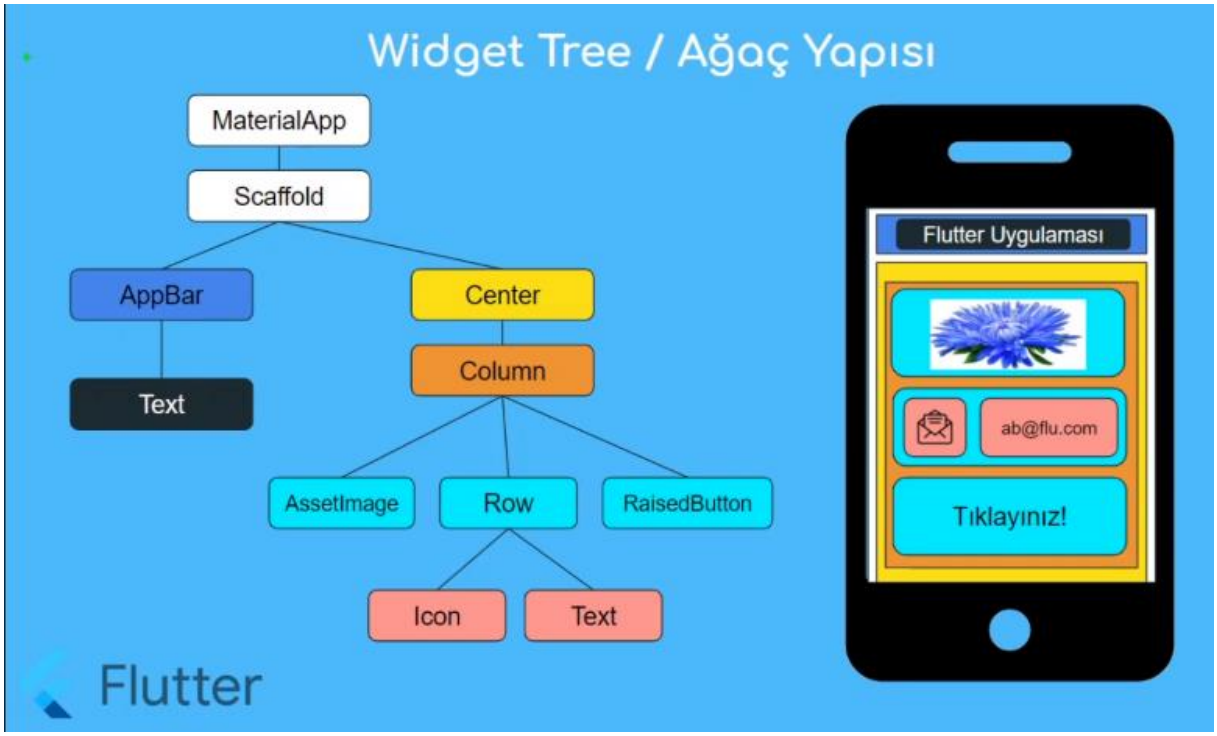
FLUTTER

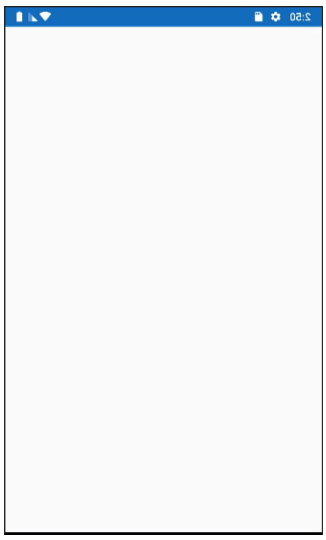


MaterialApp, Google tarafından uygulamaların belirli bir formatta olması için geliştirilmiş bir dizayn konsepti, sınıf, widget. Material.io sitesinde gerekli bilgilere erişebilirsiniz.

MaterialApp widgetı materyal tasarım widgetlarını kullanmamızı sağlar. Materyal tasarımda bir app oluşturmak için MaterialApp widgetını ekledikten sonra ilk iş Scaffold widgetını eklemektir.

Scaffold: Scaffoldun anlamı inşaat iskelesidir. Materyal tasarımda uygulamanın hangi ana bölümlerden oluşacağı ve bu bölümlerin yerleri standarttır. Çok kullanılan arayüz elemanlarını sunan bir widgettır. App bar, asıl içerik için bir gövde, ekran altında bir buton, ekran altında bir bar gibi. AppBarın, menünün, sayfa içeriğinin, nerede olması gerektiği genel hatlarıyla bellidir. Bu genel hatlları Scaffold widgetı belirler.

Uygulamayı boş beyaz bir alan olarak düşünürsek Scaffold eklediğimizde uygulamayı birçok bölüme ayıran görünmez bir yapı eklemiş oluruz. Bu bölünmüş alanlara widgetlar ekleyerek uygulamamızı oluştururuz.



runApp() fonksiyonu: Void bir fonksiyon, yani herhangi bir değer döndürmez. Widget tipinde bir parametre alır. Verilen Widget'ı ekrana getirir.

Fonksiyonlarda parantezler içinde iken CTRL+Boşluk tuşuna basıldığında o sınıf yada fonksiyonun alabileceği parametreler kaşımıza gelir.

```
runApp();
```

Container() : Türkçedeki anlamı gibi içine bir şeyler koyabildiğimiz bir kutu sınıfı, widgetı.

```
runApp(Container());
```

Bu kutunun boyutu, rengi, içindekilerinin hizalanması vs birçok şeyi istediğimiz gibi ayarlayabiliriz. Bu özellikleri parametre olarak gönderiyoruz. Containerın hangi parametreleri aldığını mousela üzerine geldiğimiz zaman açılan pencereden görebiliriz.

Köşeli parantezler içinde yazılı olanlar parametreler alabileceği **zorunlu olmayan parametrelerdir**.

Örneğin renk vermek için color objesini parametre olarak göndermemiz gerekir.

```
runApp(Container(color: 'red',));    Burada red parametresi hatalı olur üzerine geldiğimizde string tipinde olduğunu ve Color tipinde bir parametre vermemiz geretiği uyarısını gösterir.

runApp(Container(color: Colors.amber,));   Colors içinde farklı renklerin olduğu bir obje/kutu. Bu kutudan renk veriyoruz. Colors. Dediğimizde renkler karşımıza geliyor.
```

child parametresi: Bir widget içerisine başka bir widget koymamızı sağlar. En çok kullanılan parametrelerden biridir.

Örneğin contanier widgetının içerisine child parametresi ile bir widget (Text) daha ekleyelim.

```
runApp(Container(color: Colors.amber,child: Text(data),));
```

```
Text(): Bu widget yazı eklememizi sağlar. Text widget üzerine geldiğimizde String türünde bir parametre eklememizin zorunlu olduğunu gösterir.

runApp(Container(color: Colors.amber,child: Text("Mobil", textDirection: TextDirection.ltr,),));

String parametre olarak "Mobil" verdikten sonra, yazının yönünüde belirtmemiz gerekiyor. Bunun içinde textDirection parametresi ile yazı yönünü soldan sağa yada sağdan sola olacak şekilde belirtiyoruz.

Yazının rengi, boyutu gibi özelliklerini değiştirmek içi style parametresini kullanmamız gerekir. style parametresi TextStyle objesi alıyor. TextStyle objesinin color ve fontsize parametrelerini kullanarak yazı özelliklerini değiştirebiliriz.

runApp(Container(color: Colors.amber, child: Text("Mobil", textDirection: TextDirection.ltr, style: TextStyle(color: Colors.blue, fontSize: 35,),),));
```

Center(): Bu widget, içine aldığı widgetın ekran ortasına yerleşmesini sağlar.

Yukarıdaki Text widgetındaki Mobil yazısını ekran ortasına almak için Text widgetını Center widgetının içine almak gerekir.

```
runApp(Container(
  color: Colors.amber,
  child: Center(
    child: Text(
      "Mobil",
      textDirection: TextDirection.ltr,
      style: TextStyle(
        color: Colors.blue,
        fontSize: 35,
      ),
    ),
  ),
));
```

WidgetLar SABİT ve DİNAMİK olma üzere ikiye ayrılır;

Stateless Widget: Sabit, durum içermeyen, değişmeyen Widget. Sabit Widget programa eklendiği zaman eklendiği gibi kalır. Örneğin bir ayakkabı aldınız. Bunun üretimden sonra, rengi, malzemesi şekli değişmez.

Şu ana kadar kullandığımız Container, Text widgetlar sabit widgetlardır.

Widgetlar bir sınıftır. Yani bir widget oluşturmak istiyorsak bir sınıf oluşturmamız gerekir.

Yukarıdaki örneğimizde Container Widgetın tamamını yeni bir Widget olarak tanımlamak istersek;

Bir tane Stateless widget oluşturup onun içine alabiliriz. Ve runApp() içerisinde bu widgetı çağırabiliriz.

```
class Yaz extends StatelessWidget {

  const Yaz({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Container(
      color: Colors.amberAccent,
      child: Center(
        child: Text(
          'Denemeeee',
          textDirection: TextDirection.ltr,
          style: TextStyle(color: Colors.black, fontSize: 50),
        ),
      ),
    );
  }
}
```

Başka bir Örnek:

Kisi adında bir sabit sınıf oluşturalım.

Stateless widgetta tanımlanan nitelikler (değişkenler) bir kez tanımlanır ve değişmezler bu nedenle final olarak tanımlanmalıdır.

```
class Kisi extends StatelessWidget {
  //const Kisi({Key? key}) : super(key: key);

  final String ad;
  final int yas;
  Kisi(this.ad, this.yas);

  @override
  Widget build(BuildContext context) {
    return Text("Ad : $ad  -- Yaş : $yas",textDirection: TextDirection.ltr,);
  }
```

`Kisi(this.ad, this.yas)` // Kisi sınıfının yapıcısı. Gönderilen parametreleri niteliklere kaydeder.

build metodu çalıştığı zaman return ile belirttiğimiz widgetları ekrana getirir. Return ile içiçe birden fazla widgetta döndürülebilir.

Buildcontext, yazdığımız widgetın, widget ağacındaki konumudur.

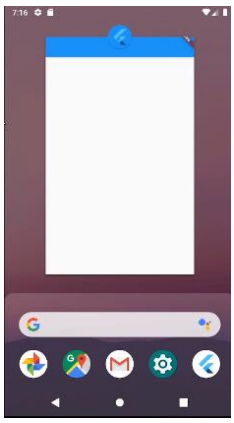
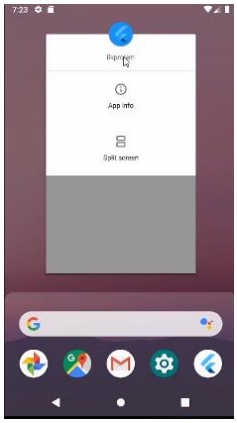
Yeni bir DEMO proje oluşturalım. DEMO projenin hızlı bir şekilde çalışmasından bahsedelim.

Sonra

Oluşturulan projedeki MyHomePage sınıfından başlamak üzere alttaki kodları silelim. MyApp sınıfında en sondaki home: kısmından MyHomePage sınıfını silip Scaffold widgetı ve AppBar ekleyelim, aşağıdaki gibi değiştirelim.

```
import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(

        primarySwatch: Colors.blue,
      ),
      home: Scaffold(appBar: AppBar(),),
    );
  }
}
```

Uygulamayı çalıştıralım. Emulatör de tüm uygulamaların listelendiği ekranı kare tuşuna basarak getirelim. Uygulamamızın başlık kısmına bakarsak flutter ikonunu ve mavi renkli başlık barını görürüz.

title: MaterialAppın title parametresine girilen değer, uygulama başlığında çıkması gereken ifadeyi belirtiyor. Ama bu ifadenin görünmesi android sürümüne göre değişiyor. Bizim bu sürümümüzde ekrana gelmiyor. Başlıktaki ikona tıkladığımızda ise title yerine uygulama ismini görüyoruz.

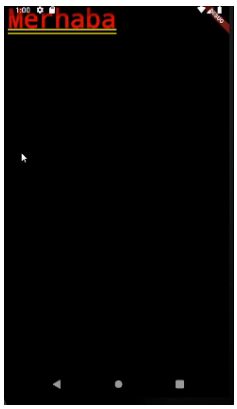
theme: Bu parametre Materyal widgetlar için renkler, yazı tipleri, şekiller gibi görsel özellikleri belirler. Bu parametre ile ThemeData objesi gönderiyoruz.

ThemeData widgetındaki, **primarySwatch:** paremetresi ile materyal rengini belirliyoruz.

Home: Uygulama çalıştırıldığında ilk çalışacak ve kullanıcının karşısına gelecek ilk widgetı belirler.

Örneğimizdeki **Scaffold widgetı** silip yerine **Text widget** koyarak sonucu tekrar görelim.

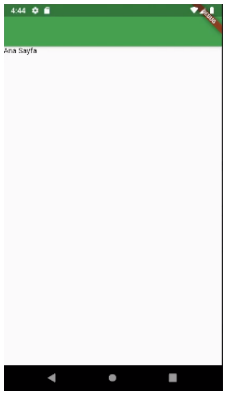
```
home: Text("Mobil"),
```



home parametresini eski haline getirelim ve **Scaffold widgetının** diğer bölümlerini uygulamamıza ekleyelim.

body: Bu parametre ile sayfanın ana içeriği eklenir.

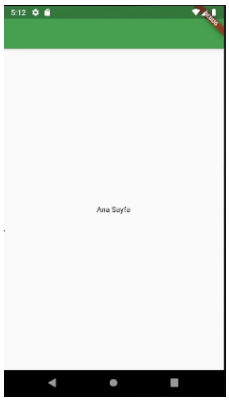
```
body: Text("Ana Sayfa"),
```



body parmeteresi ile ekrana yazılan Ana Sayfa yazısını ortaya almak için **Center widgetı** kullanmalıyız.

```
body: Center(child: Text("Ana Sayfa")),
```

Text widgettın üzerindeyken soldaki sarı ampül işaretine tıklayıp **Wrap with Center** seçersek otomatik olarak Center widget eklenir.



Stateless widget Örneği:

Android Studioda stl yazdığımızda Stateless Widget eklemek için seçenek çıkar. Tıkladığımızda yeni bir stateless widget yapısı koda eklenir.

```
class ? extends StatelessWidget {
  const ({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```

? işareti kısmına oluşturmak istediğimiz widgetın ismini yazarız. Örneğin **“Yaz”.** Sonra body parametresindeki Text widgetını, bu widget ile değiştiririz.

```
body: Center(child: Yaz()),
```

Yaz widgetın parametre alabilmesi için bir değişken ve gönderilen parametreyi alması için bir yapıcı ekleriz. Ve Text widgetına parametre değişkeninin ismini yazarız.

```dart
class Yaz extends StatelessWidget {

  final String parametre;
  Yaz(this.parametre);
  @override
  Widget build(BuildContext context) {
    return Text(parametre);
  }
}
```

Sonra body parametresindeki Yaz widgetına parametre için değer yazarız.

```dart
body: Center(child: Yaz("Ana Sayfa") ),
```

Çalıştırdığımız zaman Ana Sayfa bilgisinin önceki ile aynı şekilde ortada çıktığını görürüz. Tüm bunların sonucunda bir stateless widget oluşturarak kullanmayı öğrendik.

Şimdi daha önce oluşturduğumuz Kisi adlı stateless widgetını kullanarak aynı uygulamamızı tekrar deneyelim.

```dart
body: Center(child: Kisi("Ali","AK",35) ),
```

Column(): Birden fazla Kisi bilgilerini ana sayfada göstermek istersek Center widgetının **child parametresini** kullanamayız. Çünkü child parametresi tek bir widget alır. Bunun yerine Column widgetını kullanırız. Column widgetının children parametresi birden fazla widget alabilir.

children: Bu parametreye liste içerisinde birden fazla widget gönderebiliriz.

```dart
import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.red,
      ),
      home: Scaffold(
        appBar: AppBar(),
        body: Column(
          children: [
            Kisi("Ali", "ABC", 30),
            Kisi("Hasan", "XYZ", 25)
          ],
        ),
      ),
    );
  }
}
```

```
class Yaz extends StatelessWidget {
  final String parametre;
  Yaz(this.parametre);
  @override
  Widget build(BuildContext context) {
    return Text(parametre);
  }
}

class Kisi extends StatelessWidget {
  final String ad;
  final String soyad;
  final int yas;

  Kisi(this.ad, this.soyad, this.yas);

  @override
  Widget build(BuildContext context) {
    return Text("Ad: $ad Soyad: $soyad Yaş: $yas");
  }
}
```

Stateful widget Örneği:

Stateful widget, iki bölümden oluşur. İlk bölüm stateless widget kısmı, ikinci bölüm ise state kısmı.

Oluşturmak için android studioda stl yazıp açılan listeden statefulwidget seçilerek yapının otomatik olarak gelmesi sağlanır. Sonra bir isim girilerek tamamlanır.

Bir önceki örneğimizdeki MyApp widgetın home parametresindeki Scaffold kısmını sileriz. Buraya oluşturacağımız Sayac widgetını yazarız.

```
home: Sayac("AAA",),
```

Sayac widgetınıda aşağıdaki gibi değiştiririz. State bölümündeki return kısmındaki Container widgetını silip ekranımızda olması gereken Scaffold widgetını buraya ekleriz.

```
class Sayac extends StatefulWidget {
  final String adi;

  Sayac(this.adi);

  @override
  _SayacState createState() => _SayacState();
}

class _SayacState extends State<Sayac> {
  int sayi = 0;
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(),
      body: Text("${widget.adi} sayısı: $sayi"),
    );
  }
}
```

Şimdi uygulamamıza bir buton koyalım ve sayaç işlemini gerçekleştirelim

Bunun için uygulama ekranımızın **body parametresine** bir buton eklemesi yapıyoruz.

floatingActionButton: Bu paramatreye bir tane **FloatingActionButton widgetı** ekleriz.

```
body: Text("${widget.adi} sayısı: $sayi"),
floatingActionButton: FloatingActionButton(onPressed: onPressed),
```

onpressed: bu parametreye butona tıklandığı zaman çalışacak fonksiyonu yazmamız gerekir.

```
class _SayacState extends State<Sayac> {
  int sayi = 0;
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(),
      body: Text("${widget.adi} sayısı: $sayi"),
      floatingActionButton: FloatingActionButton(onPressed: (){setState(() {
      sayi++;
      });
      }),
    );
  }
}
```

butonda kullanılan anonim fonksiyon **yerine ayrı bir fonksiyon** kullanabiliriz.

```
class _SayacState extends State<Sayac> {
  int sayi = 0;
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(),
      body: Text("${widget.adi} sayısı: $sayi"),
      floatingActionButton: FloatingActionButton(onPressed: ekle),
    );
  }
  void ekle(){
    setState(() {
      sayi++;
    });
  }
}
```