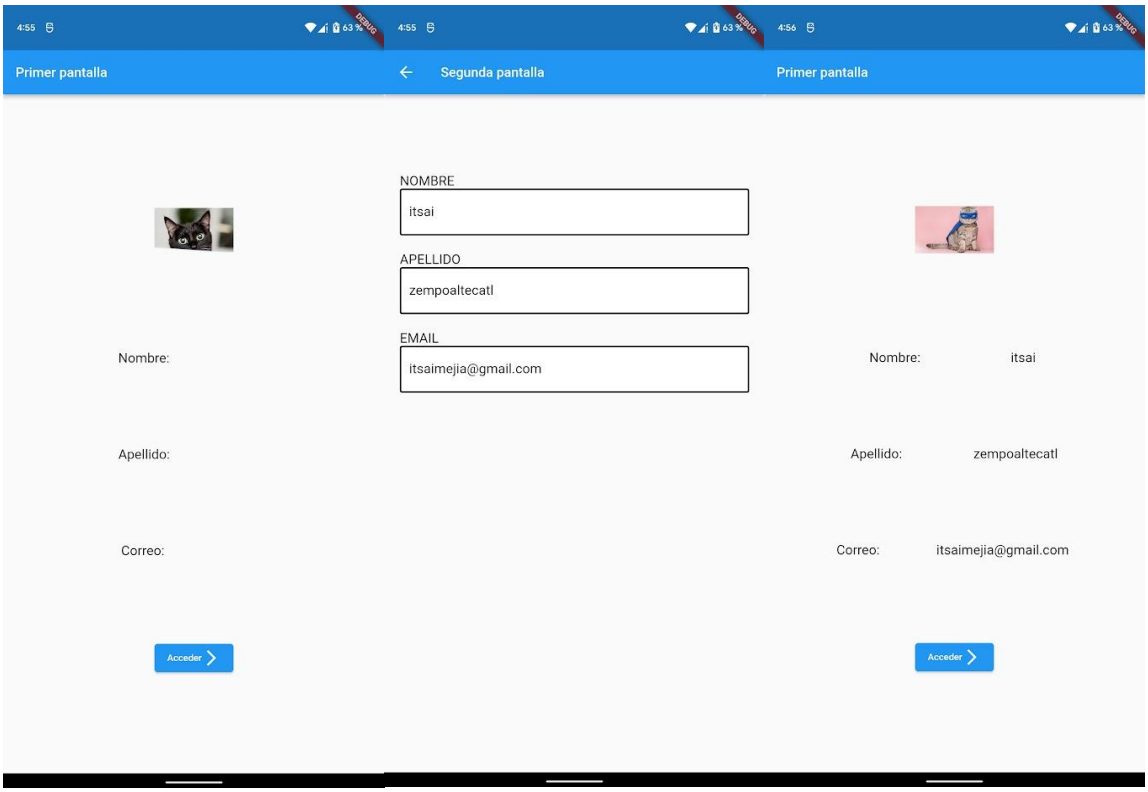
```
final data = await Navigator.push(context,
    MaterialPageRoute(builder: (__) => SecondPage()));
setState(() {
  name = data.name;
  secondName = data.secondName;
  email = data.email;
});
```

Una vez obtenida la informacion de la pantalla 2, cambie el valor de las variables locales por las de la clase que esta guardara en ‘data’

```
Future getImage() {
  return Future.delayed(
      const Duration(seconds: 5),
      () =>
          'https://www.hola.com/imagenes/mascotas/20200911175064/gatos-imagenes-curiosidades/0-864-253/gatos-t.jpg');
}
```

```
getImage().then((value) {
  setState(() {
    image = value;
  });
});
```

Como el código después de await se ejecutara hasta que await sea resuelto, tenemos que ejecutara el método Future getImage(), este método devuleve el link que se aprecia después de 5 segundos a partir desde que se instancia.
La primer pantalla cuenta con una Image.network() que toma su valor de la variable ‘image’ la cual es un url a una imagen, cuando ejecutamos getImage, usamos .then para obtener el valor que devuelva el método y con ayuda de setState re asignamos a image el url que obtenemos del método getImage.
Por lo que el Image.network() cambiara su imagen 5 segundos después de obtener el valor de Navigator.push()




Resultados

## CONCLUSIONES

Los métodos asíncronos son muy importantes para el desarrollo de cualquier sistema, ya que nos ayudan a evitar una mala experiencia de usuario, al igual que agiliza el funcionamiento del sistema. En el caso de las apps móviles la mayoría de casos necesitamos obtener datos a partir de consultas a bases de datos en servidores (web services), si esperáramos a realizar la consulta sin métodos asíncronos detendríamos la ejecución de la aplicación.

## REFERENCIAS

Xenia Padilla. [Xenia Padilla] (2021). 34 FLUTTER ASYNC AWAIT [Video]. Youtube. https://www.youtube.com/watch?v=AoAlp7V7nA0&t=317s

Fernando Herrera. [Fernando Herrera] (2021). 14/14 Async & Await [Video]. Youtube. https://www.youtube.com/watch?v=gWS-MGS10hc&t=265s

Fernando Herrera. [Fernando Herrera] (2021). 13/14 Futures [Video]. Youtube. https://www.youtube.com/watch?v=lJJuZAcDkz0&t=396s

Dart. (2022). Asynchronous programming: futures, async, await. Dart. https://dart.dev/codelabs/async-await