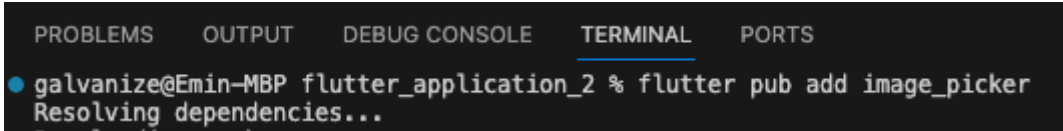


## Cihaz Kamerası Kullanımı

Bu derste, Flutter'da image\_picker paketi kullanarak cihazın kamerasını nasıl kullanacağınızı öğreneceğiz.

### 1. Image\_picker Paketinin Eklenmesi:

[https://pub.dev/packages/image\\_picker/install](https://pub.dev/packages/image_picker/install) sayfasına giderek kütüphaneyi bulun ve VS Code terminal içerisinde `flutter pub add image_picker` komutunu çalıştırın.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
galvanize@Emin-MBP flutter_application_2 % flutter pub add image_picker
Resolving dependencies...
```

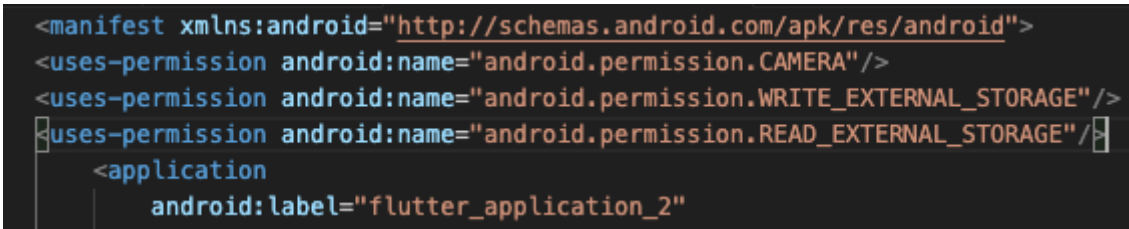
### 2. Android İzinleri Sağlanması:

Android uygulama geliştirme sürecinde belirli özellikleri kullanabilmek için "izinler" (permissions) gereklidir. Bu izinleri ayarlayabilmek için **android/app/src/main/AndroidManifest.xml** dosyasını açın. **<application>** etiketi dışına aşağıdaki kodları ekleyin.

```
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

Bu izinler sıra ile:

- Uygulamanın cihaz kamerasına erişimini,
- Depolama alanına veri yazma ve
- Depolama alanından okuma işlemleri izinlerini kapsar.



```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<application
    android:label="flutter_application_2"
```

3. **Gerekli Kütüphanelerin İçe Aktarılması:** Flutter'ın temel bileşenleri (**material.dart**) ve kamera işlemleri (**image\_picker**) ve dosya işlemleri için (**dart:io**) kütüphanelerini main.dart dosyasına ekleyelim.

```
1 import 'package:flutter/material.dart';
2 import 'package:image_picker/image_picker.dart';
3 import 'dart:io';
4
```

4. **Main Fonksiyonunu Yazma:** Kodlardan biri olan main fonksiyonu yazarak projemize devam edelim. RunApp içerisinde **MyApp** sınıfını çalıştıralım.

```
void main() {
  runApp(MyApp());
}
```

5. **Kullanıcı Arayüzünün Temellerini Oluşturma:** Kullanıcı arayüzü için bir **StatelessWidget** oluşturalım ve uygulama ana görünümü için **CameraApp** sınıfını çağıralım.

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: CameraApp(),
    );
  }
}
```

CameraApp statefull widget'ını değişken arayüz için oluşturalım.

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

  @override
  State<CameraApp> createState() => _CameraAppState();
}

class _CameraAppState extends State<CameraApp> {
  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```

6. **Scaffold ve Başlık Çubuğu Ekleme:** Scaffold widget'ı, tipik bir Flutter uygulaması için iskelet yapıyı oluşturur. AppBar widget'ı, uygulamanın üst kısmındaki başlık çubuğunu oluşturur.

```
class _CameraAppState extends State<CameraApp> {  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: const Text("Fotoğraf Çekme Uygulaması"),  
        centerTitle: true,  
      ), // AppBar  
    ); // Scaffold  
  }  
}
```

7. **Uygulama Gövdesinin Oluşturulması:** Center widget'ı, içeriği ekranın tam ortasına hizalar. Column widget'ı, birden fazla widget'ı dikey olarak hizalar. mainAxisAlignment: MainAxisAlignment.center, widget'ların dikey olarak ortalanmasını sağlar.

```
), // AppBar  
body: const Center(  
  child: Column(  
    mainAxisAlignment: MainAxisAlignment.center,  
    children: [],  
  ), // Column  
), // Center
```

8. **Fotoğraf Çekme Düğmesinin Eklenmesi:**

```
body: Center(  
  child: Column(  
    mainAxisAlignment: MainAxisAlignment.center,  
    children: [  
      ElevatedButton(onPressed: () {}, child: const Text("Foto Çek")),  
    ],  
  ),  
)
```

9. **Fotoğrafı Tutacak Bir Değişken Tanımlanması:** Çekilen fotoğrafın uygulama içinde gösterilmesi ve üzerinde işlem yapılabilmesi için dosya yolunun saklanması gerekir.

```
class _CameraAppState extends State<CameraApp> {  
  File? _image;  
  
  @override
```

- File sınıfı, çekilen fotoğrafın dosya yolunu tutmak için kullanılır.

- `?` ifadesi, bu değişkenin null olabileceğini belirtir. Uygulama başladığında henüz fotoğraf çekilmediği için `_image` değişkeni başlangıçta null olacaktır.

**10. Fotoğraf Çekim Nesnesini Oluşturma: ImagePicker sınıfı,** Flutter'ın sağladığı bir kütüphanedir ve cihazın kamerasını açmayı sağlar.

```
File? _image;  
final ImagePicker _picker = ImagePicker();
```

- **ImagePicker:** Flutter'da kamera veya galeriden fotoğraf/video seçmek için kullanılan bir sınıftır.
- **\_picker:** Fotoğraf çekme işlemini gerçekleştirmek için bu nesne kullanılır.
- **final:** Bu nesne değiştirilemez, ancak tekrar tekrar kullanılabilir.

**11. Fotoğraf Çekme Fonksiyonunun Tanımlanması:**

```
final ImagePicker _picker = ImagePicker();  
  
Future<void> _takePhoto() async {}
```

- **Future:** bu işlemin sonucunun gelecekte döneceğini ifade eder.
- **void:** Bu fonksiyon herhangi bir değer döndürmez, yalnızca bir işlem yapar.
- **async:** Bu anahtar kelime, fonksiyonun içerisinde zaman alabilen (asenkron) işlemlerin yapılacağını belirtir. **await** ile birlikte çalışır ve asenkron işlemlerin tamamlanmasını bekler.

**12. Fonksiyon Gövdesinin Oluşturulması:** Bu fonksiyonun şu anki hali boş. Ancak içerisine fotoğraf çekme işlemini gerçekleştiren kodlar eklenmelidir.

```
Future<void> _takePhoto() async {  
  final XFile? photo = await _picker.pickImage(source: ImageSource.camera);  
}
```

- **XFile:** Çekilen veya seçilen dosyanın (fotoğraf veya video) bilgilerini tutan bir sınıftır. Dosyanın yolu gibi bilgilere erişmeyi sağlar. **?** (**nullable**): Bu işaret, `photo` değişkeninin `null` olabileceğini ifade eder.

- **photo**: Kullanıcının çektiği veya seçtiği görüntü bilgilerini tutar.
- **await \_picker.pickImage(source: ImageSource.camera)**: Kamerası açar bekler ve kullanıcının fotoğraf çekmesini sağlar.

### 13. Dönen Fotoğraf Değişkeninin Kontrolü:

```
final XFile? photo = await _picker.pickImage(source: ImageSource.camera);  
if (photo != null){}
```

- **!=** operatörü, "eşit değil" anlamına gelir.

### 14. Çekilen Fotoğrafın Dosya Yolunu Değişkenine Atama: Bu işlem fotoğrafı görüntülemek için gereklidir.

```
if (photo != null) {  
  setState(() {  
    _image = File(photo.path);  
  });  
}
```

- **\_image**: Daha önce tanımlanmış bir **File?** türünde değişkendir.
- **File** sınıfı, bir dosyayı temsil eder.
- **photo.path**: Çekilen fotoğrafın cihazda depolandığı dosya yoludur.
- **File(photo.path)**: Dosya yolunu kullanarak fotoğrafı bir **File** nesnesine dönüştürür.

### 15. Düğme ile Fotoğraf Çekim Fonksiyonun Çağırılması:

```
ElevatedButton(  
  onPressed: _takePhoto, child: const Text("Foto Çek")),
```

**16. Kullanıcı Arayüzüne Image Eklenmesi:** Bu işlemi children içerisinde kullanabilmek için bir widget olarak yapacağız.

```
Widget buildImage() {  
  if (_image != null) {  
    return Image.file(  
      _image!,  
      width: 300,  
      height: 300,  
    );  
  } else {  
    return const Text("Henüz fotoğraf çekilmedi.");  
  }  
}  
  
@override
```

- **\_image!**, null güvenliği olan Dart'ta, bir değişkenin kesinlikle null olmadığını bildirmek için kullanılır.

**17. Body'de BuildImage Widget'ının Çağırılması:**

```
mainAxisAlignment: MainAxisAlignment,  
children: [  
  buildImage(),  
  ElevatedButton(  
    onPressed: () {  
      // ...  
    },  
    child: Text("Fotoğraf Çek"),  
  ),  
],
```