Como tomar una foto con Flutter o elegir una foto de la galería.

Empecemos por las dependencias.

Como primer paso, vamos a agregar los plugins a nuestras dependencias. Abrimos nuestro **pubspec.yaml** y agregamos los siguientes plugins:

- **camera:** Este plugin nos ayudará a trabajar con las cámaras de los dispositivos.
- **path_provider:** Este plugin nos proporciona el **path** correcto para saber donde almacenar las imagenes en nuestro dispositivo, ya que cambian entre una plataforma y otra.
- **image_picker** este nos ayuda a seleccionar una foto de la galería.

Despues de agregar estos plugins nuestro **pubspec.yaml** debería verse así.

```
name: fluttercamera
description: A Flutter project that takes a picture and shows the preview.
version: 1.0.0+1
environment:
sdk: ">=2.1.0 <3.0.0"
dependencies:
flutter:
sdk: flutter
camera:
path_provider:
path:
image_picker:
cupertino_icons: ^0.1.2
dev_dependencies:
flutter_test:
sdk: flutter
flutter:
```

> uses-material-design: true

Aumentamos la versión minima de Android

El plugin de **camera** en Flutter solo funciona con sdk 21 en adelante en Android, lo que quiere decir que solo dispositivos Android con sistema operativo **Lollipop** o superior pueden utilizar una app con este plugin. Para esto vamos a abrir el archivo build.gradle ubicado en `android/app/build.gradle` y buscar la linea que dice «`minSdkVerrsion`«. Finalmente cambiamos la versión de 16 a 21. Luego de cambiar la versión debería verse así

¡Creemos nuestro primer Screen!

Para empezar vamos a crear un **StatefulWidget** llamado **PhotoPreviewScreen**. Debería verse de esta manera.

```
class PhotoPreviewScreen extends StatefulWidget {
@override
_PhotoPreviewScreenState createState() => _PhotoPreviewScreenState();
}
class _PhotoPreviewScreenState extends State<PhotoPreviewScreen> {
@override
Widget build(BuildContext context) {
return Scaffold(
);
}
}
```

Ahora, vamos a crear un **Scaffold** con un widget Column centrado el cual va a mostrar una vista previa de nuestra imagen después que tomemos una foto o la seleccionemos de la galería. Y como paso final vamos a agregar

un **floatingActionButton**, que al tocar nos mostrará un dialogo con las opciones para elegir una foto de la galería o desde la cámara.

```
class PhotoPreviewScreen extends StatefulWidget {
@override
_PhotoPreviewScreenState createState() => _PhotoPreviewScreenState();
}
class _PhotoPreviewScreenState extends State<PhotoPreviewScreen> {
@override
Widget build(BuildContext context) {
return Scaffold(
body: Center(
child: Column(
mainAxisAlignment: MainAxisAlignment.center,
children: <Widget>[
_setImageView()
],
),
),
floatingActionButton: FloatingActionButton(
onPressed: () {
_showSelectionDialog(context);
},
child: Icon(Icons.camera_alt),
),
);
}
```

¡Genial! Ahora seguramente te estés preguntando qué hace el **_setImageView()** y el **_showSelectionDialog(Context).**

El **_setImageView()** es un método que devuelve un Image Widget en caso de que la imagen obtenida no sea nula, de lo contrarío devolverá un Text Widget con un mensaje de Error. El método **_showSelectionDialog(Context)** muestra un dialogo con dos opciones, seleccionar una imagen de galería o de la cámara.

Empecemos creando el último, este método debe usar la función **showDialog()** y pasarle el **Context** y un **Builder** el cual va crear un AlertDialog con un titulo y dos opciones.

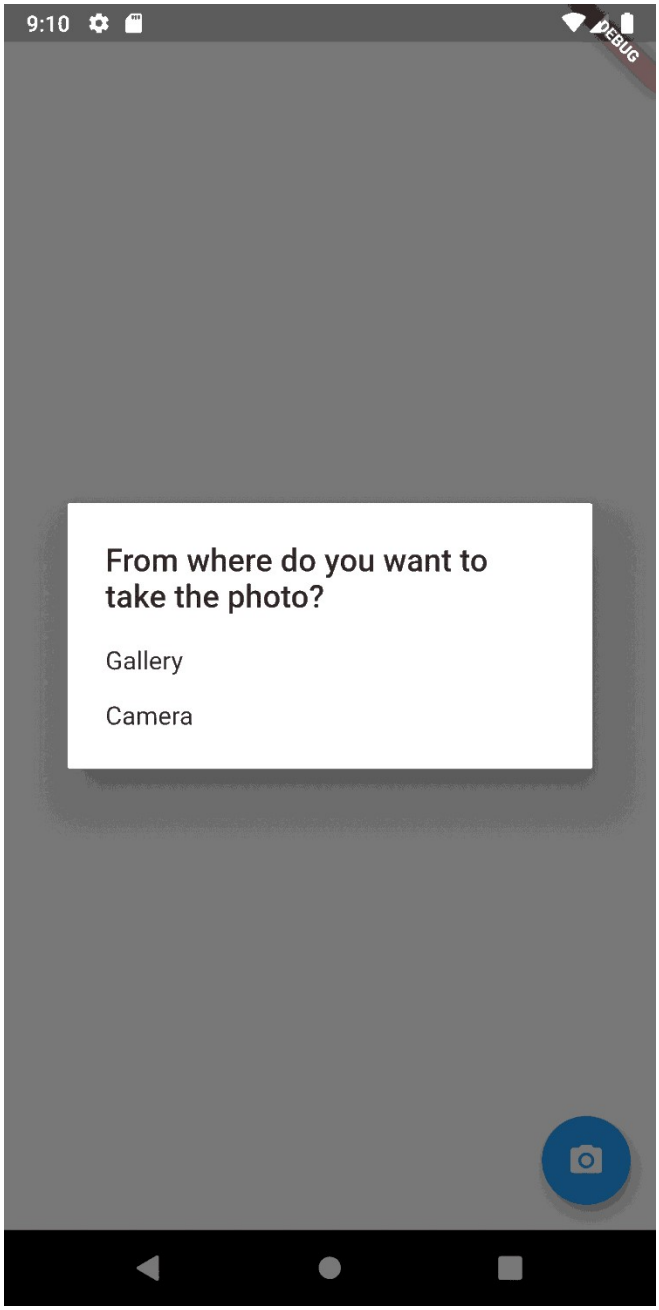
En el constructor del **AlertDialog**, vamos a pasar como contenido un **SingleChildScrollView** el cual es un Widget que nos ayudará a hacer scroll en la lista, y como **child** de este Widget vamos a pasarle un **ListBody** con dos **GestureDetector** como hijos para detectar cuándo el usuario ha tocado el texto.

Cada **GestureDetector** va a tener un Text Widget con el texto «Galería» y «Cámara» respectivamente. Y cada Text widget tambíen tendrá como parámetro de su función onTap el

método **_openGallery()** y **openCamera()** segun corresponda. Estos métodos los crearemos más adelante. Al terminar el método **_showSelectionDialog(context)** estará así.

```
Future<void> _showSelectionDialog(BuildContext context) {
return showDialog(
context: context,
builder: (BuildContext context) {
return AlertDialog(
title: Text("From where do you want to take the photo?"),
content: SingleChildScrollView(
child: ListBody(
children: <Widget>[
GestureDetector(
child: Text("Gallery"),
onTap: () {
_openGallery(context);
},
),
Padding(padding: EdgeInsets.all(8.0)),
GestureDetector(
child: Text("Camera"),
onTap: () {
_openCamera(context);
},
)
],
),
));
});
}
```

Ahora al tocar nuestro FloatingActionButton debería mostrar un Dialog como este.


9:10
DEBUG
From where do you want to take the photo?
Gallery
Camera

Ahora a utilizar el plugin ImagePicker

Ahora vamos a agregarle la lógica a nuestro método **_openGallery(context).** Lo primero que vamos a hacer es crear un **Field** llamado imageFile, el cual va a ser una variable de tipo **File** en nuestro **_LandingScreenState.** Luego vamos a utilizar la función del **ImagePicker**, llamada **pickImage()** y le vamos a pasar como parámetro el enum **ImageSource.gallery.** Esta función es asíncrona así que vamos a tener que utilizar las palabras reservadas async y await. Luego vamos a almacenar esta variable y asignarla a nuestra variable **imageFile**. Como paso final, vamos a llamar al método **setState()** para notificar que el **State** ha cambiado. Y nuestra función deberá verse así.

```
void _openGallery(BuildContext context) async {
var picture = await ImagePicker.pickImage(source:
ImageSource.gallery);
this.setState(() {
imageFile = picture;
});
Navigator.of(context).pop();
}
```

La función _openCamera(Context) es casi igual, la unica diferencía que tiene al respecto es que esta utiliza el enum imageSource.camera en lugar de gallery. Al realizar este cambio tu función de **_openCamera()** deberá verse así.

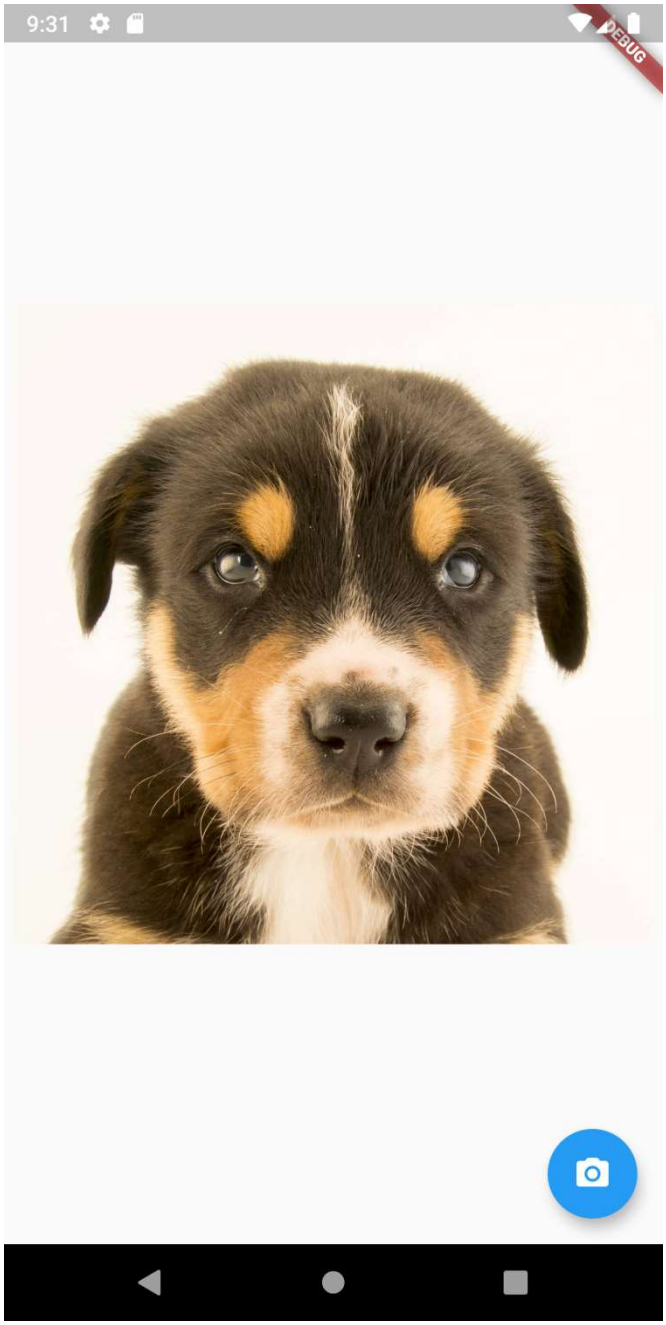
```
void _openCamera(BuildContext context) async {
var picture = await ImagePicker.pickImage(source: ImageSource.camera);
this.setState(() {
imageFile = picture;
});
Navigator.of(context).pop();
}
```

Previsualización

¿Recuerdas aquella función que mencionamos al principio llamada **_setImageView()?** Pues no podemos olvidarnos de ella. Para ello vamos a verificar si nuestra variable imageFile es distinta de null. Si lo es, devolvemos un **Image** Widget con la variable que teniamos almacenada y si aún es nula pues devolvemos un texto que diga que debemos seleccionar una imagen.
Así se vería la función terminada.

```
Widget _setImageView() {
if (imageFile != null) {
return Image.file(imageFile, width: 500, height: 500);
} else {
return Text("Please select an image");
}
}
```

Así es como se vería la app al final, luego de seleccionar una imagen de la galería.


9:31
DEBUG