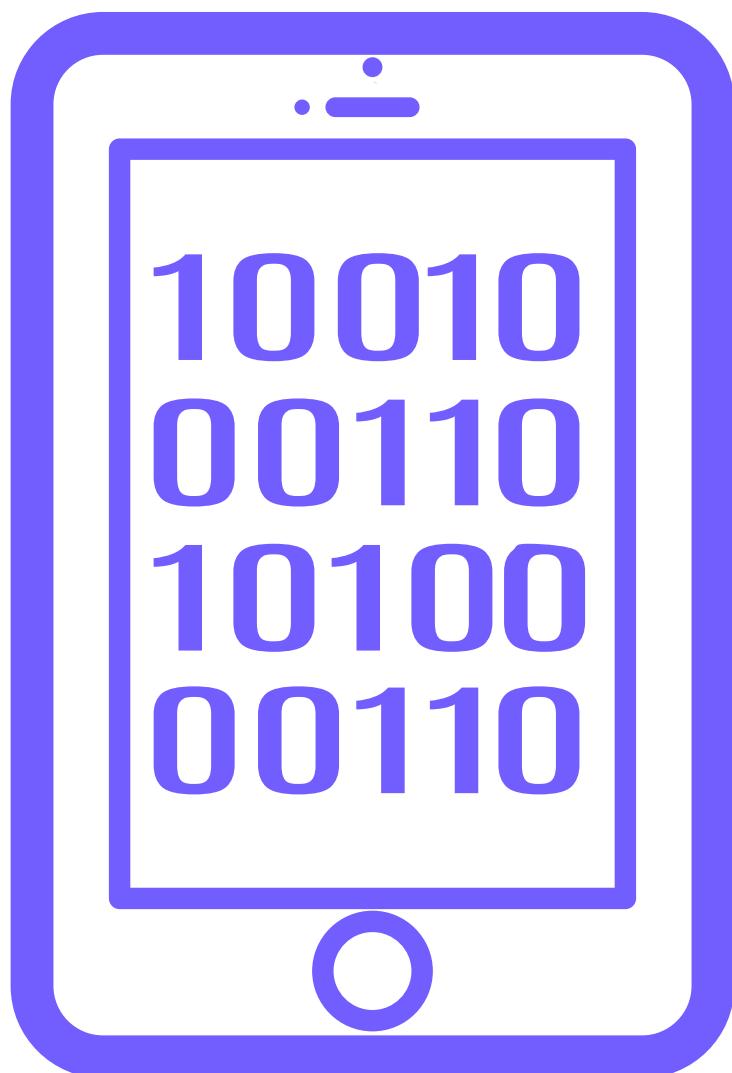


# MOBİL UYGULAMA

## LİSE



***Mobil Uygulama***  
LİSE

Doç. Dr. Onur DÖNMEZ

© Türkiye Bilimsel ve Teknolojik Araştırma Kurumu, 2023

Bu kitabın bütün hakları saklıdır.  
Yazılı ve görsel malzemeler, TÜBİTAK'tan yazılı izin alınmadan  
tümüyle veya kısmen çoğaltılamaz ve yayımlanamaz.  
Kitabın PDF formatındaki elektronik nüshasına  
“<https://yayinlar.tubitak.gov.tr/deneyap-atolyesi>” adresinden ulaşılabilir.  
TÜBİTAK Deneyap Kitapları *DENEYAP TÜRKİYE* Projesi kapsamında hazırlanmıştır.

ISBN: 978-605-312-531-0  
Yayınçı Sertifika No: 47703

Yayım Tarihi: Ekim 2023

TÜBİTAK Başkanı: Prof. Dr. Hasan MANDAL  
Bilim ve Toplum Başkanı: Ömer KÖKÇAM  
Genel Yayın Yönetmeni: Fatma BAŞAR  
Editörler: Havva Hilal KAÇAR, Doç. Dr. Şahin İDİL  
Düzeltilti: Dr. Mustafa ORHAN  
Telif İşleri Sorumlusu: Havva Hilal KAÇAR

TÜBİTAK Bilim ve Toplum Başkanlığı  
Tunus Caddesi No: 80 Kavaklıdere 06680 Ankara  
Tel: (312) 298 96 50  
e-posta: [deneyap@tubitak.gov.tr](mailto:deneyap@tubitak.gov.tr)  
<https://yayinlar.tubitak.gov.tr/deneyap-atolyesi>

**DENEYAP**  
Teknoloji Atölyeleri

**MOBİL UYGULAMA**  
**LİSE**

Doç. Dr. Onur DÖNMEZ



## İçindekiler

<b>Sunuş .....</b>	<b>5</b>
<b>1. Hafta: Mobil Uygulamalar ve Flutter.....</b>	<b>6</b>
<b>Ön Bilgi: .....</b>	<b>6</b>
<b>Haftanın Kazanımları: .....</b>	<b>6</b>
<b>Haftanın Amacı: .....</b>	<b>6</b>
<b>Kullanılacak Malzemeler:.....</b>	<b>6</b>
<b>Ders Öncesi Gereksinimler.....</b>	<b>6</b>
<b>Haftanın İşlenisi: .....</b>	<b>7</b>
<b>1. GÖZLE VE UYGULA.....</b>	<b>8</b>
1.1.    Gözle: Mobil Uygulama Ekosistemi .....	8
1.2.    Gözle: Sektör Eğilimleri .....	8
1.3.    Uygula: Kullandığımız Mobil Uygulamalar?.....	10
1.4.    Gözle: Güncel Mobil Uygulama Geliştirme Eğilimleri.....	10
1.5.    Uygula: Hangi Eğilimleri Takip Etmeliyiz?.....	11
1.6.    Gözle: Mobil Uygulama Geliştirme Yaklaşımları .....	11
1.7.    Uygula: Mobil Uygulama Geliştirme Yaklaşımlarını Karşılaştırıyalım .....	12
<b>2. TASARLA ve ÜRET.....</b>	<b>13</b>
<b>3. GÖZLE VE UYGULA.....</b>	<b>14</b>
3.1.    Gözle: Flutter Nedir? .....	14
3.2.    Gözle: Flutter'ın Hedeflediği Platformlar .....	15
3.3.    Gözle ve Uygula: İlk Flutter Uygulamanız.....	15
3.4.    Gözle ve Uygula: İlk uygulamayı WebApp Olarak Koşturmak .....	16
3.5.    Gözle: Flutter Uygulamalarındaki Önemli Dosyalar .....	18
3.6.    Gözle: main.dart Dosyasının Anatomisi .....	20
3.7.    Uygula: main.dart Dosyasını Düzenleyelim .....	22
<b>4. DEĞERLENDİR .....</b>	<b>23</b>
<b>5. İLAVE ETKİNLİK.....</b>	<b>23</b>
5.1.    Tasarımınızı Geliştirin .....	23
<b>6. KAYNAKÇA .....</b>	<b>24</b>
<b>2. Hafta: Dart Programlama Dili .....</b>	<b>25</b>
<b>Ön Bilgi: .....</b>	<b>25</b>
<b>Haftanın Kazanımları: .....</b>	<b>25</b>
<b>Haftanın Amacı: .....</b>	<b>25</b>
<b>Kullanılacak Malzemeler:.....</b>	<b>25</b>
<b>Ders Öncesi Gereksinimler.....</b>	<b>25</b>
<b>Haftanın İşlenisi: .....</b>	<b>26</b>

<b>1. GÖZLE VE UYGULA.....</b>	<b>26</b>
1.1. Gözle: Temel Kavramlar .....	26
1.2. Gözle ve Uygula: Dart Uygulamaları.....	27
1.3. Gözle ve Uygula: Yorum Satırları .....	28
1.4. Gözle ve Uygula: İşleçler (Operatörler) .....	29
1.5. Gözle: Anahtar Sözcükler .....	34
1.6. Gözle ve Uygula: Değişkenler .....	34
1.7. Gözle ve Uygula: Null-Safety Mekanizması .....	35
1.8. Gözle ve Uygula Öntanımlı Veri Tipleri.....	37
1.9. Gözle ve Uygula: Akış Yöneticiler .....	48
1.10. Gözle ve Uygula: Fonksiyonlar .....	54
1.11. Gözle ve Uygula: Etki Alanları.....	56
1.12. Dart ile Nesneye Yönelik Programlama .....	59
<b>2. TASARLA ve ÜRET.....</b>	<b>64</b>
<b>3. DEĞERLENDİR .....</b>	<b>64</b>
<b>4. İLAVE ETKİNLİK.....</b>	<b>64</b>
<b>5. KAYNAKÇA.....</b>	<b>65</b>
<b>3. Hafta: Bileşenler .....</b>	<b>66</b>
<b>Ön Bilgi: .....</b>	<b>66</b>
<b>Haftanın Kazanımları: .....</b>	<b>66</b>
<b>Haftanın Amacı: .....</b>	<b>66</b>
<b>Kullanılacak Malzemeler:.....</b>	<b>66</b>
<b>Haftanın İşlenisi: .....</b>	<b>66</b>
<b>1. GÖZLE VE UYGULA.....</b>	<b>67</b>
1.1. Gözle: Bileşen (Widget) Kavramı .....	67
1.2. Uygula: Temel Flutter Uygulamasındaki Bileşenleri İnceleyelim.....	68
1.3. Gözle: Simgeler.....	69
1.4. Uygula: Simge türleri.....	70
1.5. Uygula: semanticLabel parametresi .....	70
1.6. Text Bileşeni .....	71
1.7. Gözle ve Uygula: GoogleFonts Paketini Kullanmak .....	77
1.8. Gözle ve Uygula: Temaları Kullanmak .....	79
1.9. Resimler.....	82
<b>2. TASARLA ve ÜRET.....</b>	<b>86</b>
<b>3. DEĞERLENDİR .....</b>	<b>88</b>
<b>4. İLAVE ETKİNLİK.....</b>	<b>88</b>
<b>5. KAYNAKÇA.....</b>	<b>88</b>
<b>4. Hafta: Yayılmış Bileşenleri ve Arayüz Tasarımı.....</b>	<b>89</b>
<b>Ön Bilgi: .....</b>	<b>89</b>
<b>Haftanın Kazanımları: .....</b>	<b>89</b>
<b>Haftanın Amacı: .....</b>	<b>89</b>

<b>Kullanılacak Malzemeler:</b> .....	<b>89</b>
<b>Haftanın İşlenisi:</b> .....	<b>89</b>
<b>1. GÖZLE ve UYGULA .....</b>	<b>90</b>
1.1.    Gözle: Tek Elementli Kapsayıcı Bileşenler .....	90
1.2.    Gözle: Çok Elementli Kapsayıcı Bileşenler .....	96
1.3.    Gözle: Kapsayıcı Elementlerle Arayüzler Oluşturmak .....	108
<b>2. TASARLA ve ÜRET.....</b>	<b>111</b>
<b>3. DEĞERLENDİR .....</b>	<b>111</b>
<b>4. İLAVE ETKİNLİK.....</b>	<b>112</b>
<b>5. Hafta: Etkileşimli Uygulamalar .....</b>	<b>113</b>
<b>Ön Bilgi:</b> .....	<b>113</b>
<b>Haftanın Kazanımları:</b> .....	<b>113</b>
<b>Haftanın Amacı:</b> .....	<b>113</b>
<b>Kullanılacak Malzemeler:</b> .....	<b>113</b>
<b>Haftanın İşlenisi</b> .....	<b>113</b>
<b>1. GÖZLE VE UYGULA.....</b>	<b>114</b>
1.1.    Gözle: Kompozit Bileşenler.....	114
1.2.    Durum (state) Kavramı .....	118
1.3.    Gözle: http İstekleri .....	125
1.4.    Gözle: Çok Sayfalı Uygulamalar .....	132
1.5.    Gözle: Drawer (Çekmece Bileşeni) .....	138
1.6.    Gözle: İsimli Rota Kullanımı .....	140
1.7.    Gözle: Rotalara Veri Aktarmak .....	143
1.8.    Gözle: İsimli Rotalara Veri Göndermek .....	148
<b>2. TASARLA ve ÜRET.....</b>	<b>150</b>
<b>3. DEĞERLENDİR .....</b>	<b>150</b>
<b>4. İLAVE ETKİNLİK.....</b>	<b>150</b>
<b>5. KAYNAKÇA.....</b>	<b>151</b>
<b>6. Hafta: Form Uygulamaları .....</b>	<b>153</b>
<b>Ön Bilgi:</b> .....	<b>153</b>
<b>Haftanın Kazanımları:</b> .....	<b>153</b>
<b>Haftanın Amacı:</b> .....	<b>153</b>
<b>Kullanılacak Malzemeler:</b> .....	<b>153</b>
<b>Haftanın İşlenisi</b> .....	<b>153</b>
<b>1. GÖZLE VE UYGULA.....</b>	<b>154</b>
1.1.    Gözle: Flutter Dokümantasyonu .....	154
1.2.    Gözle: Form Uygulaması.....	156
<b>2. TASARLA ve ÜRET.....</b>	<b>171</b>

3.	DEĞERLENDİR .....	171
4.	İLAVE ETKİNLİK.....	171
5.	KAYNAKÇA.....	172

## Sunuş

Mobil cihazlar ve giyilebilir teknolojiler günlük hayatımızda devrim yaratıyor. Kameralar, konumlama sistemleri, vücut verilerini toplayabilen algılayıcılar ve yakın alan iletişim gibi donanımlar mobil cihazların yeteneklerini günden güne arttırıyor. Fakat bu donanım birimleri, kendilerini yöneten uygulamalar olmadan bizlere hizmet veremiyor. Mobil cihazlar üzerinde koşturulan mobil uygulamalar, cihazların yeteneklerine erişim kapımız niteliğinde. Bu uygulamalarla eğitim, sosyalleşme, alışveriş, bankacılık, spor, sağlık, oyun, eğlence gibi gereksinimlerimizi yer ve zamandan bağımsız şekilde gerçekleştirebiliyoruz. Günlük hayatımızın bir parçası olan mobil uygulamalar doğal olarak önemli bir pazar da oluşturuyor. Pek çok şirket ve girişimci devasa mobil uygulama pazarından pay alabilmek için yarışıyor.

Mobil uygulama geliştirmek için kullanılabilecek birçok strateji ve onlarca araç bulunabilir. Bu kitapta, ilerleyen yıllarda adından sıkıkla bahsedilmesini umduğumuz Flutter uygulama geliştirme ekosistemi temel alındı. Google firması tarafından geliştirilen Flutter, pek çok ortamda koşturulabilen uygulamalar geliştirmek için kullanılabiliriyor. Kendine has geliştirme pratiklerine sahip olan çerçeveyin temel uygulamalarını altı haftaya yayılan bir programda açıklamaya çalıştık.

Birinci Bölüm Mobil Uygulama Geliştirme sektörüne genel bir bakış sunarak başlıyor. Ardından Flutter uygulama geliştirme çerçevesinin tanıtımı ve bilgisayarlarla kurulum işlemleri aktarılıyor. İkinci bölümde Dart programlama dilinin temel mekanikleri sunuluyor. Üçüncü bölüm ara yüzlere içerik eklemek için kullanılabilecek metinler, simgeler ve fotoğraflar gibi temel bileşenlere ayrıldı. Dördüncü bölümde ise Flutter ile ara yüzlerin düzenlenmesine olanak veren yayılım bileşenleri ve ara yüz tasarımlarına değineceğiz. Etkileşimli ve çok sayfalı uygulamaların üretiminin sunulduğu beşinci bölümün ardından, kullanıcılarından veri alabilen form uygulamalarının geliştirilmesi ile altıncı ve son bölüm bitireceğiz.

Öğretim tasarımindı Deneyap atölyelerinin diğer derslerinde de kullanılan Gözle, Uygula, Tasarla, Üret ve Değerlendir (GÜTUD) döngüsünü temel aldık. Bu döngüde Gözle basamağında temel bilgi ve uygulamaların öğrencilerle incelenmesinin ardından, Uygula basamağında çözümü örnekler üzerinde çalışacağız. Her üitede özgün çözümler üretmenizi beklediğimiz problemlerle Tasarla, Üret ve Değerlendir basamaklarını işleteceğiz. Bu problemler derste anlatılanların ötesinde bilgiler edinmenizi ve bağlantılar keşfetmenizi sağlayacak. Bölüm sonlarındaki İlave Etkinliklerle öğrenciklerinizi yeni durumlara transfer etmenizi bekliyoruz. GÜTUD temel bilgilerin sunulması yanında, bu bilgilerin gerçek problemlere uygulanmasını da içerdiginden derslerin verimini artıracaktır.

Mobil uygulama geliştirme dünyasına giriş niteliğindeki bu kitapta, mobil uygulama sektörünü, mobil uygulama geliştirme araçlarını ve süreçleri tanıtmayı amaçladık. Alanın dinamik doğası gereği bu kitap bir referans kaynaktan çok, giriş niteliğinde bir ders içeriği olarak ele alınmalı. Fakat kitabın Flutter ile uygulama geliştirmeye giriş için yeterli olacağını umuyoruz.

Tüm paydaşlarımıza faydalı olması dileğiyle.

Doç. Dr. Onur Dönmez

İzmir - 2023

# 1. Hafta: Mobil Uygulamalar ve Flutter

## Ön Bilgi:

- Temel programlama bilgisi
- Kullanılan işletim sisteminde uygulama yükleme bilgisi
- Temel internet kullanım bilgisi

## Haftanın Kazanımları:

- Mobil uygulama kavramını tanımlar.
- Mobil uygulama geliştirme süreçlerini açıklar.
- Mobil uygulama geliştirmede güncel eğilimleri açıklar.
- Çok ekranlı bir mobil uygulamanın arayüzüünü ve fonksiyonlarını tasarlars.
- Flutter uygulama geliştirme ortamını tanır.
- Bir Flutter uygulaması oluşturur.
- Flutter uygulamalarındaki dosyaların temel görevlerini açıklar.
- Flutter uygulamalarını Web tarayıcıda çalıştırır.
- Ekranda “Merhaba!” yazan bir Flutter uygulaması üretir.

## Haftanın Amacı:

Bu haftanın amacı öğrencilerin mobil uygulama kavramını algılaması ve Flutter kullanarak ürettikleri mobil uygulamaları Web tarayıcı üzerinde çalıştırımlarının sağlanmasıdır. Bu amaçla öncelikle mobil uygulama kavramı, mobil uygulama sektörünün büyülüğu ve mobil uygulama geliştirme süreçleri hakkında temel bilgiler sunulacaktır. Ardından öğrencilerden bir mobil uygulama tasarlamları ve bu uygulamanın arayüzü üremeleri istenecektir. Son olarak Google Flutter ekosistemi tanıtılarak, bu araç ile bir mobil uygulama üremeleri istenecektir.

## Kullanılacak Malzemeler:

Bilgisayar, internet, proto.io internet sitesi

## Ders Öncesi Gereksinimler

1. Bu derste anlatılacak uygulamalar için öğrencilerin bilgisayarlarında Flutter uygulama geliştirme ortamının kurulu olması gerekmektedir. Bu amaçla Windows ve MacOS işletim sistemlerinde yükleme işlemlerinin anlatıldığı videolar hazırlanmıştır. Öğrencilerin ders öncesinde bu videoları izleyerek kurulumları tamamlamış olmaları gerekmektedir.

**Windows:** [https://youtu.be/-k9iz\\_XR7A](https://youtu.be/-k9iz_XR7A)

### MacOS:

2. Dersdeki tasarım etkinliği için proto.io internet sitesi kullanılacaktır. Ders öncesinde öğrencilerden bu siteye kayıt yaptırmaları gerekmektedir.

Kayıt formu adresi: <https://proto.io/en/signup/>

3. Bu derste projelerin yönetimi ve kod yazma işlemleri için VSCode düzenleyicisinin kullanılması önerilmektedir. Ücretsiz dağıtılan bu düzenleyici <https://code.visualstudio.com/download> adresinden edinilebilir.
4. Öğrencilerin eğitim başındaki programlamaya yönelik tutumlarını belirlemek için <https://forms.gle/pwN8E2Lato53iue67> adresinde yer alan formu uygulayınız. Bu form öğrencilerin e-posta bilgilerini toplamaktadır. Eğitim sonunda aynı form tekrar uygulanacaktır. Verilerin eşleştirilmesi amacıyla, öğrencilerinize her iki uygulamada da aynı e-posta adresini kullanmaları gerektiğini bildiriniz.

## **Haftanın İşlenisi:**

**Gözle:** Mobil uygulama nedir, sektör büyülüğu ne kadardır, mobil uygulama geliştirme eğilimleri nelerdir?

**Uygula:** Flutter ile bir mobil uygulama üreterek Web tarayıcıda çalıştır.

**Tasarla:** Hayalindeki mobil uygulamanın arayüzüne tasarıla.

**Üret:** Ekrana “merhaba!” yazan bir mobil uygulama üret.

**Değerlendir:** Haftanın içeriği ile ilgili yansıtma etkinliği.

# 1. GÖZLE VE UYGULA

## 1.1. Gözle: Mobil Uygulama Ekosistemi

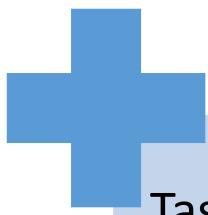
### Tartışma

Mobil cihazların ne gibi avantajları ve dezavantajları vardır?

Bu avantaj ve dezavantajların mobil uygulama geliştirme sürecine yansımaları nelerdir?

Mobil uygulamalar, akıllı telefonlar, tabletler ve akıllı saatler gibi küçük, taşınabilir ve kablosuz cihazlar için özel üretilmiş yazılımlardır. Mobil uygulama geliştirme süreci bu cihazlara özgü uygulamaların yazılması, denenmesi ve iyileştirilmesini kapsar.

Diğer uygulama geliştirme süreçlerinde olduğu gibi, mobil uygulama geliştirme de temel yazılım geliştirme süreçlerinden ve mekaniklerinden beslenir. Bununla birlikte, mobil uygulama geliştirme sürecinde bu cihazların avantajları ve dezavantajları dikkate alınır. Örnek vermek gerekirse, masaüstü bilgisayarların işlem güçleri çoğunlukla mobil cihazlara oranla çok daha yüksektir. Bunun en önemli nedeni mobil cihazlardaki işlemcilerin kısıtlı batarya kapasitesini etkin kullanma amacıdır. Bununla birlikte, masaüstü bilgisayarlar genellikle mobil cihazlardaki kablosuz bağlantı özellikleri ya da parmak izi okuyucu ve kamera gibi yerleşik algılayıcılara sahip değildir. Bu nedenle mobil uygulama geliştiricilerin yüksek işlemci gücü ve pil tüketiminden uzak durmaları gereklidir. Öte yandan, mobil cihazların yetenekleri sayesinde mobil bir uygulamada kullanıcı yetkilendirmesi parmak izi okuma ya da yüz tanıma yöntemleri ile yapılabilir.

	<p>Taşınabilirlik ve sürekli erişim Kablosuz bağlantı özellikleri Yerleşik algılayıcılar Nesnelerin interneti</p>	<p>Düşük işlem gücü Batarya gereksinimi Yüksek fiyatlar Aşırı değişken cihaz özellikleri</p>
---	---	--

## 1.2. Gözle: Sektör Eğilimleri

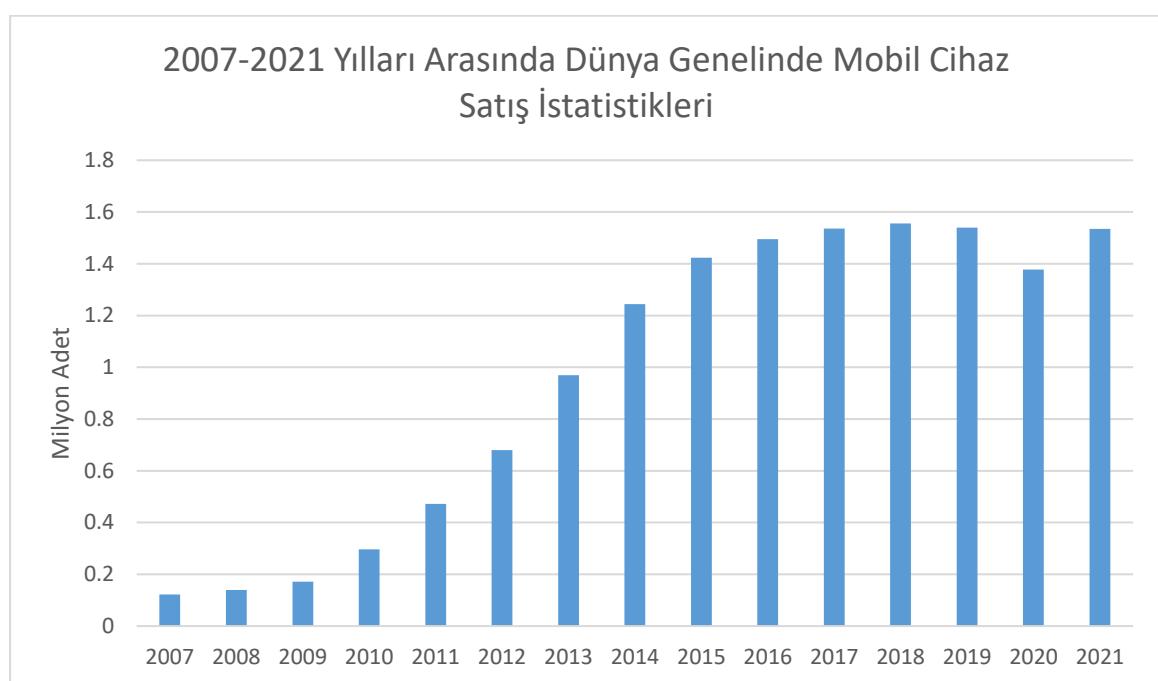
Mobil uygulamalar genellikle hedeflenen işletim sistemlerine özgü paketler (Ör: apk, app) haline getirilerek dağıtılmaktadır. Bu işlem, mobil işletim sistemlerine özgü uygulama mağazaları üzerinden gerçekleştiriliyor. Günümüzde mobil uygulama sektöründe baskın iki işletim sistemi

ve buna bağlı olarak da iki uygulama geliştirme mağazası bulunmaktadır. Güncel verilere göre mobil uygulama piyasasının lideri yaklaşık %72'lik pazar payı ile Google firmasının Android işletim sistemi ve bu işletim sistemine özgü uygulama mağazası Play Store'dur. Bu sektörün ikinci büyük temsilcisi ise yaklaşık %26'lık pazar payı ile Apple firmasının iOS işletim sistemi ve App Store uygulama mağazasıdır. Bu iki firma dışında büyük firmalarca (ör: Microsoft, Samsung) pek çok girişim başlatılsa da bu girişimler piyasadan beklediği ilgiyi görememiştir. Kalan %2'lik pazar payının bu girişimler arasında dağıldığı düşünülebilir.

### Not

Bu bölümün yazıldığı dönemde Apple firması tablet bilgisayarlara özgü iPad OS işletim sistemini ve M işlemci ailesini duyurmuştur. M işlemci ailesi sayesinde, AppStore uygulamaları hem masaüstü cihazlarda hem de mobil cihazlarda kullanılabilecektir.

Mobil cihazlar ciddi bir kullanıcı kitlesine sahiptir. Grafik 1, 2007 – 2021 yılları arasında dünya genelinde mobil cihaz satışlarını göstermektedir.



Grafik 1. 2007-2021 yılları arasında dünya genelinde mobil cihaz satış istatistikleri (Kaynak: O'dea, 2021)

Statista 2021 yılında 1,59 milyon mobil cihazın satılacağını tahmin etmektedir. 2020 yılında bu pazarın liderleri Samsung (%22,7), Huawei (%14,6), Xiaomi (%13,1) ve Apple (%10,5) markaladır. Bu cihazlar üzerinde koşturulan uygulamaların ticareti olarak düşünülebilecek mobil uygulama pazarı oldukça büyük bir ekonomidir. App Annie firmasının tahminlerine göre 2020 yılında mobil uygulama pazarının büyüklüğü 101 milyar doları bulmuştur. Statista'ya göre 2016 yılında 149,3 milyar adet mobil uygulama indirilirken, bu sayının 2021 yılında 352,9 miliyari bulması beklenmektedir. Pandemi koşullarının mobil uygulama pazarını hareketlendirdiği bilinmektedir. Kullanıcılar mobil cihazlardaki zamanlarının %89,2'sini mobil uygulamalar üzerinde geçirmektedir (Jay, t.y.). Chan'e (2021) göre, 2021 yılının ilk yarısında dünya genelinde mobil uygulamalar için gerçekleşen harcama 64,9 milyar doları bulmuştur. Bu durum 2020 yılının ilk yarısına oranla %24'lük bir artışı işaret etmektedir. Mobil uygulama

sektörü gelirlerinin yıllar geçtikçe artması beklenmektedir. Yine Statista (2019) sitesinde yer alan bir habere göre 2023 yılında mobil uygulama geliştirme sektörü gelirinin 935.2 milyar dolar olması beklenmektedir.

### 1.3. Uygula: Kullandığımız Mobil Uygulamalar?

Öğrencilerin sıkılıkla kullandığı mobil uygulamaların listesini oluşturunuz. Bu listedeki uygulamaları öğrencilerle birlikte karar vereceğiniz uygulama sınıflarına (Ör: Oyunlar, İletişim, Finans, e-Ticaret) ayırınız. Bu sınıflamada uygulama merkezlerindeki (Ör: App Store) başlıklardan yararlanılabilir.

### 1.4. Gözle: Güncel Mobil Uygulama Geliştirme Eğilimleri

Mobil cihazların ve veri iletim teknolojilerinin gelişimi paralelinde başarılı mobil uygulamaların özellikleri de evrilmektedir. The Scalers (t.y.) sitesi 2021 yılı itibariyle mobil uygulama geliştirme eğilimlerini aşağıdaki şekilde listelenmiştir:

- Mobil Ticaret:** Özellikle COVID-19 salgını nedeniyle ticaret büyük oranda elektronik ortam üzerinden gerçekleştiriliyor hale gelmiştir. Yakın zamanlı raporlara göre 2021 yılında dünya genelinde e-ticaret hacminin 2.8 trilyon Euro olması beklenmektedir. Mobil uygulama geliştiriciler, uygulamalarına tek tuşla sipariş verme, sesli alışveriş ve bütünleşmiş platformlar ve kaynaklar üzerinden satış (omnichannel retail) gibi deneyimleri entegre etmeye çalışmaktadır.
- Platform Bağımsız (Hibrit) Uygulamalar:** Platformlardan bağımsız uygulama üretimi gün geçtikçe popülerliğini artırmaktadır. Bu sayede üretim emekleri ve maliyetleri düşürülerek, programcı zamanı daha etkin kullanılabilmektedir. Bu konu “Hibrit Uygulamalar” başlığı altında daha geniş ele alınmıştır.
- Beacon Teknolojisi:** Beaconlar (ışaretleyiciler), düşük enerjili Bluetooth sinyallerini yayan küçük cihazlardır. Beaconların yaydığı sinyaller cep telefonları gibi Bluetooth iletişimini kullanabilen cihazlar tarafından yakalanabilmektedir. Bu iletişim sayesinde mobil uygulamalar üzerinden yerelleştirilmiş kullanıcı deneyimi sağlanabilmektedir. Örneğin, bir mobil uygulama önünden geçilmekte olan mağaza içindeki işaretçiyi tanıyararak kullanıcıya mağazadaki indirim fırsatları ya da yiyecek menüsü hakkında uyarı verebilmektedir. Bu teknolojinin perakende, kültür-sanat, turizm, taşımacılık gibi sektörlerde önemli uygulama potansiyeli taşıdığı düşünülmektedir.
- Katlanabilir Ekranlar:** Katlanabilir ekranlı yeni mobil cihazlar yüksek talep görmektedir. Katlanabilir ekranların oldukça farklı bir kullanıcı deneyimi sunması beklenmektedir. Örneğin cihaz katlandığında küçük bir ekran alanı sunulurken, cihazın açılması halinde kullanılabilecek ekran alanı iki katına çıkabilecektir. Mobil uygulamaların katlama, açma, yatay/dikey kullanım gibi ekran dönüşümlerine yanıt verebilecek şekilde tasarlanması ve her durumda en iyi kullanıcı deneyimini sunması beklenmektedir. Katlanabilir ekranlar, kullanıcı arayüzü açısından önemli fırsatlar ve riskler barındırabilir.
- Mobil Cüzdanlar:** Mobil cüzdanlar kolay kullanılabilmeleri sayesinde en çok kullanılan ödeme yöntemleri arasında yerini almıştır. Kullanıcılar hesap detaylarını mobil cüzdanlarına bağlayarak fatura ödeyebilir, e-ticaret sitelerinden alışveriş yapabilir ya da aile üyelerine para transferi gerçekleştirebilir. Mobil cihaz geliştiricileri sesli ödeme, NFC (yakın alan haberleşmesi), RFID (RFID açılımı yazılabilir.) etiketleri gibi teknolojileri cüzdanlarla entegre etmeye çalışmaktadır. Uzmanlara göre mobil cüzdanlar uzun vadede en

çok kullanılan ödeme yöntemi haline gelecektir. The Scalers sitesine göre, 2024 yılında 1,7 milyar insan mobil cüzdan kullanıcısı olacaktır.

6. **Giyilebilir Cihazlar:** Akıllı saatler, vücut algılayıcıları, akıllı mücevherler ve hatta akıllı lensler gibi giyilebilir teknolojiler önemli gelişmeler kaydetmektedir. Bu cihazlar akıllı telefonlarla haberleşerek kullanıcıların verilerini (Ör: kalp ritmi, kandaki oksijen miktarı) paylaşabilmektedir. Günümüzde çoğunlukla sağlık sektöründe kullanım bulmuş olsa da sanal ekranlı lensler/gözlükler, sanal klavyeler gibi donanımların yakın zamanda kullanıcılarla buluşması beklenmektedir.
7. **Arttırılmış (Augmented) ve Sanal (Virtual) Gerçeklik:** Arttırılmış gerçeklik, cihazların bulundukları ortamla bütünleşebilmesi; sanal gerçeklik ise ortamdan bağımsız bir sanal kullanıcı deneyimi oluşturmaktır. Özellikle arttırmış gerçeklik teknolojisi alanında önemli gelişmeler kaydedilmektedir. Apple (ARKit) ve Google (ARCore) firmaları cihazlardaki kameralar, mesafe ve hareket algılayıcılarını kullanarak arttırmış gerçeklik deneyimi sağlamayı amaçlayan (ör: odanın hacminin hesaplanması) geliştirme kitleri sağlamaktadır.

#### Not

Google firması tarafından üretilen arttırmış gerçeklik uygulaması ARCore'un yetenekleri <https://www.youtube.com/watch?v=boIfc1PsakA> adresinden incelenebilir.

### 1.5. Uygula: Hangi Eğilimleri Takip Etmeliyiz?

Öğrencilere “Bahsedilen eğilimlerden hangisinin/hangilerinin ticari potansiyelini daha yüksek görünsünüz?” sorusunu yöneltiniz. Aydından her bir eğilime yönelik olası uygulama fikirlerini tartışarak tahtaya yazınız (Ör: Çocukların takip edilmesi için giyilebilir teknolojiler nasıl kullanılabilir?).

### 1.6. Gözle: Mobil Uygulama Geliştirme Yaklaşımı

Mobil uygulamaların geliştirilmesinde çeşitli yaklaşımlara başvurulmaktadır. Her bir yaklaşımın kendine özgü avantajları ve dezavantajlarından söz edilebilir.

**Yerel (Native) Uygulamalar:** Yerel uygulama olarak ifade edilebilecek bu yaklaşımda mobil uygulama hedeflenen platform özgü araçlar ve mekaniklerle tasarılanır. Örneğin Apple'ın iPhone marka telefonlarında koşturulması beklenen bir uygulama, XCode yazılım geliştirme platformunda Objective-C dili ile hazırlanarak Apple Store uygulama mağazasından yayınlanır. Benzer şekilde, Google'ın Android işletim sistemini hedefleyen bir uygulamanın ise Java dilinde Android Studio ile hazırlanması ve Google Play Store üzerinden yayınlanması beklenir. Bu farklılaşma nedeniyle bir platform için geliştirilen uygulama diğerinde çalıştırılamaz. Bu nedenle bu uygulama geliştirme yöntemine yerel (native) geliştirme adı verilir. Bu yaklaşımın iki önemli avantajından söz edilebilir: Bu uygulamalar doğrudan platform geliştiriciler tarafından desteklendiğinden, bu yolla geliştirilen yazılımlarda işletim sistemleri ya da yeni çıkan cihazların en son özellikleri doğrudan kullanılabilir. İkinci önemli avantaj ise uygulamalar doğrudan bu cihazlar ve işletim sistemleri için oluşturulduğundan diğer yaklaşımlara oranla daha hızlı ve etkin çalışmalarıdır.

**Web Uygulamaları:** Bu uygulamalar HTML5 ve çevresindeki teknolojiler (Ör: CSS ve JS) kullanılarak geliştirilir. Özünde bu uygulamalar mobil uygulama gibi görünen, bazı mobil uygulama yetenekleri kazandırılmış web siteleridir. Kullanıcıların bu sitelere giderek uygulamaları indirmeyi kabul etmeleri beklenir. İndirme işleminden sonra bu uygulama kullanıcının ana ekranına bir ikon olarak yerleştirilir. Web teknolojilerindeki gelişmeler paralelinde bu uygulamalar gün geçtikçe daha çok mobil uygulama görünümü kazanmaktadır. Bu uygulama geliştirme yaklaşımının avantajları arasında öncelikle hızlı içerik dağıtımını ve güncelleme özelliğinden bahsedilebilir. Bu uygulamaların içeriklerinin büyük bölümünü uzaktaki sunucu bilgisayarlarda tutulduğundan bu içeriklerin tek bir sunucu üzerinde güncellenmesi tüm müşterilerde güncelleme olanağı getirir. Bunun yanında, bu uygulamaların kurulumları çok hızlı, mobil cihaz üzerindeki kaynak tüketimleri düşüktür. Ayrıca web uygulamaları oldukça zengin bir yazılım geliştirme repertuarına (repertuar yerine birikim, tecrübe, içerik vb. Türkçe sözcük kullanılabilir mi?) sahip HTML5 teknolojilerinden faydalananmaktadır. Pek çok hazır ara yüz bileşeni ve içerik kolayca bu uygulamalara entegre edilebilir. Bununla birlikte, web uygulamaları özünde web siteleri olduğundan, mobil cihazlardaki güvenlik modelleri nedeniyle bu cihazların tüm kapasitesinden faydalananamazlar. Örnek vermek gerekirse bir web uygulaması cihaz üzerinde gelen parmak izi algılayıcısına erişemeyebilir. Ayrıca web uygulamalarının içerikleri mobil cihazlarda değil, bir internet sunucusunda tutulduğundan, bu uygulamalar çoğunlukla internet erişimine bağlıdır.

**Hibrit Uygulamalar:** Güncel mobil cihaz pazarı düşünüldüğünde farklı donanım kapasiteleri ve özelliklerinde binlerce Android ve iOS cihaz bulunmaktadır. Her iki pazarın da ciddi kâr kapasiteleri bulunduğundan, geliştiriciler genellikle tüm cihazlara erişme eğilimindedir. Bu durum mobil uygulama geliştirme sürecine ciddi bir zaman ve maliyet yükü getirmektedir. Öncelikle, Android gibi hedeflenen bir platformun pek çok versiyonu bulunmaktadır. Bunun yanında geliştiriciler her sene yeni bir versiyon ekleme eğilimindedir. Ayrıca uygulamanın çok çeşitli donanım özelliklerinde çalışabilir olması beklenir. Bir teknoloji mağazasına gittiğinizde çok farklı boyut ve en/boy oranlarında mobil cihazlarla karşılaşabilirsiniz. Son olarak yerel uygulama yaklaşımı seçildiğinde üretilen içeriklerin karşı platformda kullanılamayacağı unutulmamalıdır. Bu nedenlerle yerel uygulama geliştirme pahalı bir süreçtir. Hibrit uygulama geliştirme yaklaşımı bu dezavantajlara karşı geliştirilmiştir. Bu yaklaşımda uygulama geliştirme araçları kullanılarak üretilen mobil uygulamalar hedeflenen platformlara özgü uygulama paketleri şeklinde dağıtılmaktadır. Bu ders kapsamında ele alınacak Flutter platformu da Hibrit uygulama geliştirme platformları arasında yer alır. Flutter yanında, React Native, Ionic ve Phonegap gibi pek çok Hibrit uygulama geliştirme platformu bulunmaktadır.

## 1.7. Uygula: Mobil Uygulama Geliştirme Yaklaşımlarını Karşılaştıralım

Bahsedilen üç uygulama yaklaşımında verilen avantaj ve dezavantajları bir tablo halinde özetleyiniz. Bu amaçla aşağıdaki tabloyu örnek alabilirsiniz:

Ölçüt	Yerel Uygulama	Web Uygulaması	Hibrit Uygulama
Geliştirme zamanı			
Maliyet			
...			

## 2. TASARLA ve ÜRET

**proto.io** internet sitesi (Ekran Görüntüsü 1) üzerinde farklı uygulamalar ve içeriklerin arayüzleri ve etkileşimleri (mockup) tasarlanabilmektedir. Bu arayüzler fonksiyonel olmasa da kullanıcı etkileşimi ve mobil uygulamanın görüntüsü hakkında fikir üretmek için tasarlanır.

### Not

Bu sitede alt solda sağlanan Learn bağlantısına tıklandığında editörün kullanımı ve özellikleri ile ilgili bilgi alınabilmektedir.

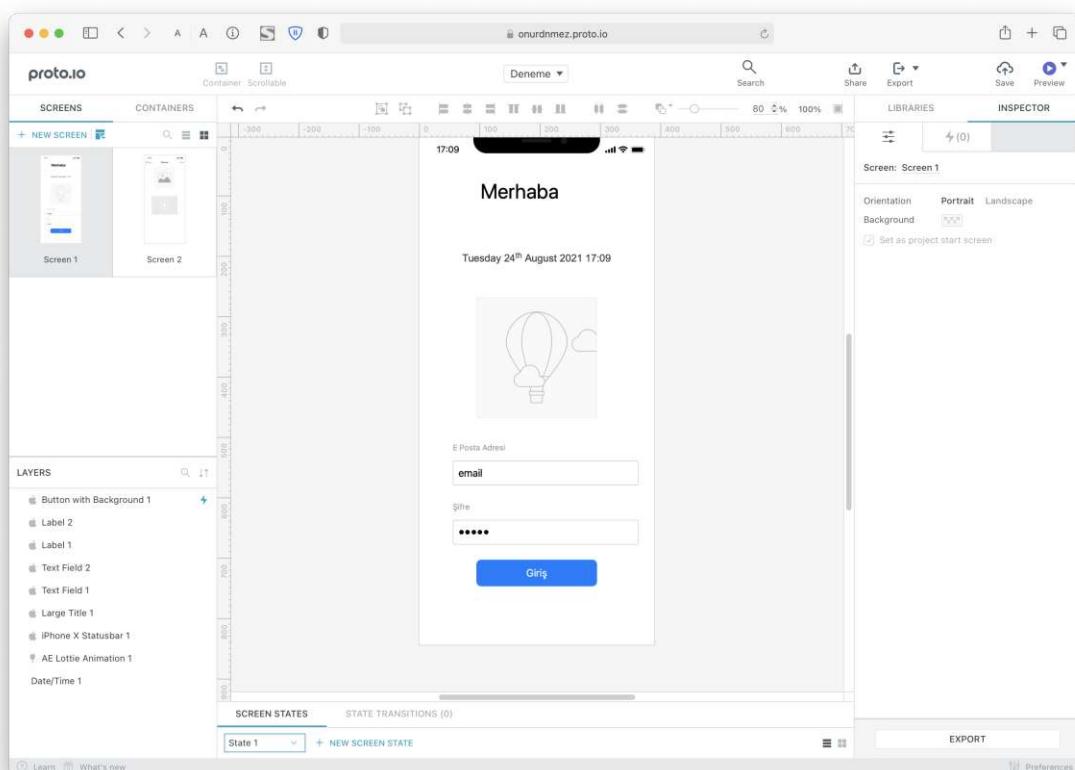
**Tanımlama:** Öğrencilerden öncelikle mobil uygulamanın sayfaları, sayfaların içerikleri ve bu sayfalarda kullanıcının gerçekleştirebileceği eylemleri maddeler hâlinde yazmaları beklenir. Bu aşamada kâğıt ve kaleml kullanarak sayfalar çizilebilir.

Örneğin;

### Giriş sayfası:

- Karşılama mesajı
- Kullanıcı giriş formu
- Şifre hatırlatma bağlantısı

Hatalı giriş yaptığı kullanıcıya bildirilir. En fazla üç hatalı giriş hakkı verilir. Üçüncü hatalı girişten sonra şifre hatırlatma sayfasına yönlendirilir.



Ecran Görüntüsü 1. proto.io internet sitesi arayüzü

Fikir üretme: Öğrencilerden en az üç farklı işlevli sayfa üretmeleri istenir (Ör: Giriş sayfası, şifre hatırlatma sayfası, hesap işlemleri sayfası). Bu sayfalar arasındaki gezinim için mekanizmalar tasarılanması ve bu tasarımın proto.io arayüzünde işlenerek sağ üstteki Preview düğmesini kullanarak prototipin çalıştırılması istenir.

### 3. GÖZLE VE UYGULA

#### 3.1. Gözle: Flutter Nedir?

Flutter, tek bir programlama dilinde yazılan kodlarla, farklı mobil platformlarda (Ör: iOS ve Android) çalışabilen yerel uygulamalar üretebilen bir araçtır. Bu kısa tanımda oldukça önemli ifadeler ve teknolojilerden bahsedilmektedir.

Öncelikle, Flutter projelerinde hedeflenen platformlara özgü yerel uygulamalar üretilir. Bu uygulamaların koşturulması (çalıştırılması) için çeviriçi/ara uygulamalara (Ör: AppInventor Companion, Webview) gerek yoktur. Flutter uygulamaları doğrudan işletim sistemi tarafından çalıştırıldığından diğer hibrit uygulama geliştirme alternatiflerine oranla çok daha hızlıdır. Phonegap gibi alternatiflerde yazılan kodlar aradaki bir katman tarafından (Ör: Webview), üzerinde çalışılan işlemcinin anlayabileceği makine kodlarına çevrilir. Bu ara çevirme nedeniyle uygulamalar performans kaybı yaşıar. Buna karşın Flutter uygulamaları doğrudan üzerinde çalışılan işlemcinin anlayacağı makine kodlarına çevrilmiş olarak dağıtilır. Bu sayede ara çevirme işleminden kaynaklanan performans kaybından etkilenmezler.

Flutter uygulamalarının üretilmesi için Dart programlama dili kullanılır. Dart kodları ile geliştirilen projelerden hedeflenen platformlara özgün yerel uygulamalar üretilerek dağıtilır. Buna karşın, iOS işletim sistemine özgün yerel uygulama üretemek için Objective C ya da Swift dillerinden birini kullanarak XCode ortamında proje üretilmesi gereklidir. Benzer şekilde Android platformuna özgün yerel uygulama üretemek ise Java ya da Kotlin programlama dilleri ile Android Studio ortamında proje üretilmesi gereklidir. Flutter sayesinde tüm bu araçların kullanılmasına, farklı programlama dilleri ve çalışma mekanizmalarının öğrenilmesine gerek kalmaz.

#### Not

Flutter, Yazılım Geliştirme Kiti [YGK] (SDK – Software Development Kit) ve Yazılım Geliştirme Kütüphaneleri olmak üzere iki temel yapı sağlar.

Yazılım Geliştirme Kütüphaneleri Dart dili ile kullanılabilecek kütüphaneleri ve Widget yapılarını sağlar. Dart dilini kullanarak mobil cihaz üzerinde işlem gerçekleştirmek için kütüphaneler kullanılır. Örneğin, mobil cihazdaki kamerasını kullanarak görüntü elde edilmesi, bu görüntünün işlenmesi ve cihazdaki disk alanına resim formatında kaydedilmesi için pek çok kütüphaneye başvurulur. Bunun yanında mobil uygulamanın kullanıcı arayüzü oluşturmak için sayfalar, düğmeler ve menülerini oluşturan geniş bir bileşen (Widget) kütüphanesi bulunur.

Yazılım Geliştirme Kiti boyutunda projelerin oluşturulması ve derlenerek uygulamaya dönüştürülmesi ile ilgili araçlar yer alır. Dart programlama dili ile yazılan kodlar Yazılım Geliştirme Kitindeki araçlar kullanılarak platforma özgün makine diline çevrilir.

Flutter uygulamaları Dart programlama dili ile yazılır. Dart programlama dilinin amacı kullanıcıların gördüğü önyüzleri (front-end) tasarlamaktır. Dart, Google tarafından tasarlanan ve geliştirilmekte olan bir programlama dilidir. Dart nesneye yönelik ve sıkı veri tipi kontrollü (strongly-typed) bir dildir. Bu nedenle değişkenlerin veri tipleri ya da fonksiyonların geri döndürdüğü değerler nedeniyle gerçekleştirilecek uyumsuzluklar derleme sırasında yakalanabilmektedir. Dart ve Flutter, Google firması tarafından yürütülen iki ayrı projedir. Flutter, Dart programlama dili tarafından kullanılabilen nesneler ve kütüphaneler sağlar. Bunun yanında, bu kütüphaneleri kullanan Dart program kodlarını hedeflenen platforma özgün uygulamaya çevirebilecek Yazılım Geliştirme Kiti de Flutter tarafından sağlanır.

### 3.2. Gözle: Flutter’ın Hedeflediği Platformlar

2021 Yılı Haziran ayı itibarıyle Flutter aşağıdaki platformlara özgün uygulama üretebilmektedir:

1. **Android:** Google firması tasafından üretilen Android dünya genelinde en çok kullanılan mobil işletim sistemidir. Güncel verilere göre Android'in pazar payı %72 civarındadır.
2. **iOS:** Apple firmasının iPhone model telefonlarında kullanılan işletim sistemidir. Güncel verilere göre iOS'un pazar payı %27 civarındadır.
3. **MacOS:** Apple firmasının Mac modeli dizüstü ve masaüstü bilgisayarlarında kullanılan işletim sistemidir. Bu uygulamalar firmanın uygulama mağazası üzerinden dağıtılabılır.
4. **Linux:** Dünya genelinde kullanılabilen özgür bir işletim sistemidir. Pek çok Linux türevi bulunmaktadır. Türkiye'de geliştirilen Pardus da bir Linux türevidir. Flutter, bu türevler üzerinde çalışabilecek snap paketleri oluşturabilmektedir. Bu paketler masaüstü ve dizüstü bilgisayarlarda kullanılabilir.
5. **WebApp:** Temelde bir internet sitesi olarak ele alınabilir. Chrome, Safari, Edge ve Firefox gibi internet tarayıcılarının masaüstü ve mobil sürümleri üzerinde çalıştırılabilen uygulamalardır.

### 3.3. Gözle ve Uygula: İlk Flutter Uygulamanız

Flutter uygulamalarını üretmek oldukça basittir. Kurulum işlemlerini bitirdikten sonra uygulamanın üretimeceği konuma geçilerek komut satırı yoluyla uygulama üretilebilir. Özellikle geliştirme ve öğrenme sürecinde birçok uygulama üretimeceğinden bu amaçla bir klasör oluşturulması önerilir. Komut satırı yoluyla bu klasöre giderek ilk uygulamamızı üretelim. Kolaylık olması açısından uygulamamızı şimdilik sıra ile isim verelim. İlk uygulamamızın adı app1 olsun. Kod 1'deki kodu komut satırına yazarak ilk uygulamamızı üretelim.

```
flutter create app1
```

Kod 1. Yeni bir Flutter uygulaması üreten kod

Flutter projelerini üretmek için **flutter create** komutu kullanılır. Bu komutun arkasından uygulamanın ismi verilir. Bu komutla birlikte ismi **app1** olan bir Flutter uygulaması üretilmiş

olur. Bulunduğumuz klasörde **app1** adında bir klasör üretilecektir. Dosya gezgini ile bu projeye bakıldığında içinde oldukça fazla klasör ve dosya üretildiği görülecektir. Aşağıda bu klasör ve dosyalarla ilgili kısa bir bilgilendirme yapılacaktır. Bundan önce üretilen projemizi inceleyelim.

### 3.4. Gözle ve Uygula: İlk uygulamayı WebApp Olarak Koşturmak

Az önce ürettiğimiz **app1** uygulamasını **WebApp** olarak koşturmak için Google firmasının Chrome tarayıcısının sistemimizde yüklü olması gerekmektedir. Bu tarayıcı sisteminizde henüz yüklü değilse [https://www.google.com/intl/tr\\_tr/chrome/](https://www.google.com/intl/tr_tr/chrome/) adresine giderek ücretsiz olarak dağıtılan Türkçe sürümünü indirebilirsiniz.

Şimdi de ürettiğimiz uygulamayı koşturmak için gerekli kodları çalıştırıralım. Flutter uygulamaları iki yolla çalıştırılabilir. Öncelikle bu amaçla komut satırı kullanılabilir. Bunun yanında, uygulamalar düzenleme aracı (Ör: VSCode) üzerindeki kontroller yoluyla da çalıştırılabilir. Öncelikle komut satırı yolunu deneyelim. Bu amaçla uygulamanın kök klasörüne geçiş yapılması gerekmektedir (Kod 2):

```
cd app1
```

Kod 2. app1 uygulamasının kök klasörüne geçmek

Uygulamanın kök klasörü içindeyken **flutter run** komutu çalıştırılarak uygulama Google Chrome tarayıcısı içinde incelenebilir.

```
flutter run
```

Kod 3. Komut satırı üzerinden bir uygulamayı çalıştmak

Bu kod çalıştırıldığında komut satırında bilgilendirme mesajları gösterilecektir (Ekran Görüntüsü 2). Kullanılan bilgisayarın hızına göre bu süre uzayabilir.

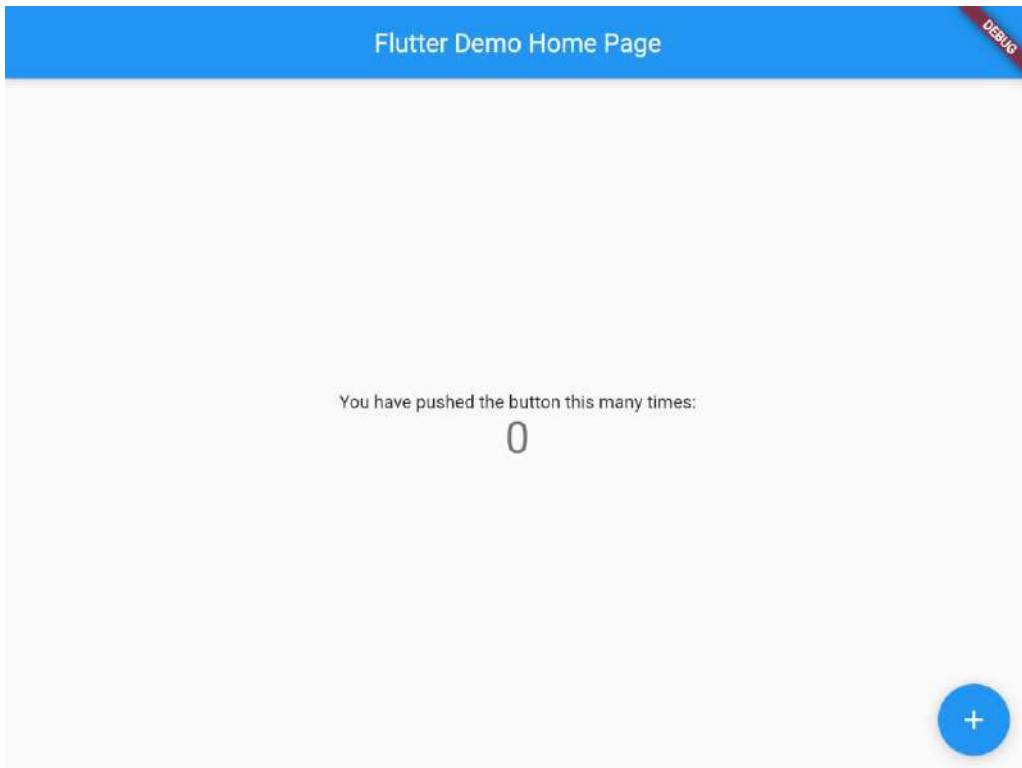
```
Launching lib/main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...                                14.4s
Debug service listening on ws://127.0.0.1:60287/y9FX9L9VNyE=/ws

Running with sound null safety
For more information see https://dart.dev/null-safety/unsound-null-safety

🔥 To hot restart changes while running, press "r" or "R".
For a more detailed help message, press "h". To quit, press "q".
```

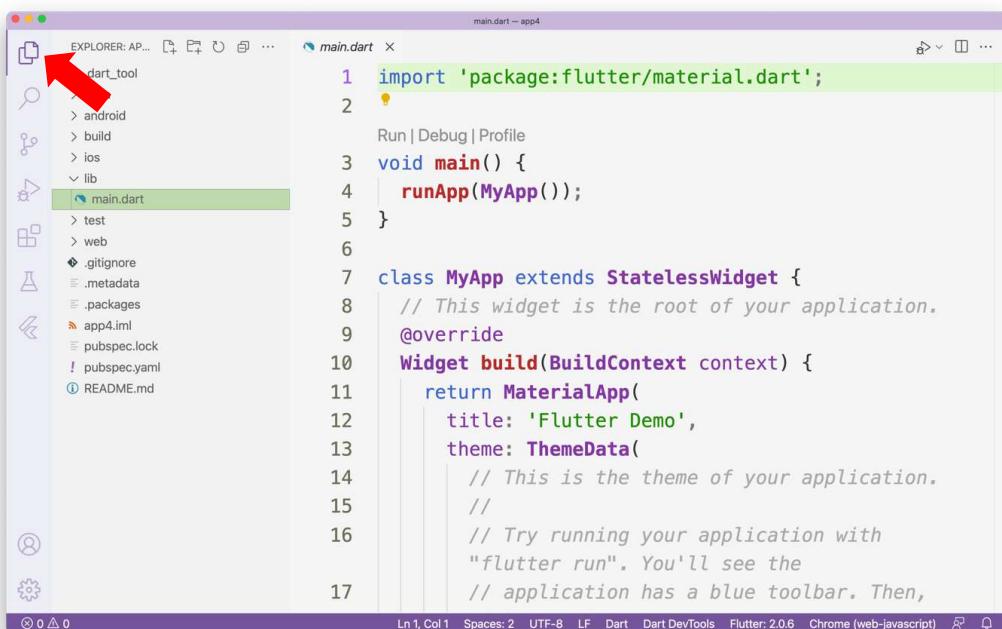
Ecran Görüntüsü 2. Uygulama çalıştırıldığında komut satırında görülen bilgilendirme ve yardım mesajları

Google Chrome tarayıcısının ekran görüntüsü ise Ekran Görüntüsü 3'te gösterildiği şekilde olacaktır. Bu ekranda sağ alta bulunan mavi düğmeye her tıklandığında ortada yer alan sıfır sayısı artacaktır. Uygulama mavi düğmeye kaç kez tıklandığını saymaktadır.



Ekran Görüntüsü 3. Temel Flutter uygulaması ekran görüntüsü

Üretilen uygulamanın klasörüne bakılırsa, birçok dosya ve klasör üretildiği görülebilir. Bir sonraki başlıkta bu dosya ve klasörlerle ilgili bilgi verilecektir. Bu dosyalardan en önemlisi lib klasörü altındaki **main.dart** dosyasıdır. Dosyaların düzenlenmesi için VSCode düzenleyicisinin kullanılması önerilmektedir. Ücretsiz dağıtılan bu düzenleyici <https://code.visualstudio.com/download> adresinden edinilebilir.

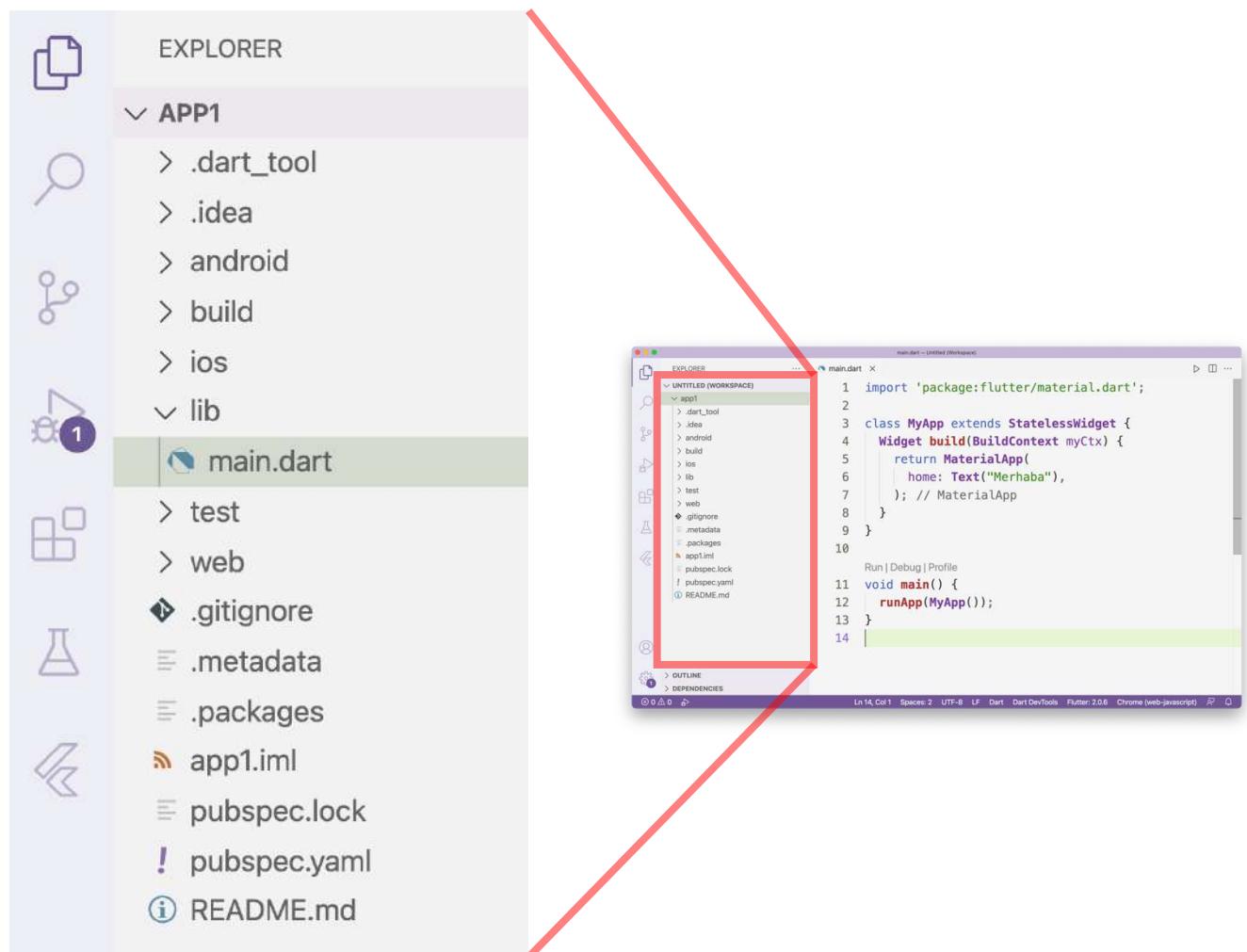


Ekran Görüntüsü 4. Flutter uygulaması ve main.dart dosyasının VSCode düzenleyicisindeki görüntüsü

Kurulum işlemi sonrasında VSCode düzenleyicisi açılarak işaretlenen Explorer düğmesine tıklanır (Ekran Görüntüsü 4). Bu düğme projelerin dosyalarının gösterildiği paneli açıp kapatır. **app1** uygulamasının klasörü bu panelin üzerine sürüklendiğinde proje VSCode düzenleyicisi içinde açılmış olur. **lib** klasörü altındaki **main.dart** dosyasına tıklanarak bu dosyanın içi görüntülenebilir. İçeriği oldukça karışık görünümekle birlikte, incelediğinde hangi kodların ne işe yaradığını ilişkin bir fikir edinilebilir.

### 3.5. Gözle: Flutter Uygulamalarındaki Önemli Dosyalar

Ekran Görüntüsü 5'te sağ tarafta **app1** isimli Flutter projesi VSCode düzenleyicisi üzerinde açılmıştır. Solda ise bu projenin dosya ve klasörlerinin gösterildiği Explorer paneli görülmektedir. Bir Flutter projesi üretildiğinde YGK bu dosya ve klasörleri otomatik olarak üretir. Her bir klasör ve dosyanın kendine özgü görevleri bulunmaktadır. Bu noktada, kullanılan düzenleyici, işletim sistemi ve Flutter YGK versiyonuna göre üretilen dosya ve klasörlerin farklılık gösterileceğine dikkat edilmelidir. Dosya ve klasörlerin isimlerinin farklılık gösterebileceği gibi, bu resimlerde görülen bazı klasör ve dosyalar üretilmeyebilir, ya da bunlar dışında farklı isimlerde dosyalar ve klasörler de sistem tarafından üretilmiş olabilir. Yine de önemli dosyalar ve klasörlerin tüm sistemlerde benzer olması beklenmektedir. Bu bölümde klasör ve dosyaların görevlerine kısaca değinilecektir.



Ecran Görüntüsü 5. Flutter uygulamalarındaki önemli dosyalar

**android:** Anroid klasörü bütün bir Android projesi tutar. Flutter yazılım geliştirme kiti, oluşturulan proje ile bu proje içindeki Android projesini birleştirerek bir Android uygulaması üretir. Bu klasör içinde herhangi bir değişiklik yapılmasına gerek yoktur. Oldukça önemli bir klasör olmasına rağmen, bu klasör pasif bir klasör gibi görülmelidir. Yalnızca projenin çıkışının alınacağı durumlarda üretilen APK dosyasına bu klasör içinden ulaşılacaktır.

**ios:** Android klasörünün iOS işletim sistemine özgün karşılığı olarak düşünülebilir. Bu klasör içinde de bütün bir iOS projesi bulunur. Proje derlenirken yazılan kodlar bu proje ile birleştirilerek sonuç iOS projesi üretilicektir. Bu klasör de pasif bir klasör olarak görülebilir. Apple şirketi kendi araçları dışında iOS uygulamaları üretilmesine izin vermediğinden Windows çalışma ortamında ios klasörü görülemez.

**build:** build klasörü Flutter YGK tarafından derlenmiş kodların tutulduğu klasördür. Bu klasör YGK tarafından düzenlenir. Bu klasöre müdahale edilmemelidir.

**lib:** lib (Library) klasörü Flutter uygulamalarında en çok kullanılan klasördür. Dart kodları bu klasördeki dosyalara eklenir. Flutter projelerinin geliştirilmesi sürecinde en önemli klasör lib klasöridür.

**test:** Test klasörü, Flutter projelerini test etmek için üretilen otomatikleştirilmiş rutinleri tutar. Daha çok İleri düzey Flutter programcılarının kullanabileceği bir klasördür.

**.gitignore:** Git kaynak kodu yönetim sistemi tarafından kullanılan bir dosyadır. Git, yazılan kodların günlük kopyalarını (snapshot) tutabilen bir versiyonlama sistemidir. Bu sayede kodlarda geriye dönüş gibi olanaklar sağlanır. Flutter uygulaması geliştirme sürecinde bu uygulamanın kullanılması zorunlu olmamakla birlikte, programcıların işlerini kolaylaştırın bir uygulamadır.

**.metadata:** Flutter uygulamasının geliştirme süreciyle ilgili Flutter tarafından takip edilen üst verileri saklamakla görevli bir dosyadır. Programcıların müdahale etmesine gerek yoktur.

**.packages:** Bu dosya Flutter projesi tarafından kullanılan kütüphanelerin takip edildiği dosyadır. Flutter YGK tarafından oluşturulur ve düzenlenir. Programcıların müdahale etmesine gerek yoktur.

**app1.iml:** IML dosyası projenin ismi ile başlar. Bu senaryodaki projenin adı **app1** olduğundan bu dosyanını adı **app1.iml** olarak belirlenmiştir. Flutter Yazılım Geliştirme Kiti tarafından projenin ayarları ve gereksinimlerinin takip edilmesi amacıyla üretilir ve düzenlenir. Programcıların müdahale etmesine gerek yoktur.

**pubspec.lock:** Bu dosya aşağıda anlatılacak **pubspec.yaml** dosyasının içeriğine göre otomatik olarak üretilir. Programcıların müdahale etmesine gerek yoktur.

**pubspec.yaml:** Flutter projesinin temel ayarları ve projeye eklenecek üçüncü parti kaynakların belirtildiği belgedir. Bu belge **yaml** dili kullanılarak yapılandırılmıştır. Projede kullanılacak fontlar, ikon paketleri, üçüncü parti kütüphaneler ya da görseller bu belgeye işlenir. Bunun yanında projenin hangi Dart versiyonunda ve hangi paketlerle geliştirildiğine ilişkin ayarlar bu belgede tutulur.

**README.md:** Projenin diğer geliştiricilerle paylaşılması halinde, proje hakkında bilgi sağlamak amacıyla README.md dosyasının içeriği değiştirilebilir.

### 3.6. Gözle: main.dart Dosyasının Anatomisi

Flutter tarafından üretilen temel uygulamada oldukça fazla kod ve açıklama satırı olduğundan, **main.dart** dosyasının temel anatomisinin anlaşılması için Kod 4 kullanılacaktır. Bu kodun işlevi oldukça basittir: Ekrana “Merhaba!” yazmak. Yine de **main.dart** dosyasının işleyişi ile ilgili temel mekanizmaların anlaşılması için faydalı olacaktır.

Bu kodu satır satır inceleyelim. Öncelikle 1. satırda bir **import** işlevi çağrılmaktadır. Bu **import** çağrıları ile Flutter tarafından sağlanan pek çok özellik kullanılabilir hale gelecektir. Örneğin bu uygulamada kullanacağımız **StatelessWidget**, **MaterialApp**, **Text** gibi sınıfların tümü **material.dart** kütüphanesi üzerinden çekilecektir. Bu sınıflar uygulamamız içinde değil, geliştirme yapılan bilgisayardaki Flutter dosyalarında saklanmaktadır. Bu sınıflara ilişkin kodlar derleme esnasında uygulamaya enjekte edilecektir. Bu satır silindiğinde yukarıda bahsettiğimiz tüm sınıfların altına kırmızı çizgi eklendiği, yani bu sınıflarla ilgili sorun olduğu gözlenecektir. Bunun nedeni az önce bahsedildiği gibi bu sınıflara ilişkin tanımlama ve kodların **material.dart** kütüphanesi üzerinden projeye çekiliyor oluşudur. **import** işlemi **pubspec** dosyasında **dependencies** bölümünde **flutter** YGK'nın içerisinde sayesinde yapılmıştır. Bu tanımlama ile Flutter kütüphanesi ve uygulama arasında bağlantı sağlanmaktadır.

```
1 import 'package:flutter/material.dart';
2
3 void main() {
4     runApp(MyApp());
5 }
6
7 class MyApp extends StatelessWidget {
8     Widget build(BuildContext context) {
9         return MaterialApp(home: Text("Merhaba!"));
10    }
11 }
```

Kod 4. Merhaba! uygulaması

3. satırda **main()** fonksiyonu tanımlanmaktadır. Dart, nesne yönelik bir dil olduğundan kodlarının başlangıcının işaretlendiği bir giriş noktasına ihtiyaç vardır. Dart sınıflarında bu giriş noktası **main()** fonksiyonudur. Daha önce bahsedildiği gibi Dart sıkı veri tipi kontrolü kullanan bir dildir. Bunun nedenle her fonksiyonun bir veri tipi ile tanımlanması gerekmektedir. İki sayının ortalamasını alıp geri çeviren bir fonksiyonun bu değeri saklayabilecek (ör: **double**) bir tiple tanımlaması gereklidir. Değer döndürmeyecek fonksiyonlar ise **void** anahtar kelimesi ile açıkça değer döndürmez şeklinde tanımlanır. Her fonksiyon tanımlamasının arkasında fonksiyona parametre göndermek için parantezler konur. Bu parantezlerin eklenmesi parametre gönderilmeyecek olsa dahi zorunludur. Fonksiyonların başlangıç ve bitişleri küme parantezleri ile işaretlenir. 3-5. satırlar arasındaki **main()** fonksiyonu **runApp** fonksiyonunu çağrılmaktadır. Bu fonksiyona 7-11. satırlar arasında tanımlanan **MyApp** sınıfını parametre olarak

gondermektedir. Bu sayede Dart, **MyApp** sınıfı içinde tanımlı kodları çalıştırması gerektiğini anlamaktadır.

Flutter uygulamalarında ekranda görünen her şey bileşen (**Widget**) olarak tasarlanmıştır. Uygulamanın kendisi de dahil olmak üzere her nesne bir bileşendir. Bileşenlerin yerleşimi ve hiyerarşisi kodlar ile belirlenir. Örneğin **Hata! Başvuru kaynağı bulunamadı.**'de yer alan temel uygulamada üstte yer alan başlık barı, orta bölümdeki iki farklı yazı alanı ve sağ alttaki düğmenin her biri bileşendir. Bunların yanında listeler, resimler, form elementleri ve yayılım düzenleyiciler gibi daha birçok bileşen türü vardır. Listeler gibi bileşenler liste satırlarını tutabilen kapsayıcı bileşenlerdir. Tüm bunlar kökü Flutter uygulamasının kendisi olan bir bileşen ağacı içinde tasarlanaarak kullanıcı ara yüzü oluşturulur.

7. satırda bileşen ağacının kökü olan, uygulamanın kendisini tanımlayan **MyApp** isimli sınıfın tanımlanmasına başlanmaktadır. Bu amaçla **class** anahtar sözcüğü kullanılır. Ardından sınıfın adı olan **MyApp** verilmiştir.

#### Not

Not: Sınıflar ve değişkenlerin tanımlanmasında bazı standartlar bulunur. Sınıflar Pascal sözdizimi stilinde tanımlanır. Bu stilde her bir sözcüğün baş harbi büyük olacak şekilde sözcükler birleştirilir (Ör: **BirinciFlutterUygulamam**). Sınıflar içindeki fonksiyonlar ise Camel sentaksı ile isimlendirilir. Bu stilde ilk sözcüğün ilk harfi küçük, diğer sözcüklerin ilk harfleri büyütür (Ör: **birinciFlutterFonksiyonum**). Dart programlama dilinin kuralları gereği bu isimlerde boşluk bulunmadığından isimlerin anlaşılması için böyle bir sistematik geliştirilmiştir. Tireler ya da alt çizgiler kullanılmaz. Bu nedenle uygulamanın adı **myApp** değil, **MyApp** şeklinde verilmiştir.

Flutter sistemindeki kalıtım (**inheritance**) mekanizması sayesinde ekranda içerik göstermek için ekstra kod yazmamıza gerek kalmayacaktır. **MyApp** sınıfı yalnızca ekranda nasıl bir içerik göstermek istediğimizi belirtecektir. Ekrandaki pikseller bizim adımıza Flutter tarafından kontrol edilecektir. Kalıtım mekanizmasını çalıştmak için **extends** anahtar kelimesi kullanılır. **extends** anahtar kelimesi ile **MyApp** isimli sınıfımız (yani Flutter uygulamamız) Flutter tarafından sağlanan **StatelessWidget** sınıfı üzerine kurgulanmaktadır. **StatelessWidget** sınıfı içindeki binlerce satırlık kod ile üretilmiş mekanizmalar artık uygulamamızda kullanılabilir hale gelmiştir. **MyApp** bu kodlara yeni kodlar ekleyerek uygulama ara yüzünü örecektr. **StatelessWidget** sınıfı sabit içeriklerin gösterilmesi için kullanılabilecek bir sınıfır.

#### Not

İleride sistem saatini göstermek gibi sistemden ya da kodlarımız dışından gelecek verilerle içeriği değişecek bileşenler için  **StatefulWidget** sınıfı kullanılacaktır.

Sınıf tanımlaması yapıldığı anda **MyApp** ifadesinin altı çizilerek **build** metodunun çalıştırılması gereği bildirilir. Bu hata **StatelessWidget** sınıfının tanımlamaları gereği **build** metodunun çalıştırılması zorunluluğundan kaynaklanmaktadır. Kodlarda **build** sözcüğü tek başına kullanılsa da aslında **StatelessWidget** sınıfı içinde tanımlı bir metottur.

## Not

Fonksiyon ve metot ifadeleri bu kitap içinde sıkılıkla kullanılacaktır. Aralarındaki ayırm oldukça basittir. Metotlar özünde birer fonksiyondur. Fakat bir sınıf içinde tanımlanan fonksiyonlar bu sınıfın metotları olarak anılır. Herhangi bir sınıf içinde yer almayan yeniden çalıştırılabilir kod blokları ise fonksiyon olarak isimlendirilir.

**build** metodu parantezleri içinde **BuildContext** isimli parametrenin verilmesi zorunludur. **BuildContext** isminden sonra istenilen bir değişken ismi verilebilir. Bu uygulamada değişken ismi olarak **context** seçilmiştir. **build** metodunun çalıştırıldığında geriye bir nesne gönderir. Bu nesnenin bir **Widget** olduğunu belirtmek için **build** fonksiyonunun önüne **Widget** anahtar sözcüğü eklenmiştir. Bu sayede build fonksiyonunun döndürdüğü nesnenin **Widget** sınıfında bir nesne olduğu tanımlanmış olur.

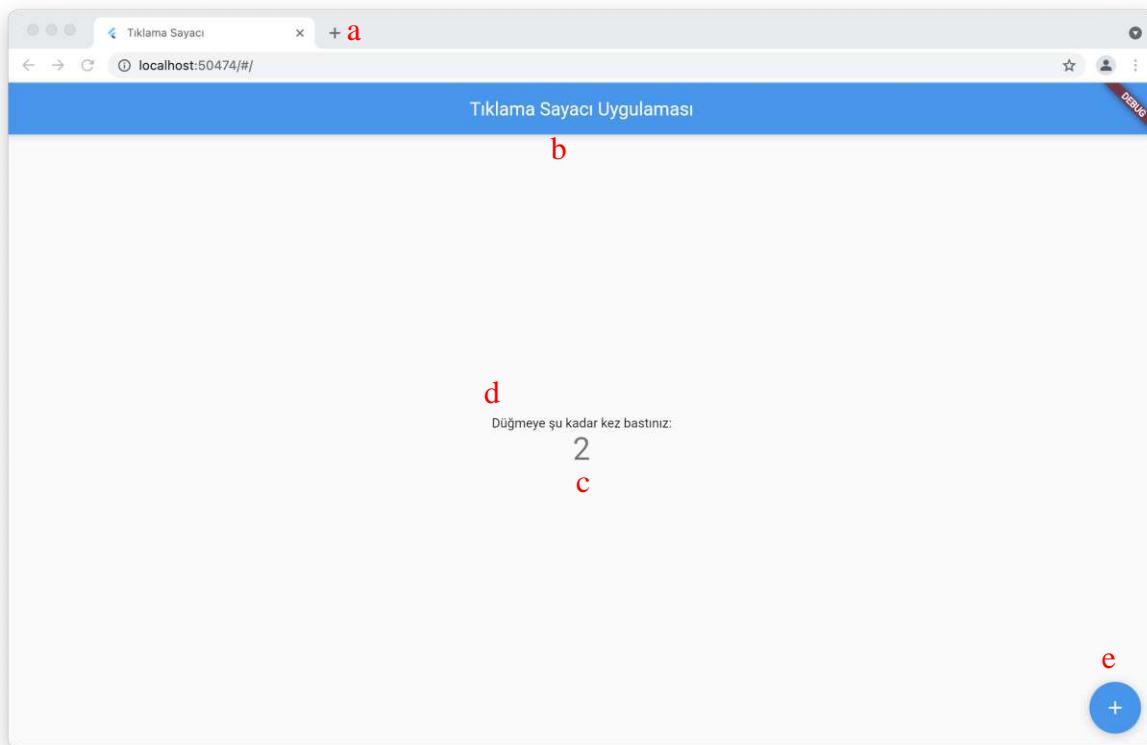
**build** metodunun ürettiği içeriği geri döndürebilmesi için **return** anahtar sözcüğü kullanılır. **MaterialApp** sınıfı üzerinden üretilen bileşen **return** sözcüğü ile geri döndürülmektedir. **MaterialApp** sınıfının tanımlandığı parametre listesinde **home:** ifadesi yer alır. **home:** ifadesi ile bu uygulama açıldığında gösterilecek içerikler tanımlanmaktadır. Yine ön tanımlı bir bileşen olan **Text** kullanılarak ekrana yazı göndermek için içerik üretilir. **Text** sınıfına “Merhaba!” değeri gönderilerek bileşen üretilmiş olur.

Sonuç olarak, kodumuzun çalışma mekaniklerini inceleyeceğiz:

Uygulama çalıştığında **main** fonksiyonu otomatik olarak çalıştırılır. **main** fonksiyonu **runApp** fonksiyonu ile ürettiğimiz **MyApp** sınıfını çalıştırır. **MyApp** sınıfı **StatelessWidget** sınıfını genişlettiğinden bu sınıfın zorunlu tanımlaması gereki geriye bir **Widget** döndüren **build** metodu çalıştırılır. **build** metodu 9. Satırda oluşturulan **MaterialApp** bileşenini geri döndürdüğünden bu bileşen ekrana gönderilmiş olur. **MaterialApp** sınıfının oluşturulmasında çalıştırılan kodlara **Text** bileşeni gönderildiğinden oluşturduğumuz **Merhaba!** yazısı ekrana basılacaktır.

### 3.7. Uygula: main.dart Dosyasını Düzenleyelim

1. Yeni bir Flutter uygulaması üretecek, temel uygulamanın arayüzüni aşağıdaki ölçütlerle göre düzenleyiniz (Ekran Görüntüsü 6).
  - a. Uygulama başlığını “Tıklama Sayacı” olarak düzenleyiniz.
  - b. Uygulama başlık çubuğundaki ifadeyi “Tıklama Sayacı Uygulaması” olarak düzenleyiniz.
  - c. Uygulamadaki sayacı 1’den başlatınız.
  - d. Uygulamada sayaç üzerindeki ifadeyi “Düymeye şu kadar kez bastınız:” şeklinde düzenleyiniz.
  - e. Uygulamadaki düğme üzerine gelindiğinde görünen yardım metnini “Arttır” şeklinde düzenleyiniz.



Ecran Görüntüsü 6. Değerlendirme ödevi arayüzü

## 4. DEĞERLENDİR

Üret adımda oluşturulan uygulama dışa aktar yapılarak kaydedilir. Uygulama dosyası eğitmen ve diğer öğrencilerle paylaşılır. Bütün öğrenciler kendi uygulamasının yanında diğer uygulamaları da inceler. Öğrencinin özdeğerlendirme yapabilmesi için **Öz Değerlendirme Formu** (<https://forms.gle/kts381jqHWcCah2j6>) uygulanır.

Proje Hazırlama

Ayrıca dersin sonunda sunum yapmak üzere öğrencilerin bir proje fikri belirleyip her hafta edindikleri bilgileri de kullanabilecekleri bir proje geliştirmeleri istenir. Bu projenin özgün, belirlenmiş amacı gerçekleştirebilen ve test edilebilir düzeyde tamamlanmış olması beklenmektedir. Öğrencilerinizi bu konuda bilgilendirerek, bir sonraki haftaya kadar proje fikri oluşturmalarını isteyiniz.

## 5. İLAVE ETKİNLİK

### 5.1. Tasarımınızı Geliştirin

Tasarlama etkinliğinde üretilen prototip siteye yeni bir sayfa ekleyiniz. Örneğin mobil bir alışveriş sitesi tasarladığınız bu uygulamaya ürün tanıtımları ya da satın alma ile ilgili bir sayfa ekleyebilirisiniz.

## 6. KAYNAKÇA

Chan, S. (2021). *Global App Spending Approached \$65 Billion in the First Half of 2021, Up More Than 24% Year-Over-Year.* 14.07.2021 tarihinde <https://sensortower.com/blog/app-revenue-and-downloads-1h-2021> adresinden erişilmiştir.

Jay, A. (t.y.). *16 Mobile App Trends for 2021/2022 and Beyond: Top Forecasts According to Experts.* 14.07.2021 tarihinde <https://financesonline.com/mobile-app-trends/> adresinden erişilmiştir.

Statista (2019). *Worldwide mobile app revenues in 2014 to 2023.* 11.08.2021 tarihinde <https://www.statista.com/statistics/269025/worldwide-mobile-app-revenue-forecast/> adresinden erişilmiştir.

The Scalers (t.y.). *7 mobile app development trends for 2021 and beyond.* 11.08.2021 tarihinde <https://thescalers.com/7-mobile-app-development-trends-for-2021-and-beyond/> adresinden erişilmiştir.

O'Dea, S. (2021). *Smartphone sales worldwide 2007-2021.* 14.07.2021 tarihinde <https://www.statista.com/statistics/263437/global-smartphone-sales-to-end-users-since-2007/> adresinden erişilmiştir.

## 2. Hafta: Dart Programlama Dili

### Ön Bilgi:

- Temel programlama bilgisi
- Temel internet kullanım bilgisi

### Haftanın Kazanımları:

- Dart programlama dilinin temel işlevlerini açıklar
- Dart programlama diline özgü çalışma mekanizmalarını açıklar
- Dart programlama dilinde yorum satırı ekleyebilir
- Dart programlama dilinde amaçlarına uygun veri yapıları oluşturabilir
- Dart programlama dilinde oluşturulan veri yapılarını kullanabilir
- Dart programlama dilinde şart ifadeleri oluşturabilir
- Dart programlama dilinde döngü yapıları oluşturabilir
- Dart programlama dilinde fonksiyon üretebilir
- Dart programlama dilinde üretilmiş fonksiyonları kullanabilir
- Dart programlama dilinde sınıf üretebilir
- Dart programlama dilinde üretilmiş sınıfları kullanabilir

### Haftanın Amacı:

Bu haftanın amacı, Flutter uygulamalarının üretilmesinde kullanılan Dart programlama dilinin temellerinin kavranmasıdır. Öncelikle Dart programlama diline özgü çalışma mekanikleri (Ör: null-safety) sunulacaktır. Ardından sırasıyla yorumlar, değişkenler, şart ifadeleri, döngü yapıları, fonksiyonlar ve nesneye yönelik programlama konularına değinilecektir.

### Kullanılacak Malzemeler:

Bu derste anlatılacak uygulamalar için daha önceki derste kullanılan kurulum yeterli olabilir. Yazılan kodların denenmesi ve çıktıların gözlenmesi için üretilen projeler Chrome tarayıcısı üzerinde çalıştırılabilir. Fakat bu işlem süreçleri uzatabileceğinden, aynı etkilerin gözlenmesi için <https://dartpad.dev> internet sitesinin kullanılması önerilmektedir. Bu sitenin kullanılması üyelik gerektirmemektedir.

#### Bilgi

Bu bölümde yazılacak kodların sonuçlarını daha hızlı inceleyebilmek için <https://dartpad.dev> internet sitesinin kullanılması önerilmektedir

### Ders Öncesi Gereksinimler

<https://dartpad.dev> sistemi kullanılacaksa çalışan bir internet bağlantısı gerekecektir.

## **Haftanın İşlenisi:**

**Gözle:** Değişkenler, yorum satırları, şart ifadeleri, döngüler gibi temel programlama yapılarının Dart dilinde nasıl kullanıldığını gözle

**Uygula:** Programlama yapılarının uygulandığı örnekleri uygula ve verilen alıştırmaları çözümle

**Tasarla:** Nesneye yönelik programlama mekaniklerine göre bir hayvanat bahçesi simülasyonu için hayvan sınıflarını tasarıla

**Üret:** En az üç hayvan için gerekli sınıfları üret

**Değerlendir:** Haftanın içeriği ile ilgili yansıtma etkinliği

## **1. GÖZLE VE UYGULA**

### **1.1. Gözle: Temel Kavramlar**

Dart çeşitli platformlarda çalışabilecek uygulamalar üretmek için geliştirilmiş bir programlama dilidir (Google, t.y.). Dart dilinin amacı platformlara (Ör: iOS, Android) özgü istemci uygulamalar üretmektir. Flutter uygulamaları Dart programlama dili ile kodlanır. Aşağıda Dart dilinin bazı temel özelliklerini sıralanmıştır. Bu özelliklerin bazıları diğer programlama dilleriyle benzerlik gösterdiğinde kolaylıkla öğrenilebilir. Bunun yanında, Dart'a özgü bazı özelliklerin temel amacı, kullanıcıların karşılaşabileceği hataları geliştirme zamanında çözümlemektir. Bu sayede kullanıcılar değer atanmış değişkenlerin kullanıldığı kodlar gibi oldukça basit nedenlerle yaşanabilecek uygulama çakılmalarından etkilenmezler. Bu çakılmalarda herhangi bir hata mesajı verilmemişinden (**silent error**), kullanıcılar bir çözüm geliştirememekte ve geliştiricilere hata bildirimi yapamamaktadır. Bu tür durumların sık yaşanması kullanıcıların uygulamaya yönelik güvenini ve kabulünü olumsuz etkilediği bilinmektedir. Bu çerçevede, Dart dilinde alınan önlemlerin oldukça faydalı olduğu düşünülmektedir.

- Dart nesne yönelimli bir dildir. Değişkenlere atanın her ifade (Ör: sayılar, fonksiyonlar) bir sınıfından türetilen nesnelerdir.
- Dart, Java gibi dillerde kullanılan **public**, **protected** erişim belirleyicilerini kullanmaz. Bir ifadenin kütüphaneye özgü kalması ve dışarıdan erişilmemesi isteniyorsa önüne alt çizgi eklenir (Ör: **\_ozelDegiskenim**).
- Dart, sıkı veri tipi kontrolü (strongly typed) kullanır. Bu sayede değişkenler tek türde veri (Ör: **int**, **String**) tutabilir. Bir değişkene atanamayacak bir değerin atanmasını isteyen kodlar derleme sırasında yakalanır ve hataların ayıklanması istenir. Örneğin sayısal değerler alabilen bir değişkene **Boolean** tipinde bir değer atamaya çalışıldığında, Dart derleyecisi bir hata mesajı vererek uygulamayı derlemez. PHP gibi dillerde bu durumda tip dönüşümü uygulanırken, JavaScript gibi dillerde değer doğrudan değişkene aktarılabilir. Fakat bu tür uygulamalar kodlarda beklenmeyen sorunlara yol açabilmektedir. Örneğin **fiyat** değişkeninde **100** değerini, **vergi** değişkeninde **18** değerini sakladığımızı düşünelim. Sıkı veri tipi kontrolü kullanmayan bir dilde, kodların yürütülmesi sırasında belli bir noktada **vergi** değişkenine **String** türünde “**kdv**” değeri atanabilir. Bu durumda bu iki değişkeni toplayacak **sayi1+sayi2** kodunun sonucu **String** türünde “**100kdv**” olacaktır. Sonuç olarak, bu değer bir ödeme sistemine

gönderilirse sorun yaratacaktır. Bu tür beklenmeyen sonuçlardan korunmak için Dart sıkı veri tipi kontrolü uygular.

- Dart dilinde değişkenlerin tanımlandığı satırlarda belirli bir veri tipinin belirtilmesi zorunluluğu yoktur. Dart'ın tip çıkarımı mekanizması gerçekleştirilen atamayı temel alarak değişkenin hangi tipte olması gereğine karar verebilir. Örneğin **sayı** değişkeni **int** tipinde tanımlamak zorunda değildir. Bunun yerine **var** anahtar sözcüğü ile tanımlanır. **sayı** değişkenine **10** değeri atandığında Dart tip çıkarımı yaparak değişkenin tipine karar verebilir. Dart geliştiricileri tip çıkarımı mekanizmasının kullanılmasını önermektedir.
- Dart null güvenliği mekanizması değer atanmamış değişkenlerin kullanılmasından kaynaklanan hataları (**null exception**) engeller. Programcı özellikle belirtmediği sürece Dart değişkenleri null değerini alamaz.
- Sıcak yükleme (**hot reload**) mekanizması sayesinde, yazılan Dart kodları koşturulmakta olan bir uygulamaya enjekte edilebilir. Bu sayede bir kez çalıştırılan bir uygulama üzerinde, uygulama halen çalışıyorumken düzenleme yapmaya devam edilebilir.

## 1.2. Gözle ve Uygula: Dart Uygulamaları

Kod 5'te Dart dilinin pek çok temel özelliği kullanılmaktadır. Bu noktada, ilgili özellikler kısaca açıklanacaktır. İlerleyen başlıklarda her bir özellik daha derinlemesine inceleneciktir.

Dart farklı görevleri olan çekirdek kütüphaneler kullanmaktadır. Temel veri tipleri, listeler ve bu verilerle ilgili işlemler için gerekli araçlar **dart:core** kütüphanesi tarafından sağlanır. Bu kütüphane uygulamalara otomatik olarak eklenir. Fakat, matematiksel işlemler, asenkron işlemleri ya da veri tabanı işlemleri gibi ekstra sayılabilecek görevlerle ilgili araçlar **dart:core** kütüphanesi tarafından sağlanmaz. Bu işlemlerin her biri ile ilgili hazırlanmış kütüphaneler bulunur.

### Çekirdek Kütüphaneler

Dart dilinin çekirdek kütüphaneleri ile ilgili bilgi almak için <https://dart.dev/guides/libraries> adresini ziyaret edebilirsiniz.

Kütüphanelerin uygulamaya eklenmesi için **import** anahtar sözcüğü kullanılır. Kod 5'te 1 numaralı satırda, bu uygulamadaki üs alma işlemini sağlayan **pow** fonksiyonunu içeren **dart:math** kütüphanesi uygulamaya eklenmektedir. Bu kütüphane **pi** ve **e** gibi sabitlerin yanında, kök alma ve trigonometrik değerleri hesaplama gibi yetenekleri uygulalamaya katabilmektedir. **dart:math** kütüphanesi hakkında daha fazla bilgi almak için <https://api.dart.dev/stable/2.14.4/dart-math/dart-math-library.html> adresi ziyaret edilebilir.

```

1 import "dart:math";
2 void main() {
3     var sayı = 40;
4     print(karesiniAl(sayı));
5 }
6 num karesiniAl(sayı) {
7     return (pow(sayı, 2));
8 }

```

Kod 5. Temel Dart uygulaması

Dart uygulamaları **main** adındaki üst düzey fonksiyondan başlatılır. Uygulama bu fonksiyondan başlayarak gerekli dallara ayrılır. 2 numaralı satırda **main** fonksiyonu tanımlanmaktadır. Sıkı veri tipi kontrolü mekanizması nedeniyle değer döndürmesi beklenen fonksiyonların hangi türde değer döndüreceğinin belirtilmesi gerekmektedir. Değer döndürmesi beklenmeyen fonksiyonların ise **void** tipi ile tanımlanması gerekmektedir. 2 numaralı satırdaki **main** fonksiyonu bir değer döndürmediğinden **void** tipi ile tanımlanmıştır. 6 numaralı satırdaki **karesiniAl** fonksiyonu ise **num** tipinde sayısal değer döndürecek şekilde tanımlanmıştır.

Dart dilinde değişkenleri tanımlamak için **var** anahtar sözcüğü ya da tip belirteçleri (ör: **String**, **num**) kullanılır. Değişkenler **var** ile tanımlanırsa, Dart ilk atamada tip çıkarımı mekanizması ile değişkenin tipine karar verir. 3 numaralı satırda **sayı** isimli değişken tanımlanarak **40** değeri bu değişikene atanmaktadır.

Dart dilinde tanımlı fonksiyonlar isimleri ile çağrırlar. Bu fonksiyonlara değer göndermek için parametreler kullanılabilir. 6 numaralı satırda tanımlanan **karesiniAl** fonksiyonu **sayım** adında bir parametre beklemektedir. 4 numaralı satırda, **sayı** değişkeni **karesiniAl** fonksiyonuna parametre olarak gönderilmektedir. **sayı** değişkenindeki **40** değerine **karesiniAl** fonksiyonu içinde **sayım** parametresi yoluyla erişilecektir. 7 numaralı satırda **dart:math** kütüphanesi tarafından sağlanan **pow** fonksiyonu kullanılarak **sayım** değişkenindeki değerin karesi hesaplanmaktadır. Bu değer **return** anahtar sözcüğü kullanılarak geriye çevrilmektedir. **return** anahtar kelimesi ile uygulama 4 numaralı satırdaki **karesiniAl** çağrı noktasına geri dönecektir. Burada **karesiniAl** çağrı ifadesi yerine döndürülen değer **print** fonksiyonuna gönderilerek konsoldan yazdırılmaktadır.

### 1.3. Gözle ve Uygula: Yorum Satırları

Yorum satırları programcıların kendilerine ya da çalışıkları takımlardaki diğer programcılara mesaj bırakmak için kullandıkları yapılardır. Dart ile iki türde yorum satırı eklenebilir. Tek satırı sığabilecek kadar kısa yorumlar eklemek için // işaretini satırın başına eklenir. Dart bu işaretin arkasındaki metni program kodu olarak algılamayacaktır. Kod 6'da 2 ve 3 numaralı satırlar yorum satırı haline getirilmiştir. Bir değişkenin görevinin açıklanması ya da yazılan bir kod satırının işletilmesinin engellenmesi gibi amaçlarla tek satırlık yorumlar kullanılabilir.

### 1.3.1. Yorum satırları

```
1 void main() {  
2     // Dart uygulamaları main() fonksiyonu ile başlar.  
3     // Bu uygulama konsola "uygulama başladı" yazmaktadır.  
4     print("uygulama başladı");  
5 }
```

Kod 6. Yorum satırı kullanımı

### 1.3.2. Yorum alanları

Tek bir satırı aşacak yorumları eklemek için yorum alanları /\* ve \*/ işaretleri ile belirlenir. Yorum alanını başlatmak için /\* kullanılır. Bu kullanımdan sonra \*/ işaretine kadar yazılan her karakter yorum olarak kabul edilir. Bir fonksiyonun işlevi, aldığı parametreler ve çevirdiği değerler gibi uzun sürebilecek yorumların bu şekilde oluşturulması kolaylık sağlayacaktır. Benzer şekilde işletilmesi istenmeyen uzun kod alanları yorum alanı içine alınarak derleyiciden saklanabilir. Kod 7'de 2 ve 5 numaralı satırların arası yorum alanı olarak işaretlenmiştir.

```
1 void main() {  
2     /*  
3         Dart uygulamaları main() fonksiyonu ile başlar.  
4         Bu fonksiyon konsola "uygulama başladı" yazmaktadır  
5     */  
6     print("uygulama başladı");  
7 }
```

Kod 7. Yorum alanı kullanımı

## 1.4. Gözle ve Uygula: İşleçler (Operatörler)

Programlama dilinin derleyicisine bir işlem gerçekleştirmeye komutunu vermek için işaretler kullanılır. Örneğin iki adet değerin toplanması işlemini gerçekleştirmek için bu değerler toplama işaretçi (+) ile birleştirilir. İki değer bir işaret ile birleştirildiğinde bir ifade oluşturulmuş olur. Derleyiciler otomatik olarak bu ifadelerin sonuçlarını üretir. Dart dilinde kullanılabilen işaretlerin temel bir özeti Tablo 1'de sunulmuştur. Son ek işaretler bir ifadenin arkasında eklenen işaretleri ifade eder. Örneğin bir nesnenin bir özelliğine erişmek için . işaretci kullanılır. Ön ek işaretler ise ifadenin önüne getirilerek kullanılan işaretleri ifade eder. Örneğin bir değişkenin içindeki değerin tersini almak için -değiskeni kullanılır. Aritmetik işaretler çarpma/bölme ve toplama/cıkarma kümelerine ayrılmıştır. İlişki kontrolünde ise büyülüklük/küçüklük ve eşitlik kontrolleri yine ayrı kümelerdir. Son olarak kontrol ifadelerini birleştirmek için VE ve VEYHA işaretleri kullanılabilir. Örneğin bir sayının sıfırdan büyük ve 10'dan küçük olduğunu anlamak için bu iki kontrolün VE işaretci (ör: sayı>0 && sayı<10) ile birleştirilmesi gereklidir.

Tablo 1. Dart işaretleri

İşleç Ailesi	İşleçler
--------------	----------

---

Son ek işaretler	<b>ifade++ ifade-- () [] . ?.</b>
Ön ek işaretler	<b>-ifade !ifade ~ifade ++ifade --ifade</b>
Çarpma / Bölme	<b>* / % ~/</b>
Toplama / Çıkarma	<b>+ -</b>
İlişki kontrolü	<b>&gt;= &gt; &lt;= &lt;</b>
Eşitlik kontrolü	<b>== !=</b>
Mantıksal VE	<b>&amp;&amp;</b>
Mantıksal VEYA	<b>  </b>

---

Bir ifade içinde birden çok işaret kullanılabileceğinden bu işaretlerin hangisinin önce işletileceğinin bilinmesi önemlidir. Buna işaret önceliği (**operator precedence**) adı verilir. Yukarıdaki tabloda her bir satırda kümelerden gelen kümelerden daha yüksek önceliklidir. Yani, aynı ifade içinde çarpma ve toplama işaretleri birlikte verilmişse, derleyici önce çarpma, sonra toplama işlemini gerçekleştirecektir. Bu sayede **3\*2+4** ifadesinin sonucu **12** değil, **10** olacaktır.

#### 1.4.1. Aritmetik İşleçler

Değerler üzerinde toplama, çıkarma, çarpma, bölme ve mod alma gibi aritmetik işlemler gerçekleştirilebilir. Kod 8 Dart dilinde kullanılabilen aritmetik işaretleri göstermektedir. Tüm ifadeler print fonksiyonuna gönderilerek işlemin sonucu konsola yazdırılmaktadır. Çıkmazı beklenen sonuç satırın sonunda yorum satırı olarak belirtilmiştir.

```

1 void main() {
2   print(4 + 6); // 10
3   print(3 - 6); // -3
4   var sayı = 10;
5   print(-sayı); // -10
6   print(3 * sayı); // 30
7   print(sayı / 3); // 3.33
8   print(sayı ~/ 3); // 3
9   print(sayı % 3); // 1
10 }
```

Kod 8. Temel aritmetik işaretler

2 numaralı satırda 4 ve 6 değerleri toplanarak 10 değeri elde edilmekte; ardından bu değer **print** fonksiyonu ile konsola yazılmaktadır. 3 numaralı satırda ise 3'ten 6 çıkarılmaktadır. İşleçler değerlerle kullanılabileceği gibi değişkenlerle de kullanılabilmektedir. 4 numaralı satırda **sayı** değişkeni tanımlanarak bu değişkene 10 değeri atamıştır. 5 numaralı satırda işaret değiştirme işaretçi (-) kullanılmıştır. **print** fonksiyonuna 10 değerinin negatif işaretlisi gönderilmiştir. Burada işaret değiştirme yapılsa da değişkendeki değerin değiştirilmediğine dikkat edilmelidir. Bu nedenle 6 – 9 numaralı satırlar arasında pozitif değerler üretilmektedir.

Değişkendeki değerin değiştirilmesi için bir atama işlecinin kullanılması (ör: **sayı = -sayı**) gerekmektedir. 6 numaralı satırda sayı değişkenindeki değer 3 ile çarpılarak 30 değerini üretmektedir. 7 numaralı satırda ise **sayı** değişkenindeki değer 3'e bölünmektedir. 8 numaralı satırda da bölüm işlemi gerçekleştirilmektedir. Fakat bu satırda ~ (**tilda**) sayesinde bölümün tamsayı kısmı alınmaktadır. Bu nedenle 8 numaralı satırın çıktısı **3,3** değil, **3** olarak görülmektedir. 9 numaralı satırda ise **sayı** değişkenindeki **10** değerinin **3**'e göre modu alınmaktadır.

#### 1.4.2. Ön Ek ve Son Ek Arttırma-Azaltma İşleçleri

Ön ek ve son ek işaretleri doğrudan işletilen, uygulandığı ifadeye etki eden işaretlerdir. Ön ek işaretlerin sonuçları ifade işletilmeden önce hesaplanarak değişkene aktarılır. Son ek işaretlerde ise, işlecin etkisi ifade işletildikten sonra değişkene aktarılır.

```
1 void main() {  
2     var sayı = 10;  
3     print(++sayı); // 11  
4     print(sayı++); // 11  
5     print(sayı); // 12  
6     print(sayı--); // 12  
7     print(--sayı); // 10  
8     print(sayı); // 10  
9 }
```

Kod 9. Arttırma ve azaltma işaretleri

Kod 9'da 2 numaralı satırda **sayı** değişkeninin değeri **10** olarak belirlenmiştir. 3 numaralı satırda **sayı** değişkeninin değeri ön ek arttırma işaretçi ile arttırıldıktan **print** işleminden önce sayının değeri artırılmaktadır. Bu nedenle bu satırın sonucu **11** olur. 4 numaralı satırda ise **sayı** değişkeninin değeri son ek arttırma işaretçi ile düzenlenmektedir. Bu nedenle önce değişkenin değeri **print** ile yazılmakta, ardından arttırma işlevi gerçekleştirilmektedir. Bu nedenle satırın sonucu **11** olarak yazılmaktadır fakat bu işlemin sonunda **sayı** değişkeninin değeri **12**'dir. 5 numaralı satırda bu değer doğrudan yazdırılarak gözlenmektedir. Aynı durum azaltma işaretleri ile 6,7 ve 8 numaralı satırlarda tekrarlanmaktadır.

#### 1.4.3. İlişkileri Kontrol Etme İşleçleri

Verilen değerlerin eşitliğini ya da büyüğünü kontrol etmek için ilişkisel işaretler kullanılabilir. Bu ifadelerin sonucu **Boolean** türünde verilir. Verilen test ifadesi doğru ise **true**; değilse **false** değeri döner.

```

1 void main() {
2     print(2==2); // true
3     print(2!=2); // false
4     print(2>2); // false
5     sayi = 2;
6     print(sayi>=2); // true
7     print(3<=sayi); // false
8 }
```

Kod 10. İlişkisel işlemler

Kod 10'da 2 numaralı satırda 2 sayısının 2'ye eşitliği sorgulanıp sonuç ekrana yazdırılmaktadır. 3 numaralı satırda ise 2 sayısının 2'ye eşit olmama durumu sorgulanmaktadır. Bu sorgulamanın sonucu yanlış (**false**) olarak dönmektedir. Benzer şekilde 4 numaralı satırda 2'nin 2'den büyük olma durumu (**false**) sorgulanmaktadır. 5 numaralı satırda **sayi** değişkenine **2** değeri atanmaktadır. 6 numaralı satırda **sayi** değişkenindeki değerin 2'den büyük ya da 2'ye eşit olma durumu sorgulandığından sonuç (**true**) doğrudur. 7 numaralı satırda ise **sayi** değişkenindeki değerin 3'e eşit ya da 3'ten büyük olma durumu sorgulandığından sonuç yanlış (**false**) olarak üretilmiştir.

#### 1.4.4. Atama ve Birleşik Atama İşleçleri

Değerlerin değişkenlere atanması için **=** işlemci kullanılır. Kod 11'de bir numaralı satırda **sayi** değişkenine **2** değeri atanmaktadır. Bunun yanında Dart birleşik atama işaretlerini de destekler. Bu işaretlerde aritmetik işlemler ve atama işlemi bir arada yapıldığından kısayol bir kullanım sunulmaktadır.

```

1 void main() {
2     sayi = 2; // 2
3     sayi += 3; // sayi = sayi + 3;
4     print(sayi); // 5
5     sayi *= 3; // sayi = sayi * 3;
6     print(sayi); // 15
7     sayi ~/= 4;
8     print(sayi); // 4
9 }
```

Kod 11. Atama ve birleşik atama işaretleri

Örneğin, Kod 11'de 3 numaralı satırda **sayi** değişkeni **3** ile toplanıp elde edilen değer tekrar **sayi** değişkenine atanmaktadır. Bu işlemin açılımı yorum satırı halinde yine 3 numaralı satırda belirtilmiştir. Birleşik işaretlerin kullanılmasında önemli bir hata tanımlanmış fakat değer atanmamış değişkenler üzerinde işlem yapılmasıdır. Bu tür değişkenlerin değeri **null** olduğundan birleşik atama işlemlerinin kullanılması bir tip uyumuzluk hatası oluşturur (**TypeError: null has no properties**). Bu hatadan kaçınmak için tanımlanan her değişkene ilk değer atanması önerilmektedir. Kod 11'de 5 numaralı satırda **sayi** değişkenindeki değer **3** ile çarpılarak yine **sayi** değişkenine gönderilmektedir. Benzer

şekilde 7 numaralı sayıda ise **sayı** değişkenindeki değer **4**'e bölünerek tamsayı kısmı değişkene atanmaktadır.

#### 1.4.5. Durumsal Atama

Dart tek satırlık durumsal atama işleçini desteklemektedir. Bu işaret bir **if-else** yapısı gibi düşünülebilir. Durumsal atama işlemci **if-else** yapılarındaki blok açma-kapama parantezleri gibi karakterleri içermediginden yine bir kısayol yazım gibi düşünülebilir.

```
1 void main() {  
2     var sayı = 3;  
3     var sonuc = (sayı % 2 == 0) ? "çift" : "tek";  
4     print(sonuc); // tek  
5     /*  
6     var sonuc;  
7     if(sayı % 2 ==0)  
8     {  
9         sonuc = "çift";  
10    }  
11    else  
12    {  
13        sonuc = "tek";  
14    }  
15    */  
16 }
```

Kod 12. Durumsal atama işlemci

Kod 12'de 2 numaralı satırda tanımlanan **sayı** değişkenindeki değerin tek ya da çift olmasına göre **sonuc** değişkenine **tek** ya da **çift** ifadeleri atanmaktadır. Bu atama işlemi 3. Satırda gerçekleştirilmektedir. Sorgulama için parantezler içinde **sayı** değişkeninin 2'ye göre modu alınarak sonucun **0** olma durumu kontrol edilmektedir. **?** karakterinden sonra **:** karakterine kadar verilen değer (**çift**) sorgulamanın doğru olması halinde sonuç değişkenine atanacak değeri temsil eder. Bu alan **if-else** yapısındaki **if**bloğu gibi düşünülebilir. **:** karakterinden sonra verilen değer ise sorgulamanın yanlış değer döndürmesi halinde değişkene atanacak değeri temsil etmektedir. Bu alan da **if-else** yapısındaki **else** bloğu gibi düşünülebilir. Kod 12'de 6-14 numaralı satırlar arasında 2 numaralı satırda gerçekleştirilen işlemin **if-else** yapısı ile uygulanışı gösterilmektedir. Görüldüğü üzere bu işaret kodu oldukça kısaltmaktadır. Bununla birlikte, bu kullanım yalnızca atama işlemlerine olanak tanır. Verilen şartın doğru olması halinde farklı işlemler yapılmak istenirse **if-else** yapısının kullanılması gerekecektir.

#### 1.4.6. Mantıksal İşleçler

Boolean türündeki mantıksal değerler üzerine işlem yapmak için mantıksal işaretler kullanılır. Kod 13'te anlaşılabilirliği kolaylaştmak adına doğrudan **Boolean** değerler üzerinde çalışılmıştır. Gerçek bir programlama senaryosunda bu değerlerin mantıksal sınımlarla üretilmesi (ör: **2==2** ya da **2 > 3**) ya da bir değişkende tutulması (ör: **ciftmi = false**) gerekecektir.

```

1 void main() {
2     print(!true); // false
3     print(true && true); // true
4     print(true && false); // false
5     print(true || false); // true
6 }
```

Kod 13. Mantıksal işleyiciler

Boolean bir değerin tersini almak için `!` işleyici kullanılır. Kod 13'te 2 numaralı satırda `!true` ifadesi ile `false` değeri üretilmektedir. Mantıksal VE işlemi için `&&` işleyici kullanılır. Mantıksal VE, her iki tarafındaki değerin doğru olması halinde `true`; en az birinin yanlış olması halinde ise `false` değeri üretir. Kod 13'te 3 numaralı satırda iki `true` değeri VE ile birleştirildiğinden sonuç `true` çıkmaktadır. 4 numaralı satırda ise bir `true` ve bir `false` VE ile birleştirildiğinden sonuç `false` olarak üretilir. Mantıksal VEYA işlemi içinse `||` işleyici kullanılır. Mantıksal VEYA işleyici verilen ifadelerden en az birinin doğru olması halinde `true` değeri üretir. 5 numaralı satırda bir `true` ve bir `false` VEYA işlemi ile birleştirildiğinden sonuç `true` olarak üretilecektir.

### Aliştırma

Tanımladığınız bir sayının 3 ile 20 arasında olma durumunu inceleyen Dart kodunu üretiniz.

## 1.5. Gözle: Anahtar Sözcükler

Her programlama dilinde olduğu gibi, Dart dilinin de rezerve edilmiş anahtar sözcükleri bulunmaktadır. Bu anahtar sözcüklerin değişken, fonksiyon ya da sınıf ismi olarak kullanılması gerekmektedir. Dart dilinin anahtar sözcüklerinin listesine <https://dart.dev/guides/language/language-tour#keywords> adresinden ulaşılabilir.

## 1.6. Gözle ve Uygula: Değişkenler

Değişkenler uygulamalar için bellek alanlarında tutulan değerlere referans veren kayıtlardır. Örneğin, Kod 14'te `sayı` ismindeki değişken, `int` türündeki `3` değerine bir referansdır. Değişkende tutulan değer atama işleçleri kullanılarak düzenlenebilir. Kodlarda `sayı` ifadesi kullanıldığında derleyici bu ifadeyi referans verdiği `3` değeri ile değiştirerek işleme sokmaktadır. Bu sayede, 2 numaralı satırındaki ifadenin sonucunda `sayı` değişkenine 1 eklenecek ekrana `4` yazdırılmaktadır.

```

1 void main() {
2     var sayı = 3;
3     print(++sayı); // 4
4 }
```

Kod 14. Değişken tanımlamak

Değişkenleri tanımlamak için `var` anahtar sözcüğü kullanılır. Bunun yanında, doğrudan değişkende saklanacak veri türünü ifade ederek de değişken tanımlaması yapılabilir. Örneğin,

Kod 14'te 2 numaralı satırda **sayı1** isimli değişken **var** ile; **sayı2** isimli değişken ise **int** ile tanımlanmıştır. Her iki değişkende de **10** değeri saklanmaktadır. Dart değişkene atanınan değere bakarak tip çıkarımı yapabilmektedir. Bu nedenle değişkenlerin tip tanımlaması ile değil, **var** anahtar sözcüğü ile tanımlanması önerilmektedir.

### Tip Çıkarmı

Dart değişkene atanınan değere göre hangi tipte olması gerektiğine karar verebilmektedir. Bu nedenle değişkenlerin **var** anahtar sözcüğü ile tanımlanması önerilmiştir.

Tanımlanan bir değişkenin tekrar değiştirilmesi istenmiyorsa bu değişken **final** anahtar sözcüğü ile tanımlanabilir. Kod 15'te 2 numaralı satırda isim değişkeni **final** anahtar sözcüğü ile tanımlanmıştır. Bu nedenle 3 numaralı satırda bu değişkene gerçekleştirilen atama yorum satırı olarak verilen hatayı üretecektir.

```
1 void main() {  
2     final isim = "Dart";  
3     isim = "Programlama"; //The final variable 'isim' can only be set once.  
4     const versiyon = 2.12;  
5     versiyon = 3; // Constant variables can't be assigned a value.  
6 }
```

*Kod 15. final ve const kullanımı*

Derleme süreci sabitlerinin tanımlanması için **const** anahtar kelimesi kullanılır. Kod 15'te 4 numaralı satırda versiyon değişkeni sabit olarak tanımlanmıştır. Sabit olarak tanımlanan bu değişkene atama gerçekleştirilmesi yine bir hata üretecektir (Ör: **Constant variables can't be assigned a value.**).

## 1.7. Gözle ve Uygula: Null-Safety Mekanizması

Null-Safety mekanizması içeriği **null** olan nesnelerin kullanılmasından kaynaklanan çalışma zamanı hatalarını önlemek için geliştirilmiştir. **Null** değer kaynaklı hatalar kolaylıkla gözden kaçabileceğinden oldukça yaygındır. Örneğin, Kod 16'da 2 numaralı satırda **int** tipindeki **sayı** değişkeni tanımlanmış fakat değer ataması yapılmamıştır. Bu durumda değişkenin içeriği boş, yani **null**'dur. 3 numaralı satırda **sayı** değişkenindeki değer **2** ile çarpılmak istenmektedir. Bu kod, **null** değerini **2** ile çarpmaya çalıştığından çalıştırıldığı anda bir hata (ör: **TypeError: null has no properties**) üretecektir.

```
1 void main() {  
2     int sayı;  
3     print(sayı*2);  
4 }
```

*Kod 16. null hatalı örneği*

Bu mekanizma Dart dilinin 2.1.2 numaralı versiyonu ile getirilmiştir. Bu mekanizma etkinleştirildiğinde Dart kodu inceleyerek çalışma zamanından önce hata mesajı (Ör: **The**

**non-nullable local variable 'sayi' must be assigned before it can be used**) verecektir.

Bir değişken tanımlanırken **null** değer alabilir (**nullable**) olarak belirtilebilir. Bu amaçla değişkenin tip belirtecinin arkasına **? işlevi eklenir**. Kod 17'de 2 numaralı satırda tanımlama null güvenliği mekanizması tarafından yakalanarak hata (Ör: **A value of type 'Null' can't be assigned to a variable of type 'int'**.) üretir. Fakat 3 numaralı satırda tanımlamada tip belirteci **int**'nin arkasında **? işlevi** olduğundan **sayi2** değişkenine null değer atanması hata mesajı üretmez. Bu tanımlamada tip belirtecinin kullanımı zorunludur. **var** anahtar sözcüğü arkasında **? kullanımı** hata üretecektir.

```
1 void main() {  
2     int sayi = null;  
3     int? sayi2 = null;  
4 }
```

Kod 17. Null güvenliği açıkken, bir değişkeni null değer alabilecek şekilde belirlemek

null değer alabilen değişkenlerle çalışırken içeriklerinin kontrol edilmesi gereklidir. Kod 18'de 2 numaralı satırda versiyon değişkeni null değer alabilecek şekilde tanımlanmıştır.

```
1 void main() {  
2     String? versiyon;  
3     print(versiyon.length);  
4     print(versiyon!.length);  
5     // (versiyon != null) ? versiyon.length : null;  
6 }
```

Kod 18. Değişkenlerde null kontrolü

3 numaralı satırda bu değişkenin içindeki değerin uzunluğunu veren **length** özelliğine erişilmeye çalışılmaktadır. Null-Safety mekanizması bu değişkenin içeriğinin **null** olabileceği ön görerek bir hata üretir (Ör: **The property 'length' can't be unconditionally accessed because the receiver can be 'null'**). Bu hatadan kaçınmak için **!** işlevi kullanılır. Bu işaret kullanıldığında derleyici geri planda 5 numaralı satırda açıkça yazılan kontrolü yapmaktadır.

## Tartışma

Pek çok uygulama **null** değer kaynaklı hatalardan dolayı çakılabilmektedir. Yine de programlama dilleri **null** değerleri kullanmaya devam etmektedir. Gördüğünüz gibi null güvenliği mekanizması varken, Dart bazı değişkenleri **null** değer alabilir şeklinde tanımlamak amacıyla özel bir sistem sunmaktadır. Neden uygulamalarda **null** değere ihtiyaç duyabileceğinizi tartışınız.

## 1.8. Gözle ve Uygula Öntanımlı Veri Tipleri

### 1.8.1. Sayılar

Dart iki tür sayısal veri tipi sağlar. Tamsayılar için **int** veri tipi kullanılır. **int** veri tipindeki tam sayıların sınırları kullanılan platforma göre değişmektedir. Mobil cihazlarda 64 bit kullanılabileceğinden **int** sayılar  $-2^{63}$  ile  $2^{63}-1$  arasında değişir. Web uygulamalarında ise bu sınırlar  $-2^{53}$  ile  $2^{53}-1$  arasındadır.

#### Tartışma

**int** veri tipinin sınırları negatif değerler için  $2^{53}$ 'e kadar uzanırken, pozitif değerlerde bir sayı eksiktir. Bu eksikliğin nedeni ne olabilir?

```
1 void main() {  
2     var sayı = 1;  
3     var onaltılık = 0xA00;  
4     var ustel = 8e5;  
5     var ondalıklı = 1.1;  
6     var ustel2 = 1.42e5;  
7 }
```

Kod 19. Kullanılabilecek sayısal değerler

Kod 19'da 2 numaralı satırda **sayı** değişkeni **1** değerini tutacak şekilde tanımlanmıştır. 3 numaralı satırda ise **onaltılık** değişkeni on altılık sayı sistemindeki **0xA00** (onluk sistemde 2560) olmasını tutacak şekilde tanımlanmıştır. 4 numaralı satırda **ustel** değişkeni **8e5** değerini (onluk sistemde 800.000) değerini tutacak şekilde tanımlanmıştır.

#### Sayısal Değerler

10'luk sayı sisteminde bir değer doğrudan ifade edilebilir.

16'lık sayı sistemindeki değerler 0x ön eki ile tanımlanır.

10'un katlarını gösteren üstel sayılar için ise sayı değerinin ardından e ve hemen ardından bu sayının 10'un kaçinci kuvveti ile çarpılacağını belirten basamak sayısı verilir.

Virgülden sonraki ondalıklı değerlerin tutulması içinse **double** veri tipi sağlanmıştır. Dört numaralı satırda **ondalıkli** değişkeni **1.1** olmasını tutacak şekilde tanımlanmıştır. 6 numaralı satırda ise **142000** sayısı ( $1,42 * 10^5$ ) **ustel2** değişkenine aktarılmıştır.

### 1.8.2. Metinler

**String** nesneleri **UTF-16** standardındaki karakterlerden oluşan katarlardır. **String** tipinde nesneler üretmek için karakterler tek tırnak ya da çift tırnaklar arasına alınır.

```

1 void main() {
2     var s1 = 'tek tırnak içinde metin.';
3     var s2 = "çift tırnak içinde metin.";
4     var s3 = 'Tek tırnak içinde kesme (\') işaretini göstermek.';
5     var s4 = "Çift tırnak içinde kesme ('') işaretini göstermek";
6     var s = 'ekleyebilir';
7     print('Dart metinler içine farklı metinleri $s');
8 }
```

Kod 20. Metin kullanım örnekleri

Kod 20'de 2 numaralı satırda tek tırnaklarla oluşturulmuş bir string **s1** değişkenine atanmaktadır. 3 numaralı satırda ise çift tırnaklar arasında oluşturulmuş bir string **s2** değişkenine atanmaktadır. İki üretim yolu arasında bir fark bulunmamaktadır. Stringler tırnaklar arasında oluşturulduğundan, bu karakterlerin String nesnelerine eklenmesi için kaçırlılması gereklidir. Bu amaçla backslash (\) karakteri kullanılır. 4 numaralı satırda tek tırnaklarla oluşturulmuş String içinde ‘ karakterinin gösterilebilmesi için önüne \ eklenmiştir. Bu değişken **print** fonksiyonu ile yazıldığında \ karakterinin basılmadığı görülebilir. Bu tanımlamada \ karakterinin kullanılmaması bir hata (**ör: Untermminated String literal – Sonlandırılmış String nesnesi**) üretecektir.

Dart **String** nesneler içine değişkenleri enjekte edebilir. Bu amaçla String nesnesi üretilirken ifadenin önüne \$ karakteri koyması yeterlidir. Kod 20'de 6 numaralı satırda **s** değişkeni tanımlanarak ekleyebilir sözcüğü bu değişkene atanmaktadır. 7 numaralı satırda bir String oluşturularak **s** değişkeni **\$s** kullanımı youyla içine enjekte edilmektedir. Bu amaçla iki String + işlevi ile de birleştirilebilmektedir (**ör: 'Dart metinler içine farklı metinleri' + 'ekleyebilir'**).

### 1.8.3. Boolean

Mantıksal verilerin tutulması için Dart **bool** veri tipini destekler. **bool** tipindeki değişkenler yalnızca **true** ya da **false** değerlerini alabilir.

```

1 void main() {
2     var tekMi = true;
3     var ciftMi = false;
4 }
```

Kod 21. Boolean değişkenler

Kod 21'de 2 numaralı satırda **bool** tipinde bir nesne üretilerek değeri **true** olarak belirlenip bu nesne **tekMi** değişkenine atanmaktadır. 3 numaralı satırda yine **bool** tipinde bir nesne üretilerek bu sefer değeri **false** olarak belirlenmektedir. Ardından bu nesne **ciftMi** değişkenine atanmaktadır.

### 1.8.4. Listeler

Programlama dillerinde en çok kullanılan dizi yapılarından biri listelerdir. Listeler benzer özellikteki nesnelerin (Ör: sayılar, metinler) gruplarından oluşur.

Kod 22'de 2 numaralı satırda bir liste oluşturularak **sayilar** değişkenine atanmaktadır. Bu liste **int** türündeki nesnelerden oluşmaktadır. Dart otomatik olarak bu listenin tipini çıkarsar. Bu listeye **String** türünden bir eleman eklenmeye çalışılırsa hata oluşacaktır (Ör: **The argument type 'String' can't be assigned to the parameter type 'int'**).

```
1 void main() {  
2     var sayilar=[1, 2, 4];  
3     var alisverisListem=[  
4         "ekmek",  
5         "süt",  
6         "yoğurt",  
7     ];  
8     print(alisverisListem[0]);  
9     print(alisverisListem.toString());  
10 }
```

Kod 22. Listeleri tanımlama ve liste elementlerine ulaşma

Liste tanımlamalarının tek bir satırda bitirilmesi zorunlu değildir. Kod 22'de 3 numaralı satırda **String** elemanlar içeren bir liste oluşturularak bu liste **alisverisListem** değişkenine atanmaktadır. Bu listenin elemanları 4, 5 ve 6 numaralı satırlarda eklenmiştir. 6 numaralı satırda **yoğurt** elementinin arkasındaki virgül bir hata oluşturmaz. Okunurluğu arttırmak adına bu virgül eklenebilir.

Bir listenin elementlerine erişmek için listenin atandığı değişkenin arkasından köşeli parantezler içinde elementin sıra numarası verilir. Dart listeleri 0 sıra numarası ile başlatılır. Bu nedenle Kod 22'de 8 numaralı satırda **alisverisListem[0]** ifadesi **ekmek** değerini döndürecekiktir. Bir listenin tümünü String olarak almak için **toString()** metodu kullanılabilir. Bu metot liste içeriğini **String** türünde çevirir.

Listelerde tanımlı elementler atama işlemci kullanılarak değiştirebilir. Kod 23'te 2 numaralı satırda **alisverisListem** listesindeki 2 sıra numaralı elementin içeriği **meyve suyu** olacak şekilde güncellenmektedir. 3 numaralı satırda ise **sayilar** listesindeki 3 sıra numaralı element **5** olacak şekilde güncellenmektedir.

```
1 void main() {  
2     alisverisListem[1]="meyve suyu";  
3     sayilar[2]=5;  
4     sayilar[3]=10;  
5 }
```

Kod 23. Liste elementlerini değiştirme ve listenin sonuna element ekleme

Listelerde tanımlı bulunmayan bir sıra numarasına atama yapılamaz. Kod 23'te 4 numaralı satırda **sayilar** listesindeki 4 sıra numaralı elemente atama yapılmaktadır. Bu element boş olduğundan bu satır bir hata üretecektir.

Listelerin sonuna element eklemek için **add** metodu kullanılır. Kod 24'te 2 numaralı satırda sayılar listesinin sonuna değeri **10** olan bir element eklenmektedir. Listede tanımlı olan bir elementi silmek için **removeAt** metodu kullanılabilir. 3 numaralı satırda **alisverisListem** listesinden 3 sıra numaralı element çıkarılmaktadır. Çıkarma işlemi sonunda listedeki ardıl elementlerin sıra numaraları düşürülür.

```
1 void main() {  
2     sayilar.add(10);  
3     alisverisListem.removeAt(2);  
4     alisverisListem.insert(2, "zeytin");  
5     print(alisverisListem.toString());  
6 }
```

Kod 24. Listelere element ekleme ve çıkarma

Listede araya bir element eklemek için **insert** metodu kullanılır. Kod 24'te 4 numaralı satırda **alisverisListem** listesinde 3. Sıraya **zeytin** elementi eklenmektedir.

Listelerdeki elementler üzerinde işlem gerçekleştirmek için **forEach** metodu kullanılabilir. Bu metot her bir elementi kendisine parametre olarak verilen anonim fonksiyona gönderir.

```
1 void main() {  
2     alisverisListem.forEach((element)=>print(element));  
3 }
```

Kod 25. **forEach** metodu ile liste elementlerini kullanma

Kod 25'te tanımlı **alisverisListem** listesinin **forEach** metodu ile her bir element konsolda ayrı bir satırda yazdırılmaktadır. **forEach** metodundaki ilk parantezler içine liste elementinin tanımlanacak anonim fonksiyona hangi isimle gönderileceği (parametre adı) belirlenmektedir. Bu örnekte parametre adı **element** olarak belirlenmiştir, fakat istenen herhangi bir değişken ismi seçilebilir. => işlecinin arkasında bu fonksiyonda işletilecek tek satırlık kod yazılarak işlem tamamlanır.

Listelerde arama yapmak içi **firstWhere** metodu kullanılabilir. Bu metot listedeki elementleri verilen test ifadesine göre tarar. Test ifadesini sağlayan ilk elementi geri çevirir. Test ifadesi sağlanmazsa **orElse** bloğundaki ifade çevrilebilir.

Kod 26'da **alisverisListem** listesinde “**gazete**” araması yapılmaktadır. Bu aramada gazeteye eşit olan ilk element çevrilir. Bu tarama başarısız olursa metot “**aranan ifade bulunamadı**” değerini çevirecektir. **firstWhere** metodu aranan elementin kendisini çevirirken; **indexWhere** metodu aranan ifadenin sıra numarasını çevirir. Arama başarısız olursa **indexWhere** metodu -1 çevirir.

```

1 void main() {
2     var sonuc = alisverisListem.firstWhere(
3         (element) => element == "gazete",
4        orElse: () {
5             return "aranan ifade bulunamadi";
6         });
7     print(sonuc);
8 }
```

*Kod 26. firstWhere metodu ile listede arama yapma*

Kod 27'de **alisverisListem** listesinde **süt** elementinin kaçinci sırada yer aldığı aranmaktadır. **indexWhere** metodunun ürettiği sıra numarası değeri **sonuc** değişkenine atanmaktadır. 5 numaralı satırda üretilen sıra numarasının -1'den büyük olma durumu kontrol edilmektedir. Bu yolla listede gerçekleştirilen aramanın başarısı sınanmaktadır. Listelerde sıra numaraları sıfırdan başlatıldığından çıktıının anlaşılabilir olması için 6 numaralı satırda **sonuc** değişkenine 1 eklenmektedir. Üretilen sonuç 7 numaralı satırda kullanıcıya bildirilmektedir. **sonuc** değişkeninin değerini -1 olması aramanın başarısız olduğu anlamına geldiğinden **else** bloğunda aranan ifadenin listede yer almadağına ilişkin bilgilendirme yapılmaktadır.

```

1 void main() {
2     var sonuc=alisverisListem.indexWhere(
3         (element)=> element == "süt"
4     );
5     if(sonuc>-1){
6         sonuc++;
7         print("aranan ifade $sonuc. sırada");
8     }
9     else
10    {
11        print("aranan ifade bulunamadi");
12    }
13 }
```

*Kod 27. indexWhere metodu ile listede arama yapmak*

İki listenin birleştirilmesi için yayılım işlevi (...) kullanılabilir. Bu işlem listelerden birinin taranarak elementlerin teker teker diğerine eklenmesi yoluyla da gerçekleştirilebilir. Fakat yayılım işlevinin kullanımı çok daha kolaydır.

```
1 void main() {  
2     var dogalSayilar=[0, 1, 2, 4];  
3     var tamSayilar = [-1, -4, -11, ...dogalSayilar];  
4     print(tamSayilar.toString());  
5 }
```

Kod 28. Listeleri birleştirmek

Kod 28'de 21 numaralı satırda **dogalSayilar** listesi, 3 numaralı satırda ise **tamSayilar** listesi oluşturulmaktadır. 3 numaralı satırda **tamSayilar** listesinde ilk elementlerin tanımlanmasının ardından yayılım işlecinin arkasından **dogalSayilar** listesinin referansı verilerek (Ör: **...dogalSayilar**) iki listenin birleştirilmesi sağlanmıştır.

## Alıştırma

**tamSayilar** ve **dogalSayilar** listelerini ayrı ayrı tanımlayarak, dogalSayilar listesindeki elementleri tamSayilar listesine forEach metodunu kullanarak ekleyiniz (Kod 29).

```
1 void main() {  
2     var dogalSayilar = [1,2,3];  
3     var tamSayilar = [-1, -2, -3];  
4     dogalSayilar.forEach((sayi)=>tamSayilar.add(sayı));  
5     print(tamSayilar.toString());  
6 }
```

Kod 29. forEach kullanarak listelerin birleştirilmesi

## İnceleme

Yer kısıtlamaları nedeniyle listelerin tüm özellikleri ve metotları bu kitapta anlatılamamıştır. Liste sınıfının tüm özellikleri ve metodlarına ulaşmak için [### 1.8.5. Kümeler](https://api.dart.dev/stable/2.14.4/dart-core>List-class.html</a> adresini ziyaret edebilirsiniz.</p></div><div data-bbox=)

Kümeler, adından da anlaşılabileceği gibi birbirinden farklı elementlerin sırasız bir listesidir. Listelerden farklı olarak kümelerde aynı ifade iki farklı element olarak yer alamaz.

```
1 void main() {  
2     var haftaIci = {"salı", "çarşamba", "perşembe", "cuma"};  
3     var gunler = <String>{};  
4 }
```

Kod 30. Küme tanımlamak

Kümeler, küme parantezleri içinde tanımlanır. Parantezler içinde aynı veri tipinde, eşsiz elementler eklenir. Kod 30'da 2 numaralı satırda haftanın bazı günlerini içeren bir küme oluşturularak **haftaIci** isimli değişkene atanmaktadır. Dart boş küme üretmeye de izin verir.

Bu senaryoda 3 numaralı satırda gösterildiği gibi kümede yer alacak elementlerin veri tipinin (Ör: <int>) kümeye parantezleri öncesinde belirtilmesi gereklidir.

Kümelere element eklemek için add metodu kullanılır. Kod 31'de 3 numaralı satırda **gunler** kümeye **pazartesi** değeri eklenmektedir. Hazır bir kümeye başka bir kümeye addAll metodu ile eklenebilir. Kod 31'de 5 ve 6 numaralı satırlarda **haftaIci** ve **haftaSonu** kümeleri, **gunler** kümeye **addAll** metodu kullanılarak eklenmektedir.

```
1 void main() {  
2     var haftaSonu = {"cumartesi", "pazar"};  
3     gunler.add("pazartesi");  
4     print(gunler.toString());  
5     gunler.addAll(hftaIci);  
6     gunler.addAll(hftaSonu);  
7     gunler.add("pazartesi");  
8     print(gunler.toString());  
9 }
```

Kod 31. Kümelere element eklemek

Kümelerde aynı değerin birden fazla kez yer alınmasına izin verilmez. Kod 31'de 7 numaralı satırda, **gunler** kümeye yer alan **pazartesi** değeri tekrar kümeye eklenmeye çalışılmaktadır. Bu işlem bir hata üremesine de kümeye incelendiğinde değerin ikinci kez kümeye eklenmediği görülecektir.

Kümedeki elementlere erişmek için **elementAt** metodu kullanılır. Bu metot bir sıra numarası ile çağrılr. Listelerde olduğu gibi kümeler de sıfır sıra numarası ile başlar. Kod 32'de 2 numaralı satırda **gunler** kümelerindeki 0 sıra numaralı element **ilkGun** değişkenine atanmaktadır. Bu işlem 3 numaralı satırda gösterildiği gibi kümeye sınıfının **first** özelliği kullanılarak da yapılabilir.

```
1 void main() {  
2     var ilkGun=gunler.elementAt(0);  
3     var ilkGun=gunler.first;  
4     var gunSayisi = gunler.length;  
5     print("günler kümelerinde $gunSayisi adet gün var.");  
6 }
```

Kod 32. Kümeye elementlerine ve kümedeki element sayısına erişmek

Bir kümede kaç element olduğunu öğrenmek için **length** özelliği kullanılır. Kod 32'de 4 numaralı satırda **gunler** kümelerinin eleman sayısı **gunSayisi** değişkenine atanmaktadır. 5 numaralı satırda ise kümedeki gün sayısına ilişkin bir bilgi mesajı yazdırılmaktadır.

**forEach** metodu listelerdekine benzer şekilde kümelerde de sağlanmaktadır. Kod 33'te **gunler** kümelerindeki elementler **forEach** metodu ile taranarak her biri konsola yazdırılmaktadır.

```
1 void main() {  
2     gunler.forEach((element)=> print(element));  
3 }
```

Kod 33. Kümedeki tüm elementleri taramak

Kümelerde bir elementi aramak için **lookup** metodu kullanılır. Bu metot, aranan nesnenin bulunması halinde bu elementi; aksi halde **null** değerini çevirir. Kod 34'te 2 numaralı satırda **lookup** metodu kullanılarak gunler kümesinde **sali** değeri aranmaktadır. Metodun döndüreceği değerin **null** olmaması kontrolü yapılarak **sali** değerinin kümede bulunduğu anlaşılmaktadır.

```
1 void main() {  
2     if(gunler.lookup("sali")!=null)  
3     {  
4         gunler.remove("sali");  
5     }  
6 }
```

Kod 34. Kümede bulunan bir elementi silmek

Kümelerden element silmek için **remove** metodu element referansı ile çağırılır. Kod 34'te 4 numaralı satırda **remove** metodu **sali** değeri ile çağrılarak silinmektedir. Bu metot elementin silinmesi halinde **true**; aksi halde **false** değeri döndürmektedir.

Kümelerden belirli bir şartı sağlayan elementlerin silinmesi için **removeWhere** metodu kullanılır. Bu metot bir test fonksiyonu ile çağırılır. Kod 35'te 2 numaralı satırda **sayilar** kümesinde **9**'dan büyük olan elementler silinmektedir.

```
1 void main() {  
2     sayilar.removeWhere((element)=> element>9);  
3     sayilar.clear();  
4 }
```

Kod 35. Kümeyi boşaltmak ve kümede belirli şartları sağlayan elementleri silmek

Kümedeki tüm elementlerin silinmesi için **clear** metodu kullanılır. Bu metot herhangi bir parametre almaz; geriye bir değer çevirmez. Kod 35'te 3 numaralı satırda **sayilar** kümesi boşaltılmaktadır.

**lookup** metoduna benzer şekilde, **contains** metodu da küme içinde bir element arar. Fakat **contains** metodu elementin bulunması halinde **true**, aksi halde **false** değerini döndürür.

```

1 void main() {
2     var aranan = "cumartesi";
3     if(gunler.contains(aranan))
4     {
5         print("$aranan günler kümesinde yer alıyor.");
6     }
7     else
8     {
9         print("$aranan günler kümesinde yer almıyor");
10    }
11 }
```

Kod 36. Küme içinde arama yapmak

Kod 36'da **cumartesi** değerinin kümede bulunup bulunmadığı **contains** metodu ile kontrol edilerek gerekli bilgilendirme mesajları verilmektedir.

Kümede belirli bir koşulu sağlayan elementleri bulmak için **where** metodu kullanılır. Bu metot bir test fonksiyonu ile çağırılır. Bu testi sağlayan elementlerin kümesi geri çevrilir.

```

1 void main() {
2     var sayilar = {1, 2, 3, 6, 10, 12};
3     var ondanKucukSayilar = sayilar.where((element)=> element<10);
4     print("sayilar kümesindeki ondan küçük sayılar: $ondanKucukSayilar");
5 }
```

Kod 37. Belirli bir koşulu sağlayan kümeye elementlerini elde etmek

Kod 37'de 2 numaralı satırda sayılarından oluşan bir kümeye tanımlanarak **sayilar** değişkenine atanmaktadır. 3 numaralı satırda bu kümedeki **10**'dan küçük değerlerden oluşan yeni bir kümeye üretmek için **sayilar** kümesi **where** metodu ve **element<10** testi ile çağırılır. Üretilen yeni kümeye **ondanKucukSayilar** değişkenine atanmaktadır. 4 numaralı satırda ise bu kümeyi içeriği **print** fonksiyonu ile yazılmaktadır.

#### 1.8.6. Haritalar

Haritalar bir anahtar ve buna bağlı bir değerden oluşan nesnelerin bir kümeleridir. Haritalarda anahtarlar eşsiz olmak zorundadır. Fakat bir değer birden fazla anahtara atanabilir. Anahtarlar ve değerler farklı veri türlerinde olabilir. Örneğin **int** tipinde bir anahtara **String** tipinde veriler atanabilir.

```

1 void main() {
2     var elementler = {
3         1: "hidrojen",
4         3: "lityum",
5     };
6     var baskentler = Map<String, String>();
7     var soyGazlar = Map<int, String>();
8 }
```

Kod 38. Harita oluşturmak

Kod 38'de üç farklı harita oluşturulmuştur. 2 ve 5 numaralı satırlar arasında harita oluşturma kalığı ile elementler haritası oluşturulmuştur. Bu yöntemde haritada en az bir element bulunduğundan ve haritadaki değerler Dart tarafından çıkarsanır. Elementler haritasında **int** tipinde anahtarlar ile **String** tipindeki veriler ilişkilendirilmektedir. Elementler birbirinden virgülle ayrılır, okumayı kolaylaştırmak adına her element bir satırda yazılabilir. Anahtar – değer çiftleri birbirlerinden **:** ile ayrılır. 6 ve 7 numaralı satırlarda **baskentler** ve **soyGazlar** haritaları tanımlanmıştır. Bu tanımlama metodunda tip çıkarımı yapılabilecek değerler verilmediğinden anahtar ve değerlerin tipleri verilmek zorundadır.

Tanımlı bir haritaya element ekleme işlemi, köşeli parantezler içinde verilen anahtara değer atanarak gerçekleştirilir. Kod 39'da 2 numaralı satırda **elementler** haritasına **4** anahtarı ile **berilyum** değeri eklenmektedir. 3 numaralı satırda ise **String** anahtar-değer çiftleri kabul eden baskentler haritasına **tr** anahtarı ile **Ankara** değeri eklenmektedir. Haritadaki bir değeri elde etmek için köşeli parantezler içinde anahtar ile sorgulama yapılır (Ör: **baskentler["tr"]**). Bu anahtar haritada bulunuyorsa ilişkilendirilmiş değer; aksi halde **null** değeri döndürülür.

```

1 void main() {
2     elementler[4] = "berilyum";
3     baskentler["tr"] = "Ankara";
4     var soygazlar = {
5         2: "helyum",
6         10: "neon",
7         18: "argon",
8         36: "kripton",
9         54: "ksenon",
10        86: "radyum",
11    };
12    elementler.addAll(soygazlar);
13    print(elementler);
14 }
```

Kod 39. Haritalara element eklemek

Anahtar değer çiftleri aynı veri tipinde olan iki harita addAll metodu ile birleştirilebilir. Kod 39'da 4 ve 11 numaralı satırlar arasında **int** anahtarlar ve **String** değerlerden oluşan soygazlar haritası oluşturulmuştur. Bu haritayı aynı anahtar – değer veri tiplerine sahip **elementler** haritasına eklemek için 12 numaralı satırda elementler haritasında **addAll** metodu çalıştırılmıştır. Farklı veri tiplerindeki anahtar – değer çiftlerine sahip haritaların birleştirilmesi bir hata üretecektir (Ör: **The argument type 'Map<String, String>' can't be assigned to the parameter type 'Map<int, String>'**).

Haritalardaki element sayısını bulmak için length özelliği kullanılabilir. Kod 40'ta 2 numaralı satırda **elementler** haritasının **length** özelliği sorgulanarak haritadaki anahtar – değer çifte sayısı elde edilmektedir. Ardından elde edilen değer konsola yazdırılmaktadır.

```
1 void main() {  
2     var elementSayisi = elementler.length;  
3     print("elementler haritasında şuan $elementSayisi adet element ekli");  
4 }
```

Kod 40. Bir haritadaki element sayısını almak

Haritalardaki anahtar – değer çiftleri forEach metodu kullanılarak taranabilir. Bu yöntem her bir element için içerisindeki tek satırlık fonksiyona anahtar ve değer parametreleri gönderir.

```
1 void main() {  
2     elementler.forEach(  
3         (an, deg)=> print("Atom numarası: $an, Element: $deg")  
4     );  
5 }
```

Kod 41. Haritadaki elementleri taramak

Kod 41'de elementler haritası **forEach** metodu ile taranarak her satırda bir elementin atom numarası ve ismi yazılmaktadır. **forEach** metodu sırasıyla anahtar ve değer değişkenlerini içerisindeki anonim fonksiyona gönderir. Bu nedenle her elementin anahtar değeri **an** değişkenine, değeri ise **deg** değişkenine atanarak **print** metoduna gönderilmektedir.

Haritalardan bir elementi silmek için **remove** metodu kullanılır. Remove metodu silinmek istenen elementin anahtarı ile çağırılır. Kod 42'de 2 numaralı satırda elementler haritasındaki 3 anahtarlı element silinmektedir.

```
1 void main() {  
2     elementler.remove(3);  
3     elementler[86] = "radon";  
4     elementler.update(86, "radon");  
5 }
```

Kod 42. Haritadan element silmek ve bir elementi güncellemek

Haritadaki bir elementi güncellemek için atama yapılabılır. Bu amaçla harita referansı köşeli parantezler içinde anahtar değeri ile verilir ve atama gerçekleştirilir. Kod 42'de **elementler**

haritasındaki **86** anahtarlı elementin değeri **radon** olarak güncellenmektedir. Benzer şekilde haritaların **update** metodu da kullanılabilir. Bu metoda parametre olarak anahtar ve değer gönderilir. 4 numaralı satırda **update** metodu kullanılarak **86** anahtarlı elementin değeri **radon** olarak güncellenmektedir.

Haritalardaki anahtarlar ve değerler ayrı listeler şeklinde elde edilebilir. Bu amaçla haritaların **keys** ve **values** özellikleri kullanılır. Kod 43'te 2 numaralı satırda anahtarları ve değerleri **String** tipinde olan elementler haritası oluşturmaktadır. Elementler haritasındaki anahtar değerlerini almak için 3 numaralı satırda haritanın **keys** özelliği kullanılmaktadır. Bu yolla simgeler değişkeninin içeriği haritaya girilen elementlerin simgelerinden oluşacaktır. Benzer şekilde 4 numaralı satırda **values** özelliği ile isimler değişkenine haritadaki elementlerin isimleri atanmaktadır.

```
1 void main() {  
2     var elementler = {"H" : "helyum", "Li": "lityum"};  
3     var simgeler = elementler.keys;  
4     var isimler = elementler.values;  
5     print(simgeler);  
6 }
```

Kod 43. Haritadaki anahtarları ve değerleri almak

**keys** ve **values** özellikleri **Iterable** tipinde listeler oluşturur.

## İnceleme

**Iterable** tipindeki listelerin kendilerine özgü metotları ve özellikleri bulunmaktadır. Bu listelerin kullanımı ile ilgili daha fazla bilgi almak için <https://api.dart.dev/stable/2.14.4/dart-core/Iterable-class.html> adresi ziyaret edilebilir.

## 1.9. Gözle ve Uygula: Akış Yöneticiler

### 1.9.1. if-else

En temel akış yöneticisi **if-else** yapılarıdır. **if** ifadesinde verilen şartın sağlanması durumuna göre tanımlanan ilk blok (**if** bloğu), aksi halde ikinci blok (**else** bloğu) çalıştırılır. Bu yolla yazılan kodların bir şarta bağlı olarak çalıştırılması sağlanmış olur. **if** bloklarının tanımlanması zorunlu, **else** blokları ise isteğe bağlıdır.

```

1 void main() {
2     var sayı = 3;
3     if (sayı % 2 == 0) {
4         print("$sayı sayısı çifttir");
5     } else {
6         print("$sayı sayısı tek");
7     }
8 }
```

Kod 44. if-else yapısı

Kod 44'te verilen bir sayının tek ya da çift olma durumu incelenmektedir. 3 numaralı satırda sayının 2'ye bölümünden kalan 0'a eşit olup olmadığı kontrol edilir. Bu şartın sağlanması halinde sayı çift olacağinden **if** bloğunda sayının çift olduğunu ilişkili bilgilendirme yapılmaktadır. Aksi halde ise sayı tek olacağından **else** bloğunda bu yönde bilgilendirme yapılmaktadır.

Verilen bir koşulun sağlanmaması durumunda else ifadesinin arkasında yeni bir koşul sınamasına imkan sunar.

```

1 void main() {
2     var mA = 80;
3     var mB = 60;
4     var mC = 40;
5     if (mA == 90 || mB == 90 || mC == 90) {
6         print("dik üçgen");
7     } else if (mA == mB && mB == mC) {
8         print("eşkenar üçgen");
9     } else if (mA == mB || mB == mC) {
10        print("ikizkenar üçgen");
11    } else {
12        print("özel bir üçgen değil");
13    }
14 }
```

Kod 45. if - else bloklarının birbirine bağlanması

Kod 45'te verilen üç açı değerine bir üçgenin dik üçgen, eşkenar üçgen ya da ikizkenar üçgen olma durumu incelenmektedir. 5 numaralı satırda açılardan herhangi birinin 90 derece olma durumu kontrol edilerek dik üçgen olup olmadığı incelenmektedir. Bu koşulun sağlanması halinde kullanıcıya üçgenin dik üçgen olduğu mesajı verilerek 14 numaralı satırda atlanacaktır. Bu koşul sağlanmıyorsa 7 numaralı satırda tüm açıların birbirine eş olma durumu kontrol edilmektedir. Bu şart sağlandığında üçgenin eşkenar üçgen olduğu sonucu kullanıcıya bildirilir. Bu koşulun sağlanmaması halinde ise 9 numaralı satırda herhangi iki açının eşitliği kontrol edilmektedir. Bu şart sağlanırsa kullanıcıya üçgenin ikizkenar üçgen olduğu bildirilir. Verilen

tüm şartların sağlanmaması halinde çalışacak **else** bloğunda ise üçgenin özel bir üçgen olmadığı bildirimi yapılmaktadır.

#### Problem Çözme Etkinliği

Kod 45'te verilen kodu, üçgenin iç açılarının toplamının 180 derece olmasını kontrol edecek şekilde geliştireiniz.

#### Problem Çözme Etkinliği

Verilen üç kenar uzunluğundan bir üçgen elde edilip edilemeyeceğini kontrol eden kodu yazınız. Bu konudaki kuralı hatırlamak için <http://www.khanacademy.org.tr/matematik-video.asp?ID=1532> adresindeki videoyu izleyebilirsiniz.

#### 1.9.2. switch-case yapıları

**switch-case** yapıları program akışını bir değişkenin farklı değerlerine göre dallandırmak amacıyla oluşturulmuştur.

```
1 void main() {  
2     var gunNumarasi = 2;  
3     switch (gunNumarasi) {  
4         case 1: print("pazartesi");  
5             break;  
6         case 2: print("sali");  
7             break;  
8         case 3: print("çarşamba");  
9             break;  
10        case 4: print("perşembe");  
11            break;  
12        case 5: print("cuma");  
13            break;  
14        case 6: print("cumartesi");  
15            break;  
16        case 7: print("pazar");  
17            break;  
18        default:  
19            print("gün numarası 1-7 arasında olmalı");  
20            break;  
21    }  
22}
```

Kod 46. switch-case kullanımı

**switch** ifadesinde verilen değişkenin farklı değerlere eşitliğine göre istenen kodlar çalıştırılabilir. Eşitlik dışında bir sorgulama gerçekleştirilemez. Kod 46'da **gunNumarası** değişkeninin aldığı değere göre haftanın hangi gününde olunduğu (pazartesi gününü ilk gün varsayıarak) konsola yazdırılmaktadır. Bir **case** bloğunda birden fazla kod çalıştırılabilir. **case** blokları dolduruldu ise bu blokların sonlarında **break** kullanılması gereklidir. Aksi halde **switch** bir sonraki **case** bloğuna düşebilir (ör: **Switch case may fall through to the next case**). Verilen **case** eşitliklerinin sağlanamaması halinde **default** bloğundaki kod çalıştırılır. Default bloğunun da **break** ile sonlandırılması gerekmektedir.

**switch-case** yapılarında bir **case** için herhangi bir kod yazılmadıysa **break** kullanımı zorunlu değildir. Bir **case** sağlandığında kod bir sonraki **break** ifadesine kadar düşecektir. Kod 47'de bu durum gösterilmektedir. **gun** değişkeninin aldığı değere göre bu günün hafta içi ya da hafta sonundaki bir gün olma durumu konsola yazılmaktadır. 3 numaralı satırda **gun** değişkenine **sali** değeri atanmaktadır. 4 numaralı satırda başlatılan switch bloğunda, 7 numaralı satırda **case** sağlanmaktadır. Fakat bu **case** bloğu boş bırakıldığından kod sonraki satırlara düşmeye başlar. Bir **break** ifadesine rastlanana kadar bloklardaki kodlar işletilecektir. Kod, 10 numaralı satırda **break** ifadesine kadar işler. Bu satırda **sonuc** değişkeninin değeri **hafta içindeki** şeklinde düzenlenmektedir. Aynı durum 11 ve 12 numaralı satırlar için de geçerlidir. Bu satırlardaki **case** ifadelerinin sağlanması durumunda 12 numaralı satırda **sonuc** değişkeninin değeri **hafta sonundaki** olarak belirlenerek **break** ile **switch** sonlanmaktadır. Bu yolla aynı değeri üretecek tüm **case** ifadeleri arka arkaya yazılıp, son case bloğunda gerekli işlem yapılarak daha az kod ile sorun çözülmektedir.

```
1 void main() {
2     var sonuc = "hafta içindeki";
3     var gun = "sali";
4     switch(gun)
5     {
6         case "pazartesi":
7         case "sali":
8         case "çarşamba":
9         case "perşembe":
10        case "cuma": sonuc = "hafta içindeki"; break;
11        case "cumartesi":
12        case "pazar": sonuc = "hafta sonundaki"; break;
13    }
14    print("$gun $sonuc günlerden biridir");
15 }
```

Kod 47. switch-case yapılarında break kullanımı

### 1.9.3. for döngüleri

Dart standart for döngülerini çalıştırılabilir. Bu döngülerde bir döngü kontrol değişkeni tanımlanarak manipüle edilir. Döngünün sonlandırılması bu kontrol değişkeninin bi koşul ifadesi ile sağlanması yoluyla gerçekleştirilir.

```

1 import 'dart:math';
2 void main() {
3     var i=0;
4     double sinus=0;
5     double radyan = 0;
6     for(i=0; i<=90; i++)
7     {
8         radyan = i*pi/180;
9         sinus=sin(radyan);
10        print("$i\u01d5:$sinus");
11    }
12 }
```

*Kod 48. for döngülerinin kullanılması*

Kod 48'de 0 ile 90 arasındaki derecelerin sinüsleri hesaplanarak ekrana yazılmaktadır. Bu amaçla 6 numaralı satırda sıfırdan başlayıp 90 kez donecek bir döngü kurgulanmıştır. Döngünün kontrol değişkeni olarak **i** değişkeni belirlenmiştir. Bu değişken **for** ifadesinde sıfıra eşitlenerek döngü başlatılır. Döngü bir kez çalıştırıldıktan sonra öncelikle döngünün sonlanıp sonlanmayacağı kontrol edilir. Burada **i** değişkeninin değerinin **90**'dan küçük ya da **90**'a eşit olması şartı verilmiştir. Bu şart sağlandığında döngü daha fazla işletilmmez. Sağlanmadığındaysa for ifadesindeki üçüncü bölüm olan manipülasyon uygulanır. Burada **i** değişkeninin değeri 1 arttırılmaktadır. Bu işlem gerçekleştirildikten sonra döngü bir kez daha çalıştırılarak yine kontrol gerçekleştirilecektir.

Hesaplama işlemleri için **dart:math** kütüphanesinin **sin** fonksiyonu kullanılmaktadır. Bu fonksiyon radyan cinsinden verilen değerin sinüsünü çevirmektedir. Döngünün her adımımda 8 numaralı satırda onluk sistemdeki derece değerinin radyanı hesaplanmaktadır. Bu değer **sin** fonksiyonuna gönderilerek onluk sistemdeki derecenin sinüsü hesaplanmaktadır. Son olarak 9 numaralı satırda her adımıda onluk sistemdeki derece ve karşılığı olan sinüs değeri yazdırılmaktadır.

#### *1.9.4. for-in döngüleri*

Bir dizideki tüm elementleri taramak için **for-in** döngüleri kullanılabilir. Bu döngüler kullanılırken herhangi bir kontrol yapılmasına gerek kalmaz. **for-in** döngüleri tanımlanırken dizideki elementin atanacağı bir değişken ismi ve taranacak dizi belirtilir. Döngü bloğu içinde, dizideki aktif elemente tanımlama bloğunda verilen değişken üzerinden erişilir. Diziler üzerinde tarama yapmak için bu döngülerin kullanılması daha pratiktir.

```

1 void main() {
2     var elementler=["hidrojen", "helyum", "lityum"];
3     for(var el in elementler)
4     {
5         print(el+" "+el.length.toString());
6     }
7     elementler.forEach((el)=> print(el + " " + el.length.toString()));
8 }
```

Kod 49. *for-in* döndürülerinin kullanımı

Kod 49'da 2 numaralı satırda taranacak **elementler** listesi oluşturulmaktadır. 3 numaralı satırda bu listenin taranması için **for – in** döngüsü tanımlanmaktadır. **elementler** dizisindeki her bir element sırayla **el** değişkenine atanacaktır. Döngü bloğu içinde her elementin adı ve kaç karakter uzunluğunda olduğu yazdırılmaktadır.

Bu işlem listelerdeki **forEach** metodu ile de yapılabilmektedir. Kod 49'da 7 numaralı satırda aynı işlem **forEach** metodu sayesinde tek satırda gerçekleştirilmektedir. **forEach** metodu her bir elementi **el** parametresi ile içerisindeki tek satırlık fonksiyona göndermektedir. Bu fonksiyonda elementin adı ve karakter uzunluğu ekrana yazdırılmaktadır.

#### 1.9.5. *while* döngüleri

**while** döngüleri verilen bir şartın sağlanması süresince yinelenen döngülerdir. Bu döngülerde for döngülerindeki tanımlama bloğunda bulunan manipülasyon ifadesi yer almadığından, döngü kontrol değişkenlerinin döngü bloğu içinde manipüle edilmesi gerekmektedir. Aksi halde sonsuz döngüler oluşarak uygulamaları kilitleyebilir.

```

1 void main() {
2     var elementler = ["hidrojen", "helyum", "lityum"];
3     while(elementler.isNotEmpty)
4     {
5         var el = elementler.first;
6         print(el);
7         elementler.removeAt(0);
8     }
9 }
```

Kod 50. *while* döngülerinin kullanımı

Kod 50'de 2 numaralı satırda **elementler** listesi tanımlanmaktadır. 3 numaralı satırda **elementler** listesinin boş olma durumunu kontrol eden **isNotEmpty** özelliği kullanılmıştır. Bu özellik listenin boş olması durumunda **false**, dolu olması durumunda ise **true** değerini çevirmektedir. Bu yolla **while** döngüsünün listede element bulunduğu sürece dönmesi sağlanmıştır. Döngü bloğu içinde 5 numaralı satırda **elementler** listesinin ilk elementi **el** değişkenine atanarak 6 numaralı satırda yazdırılmaktadır. 7 numaralı satırda ise listedeki ilk elementi çıkarmaktadır. Bu sayede, verilen şart ifadesinin oluşması sağlanır. 7 numaralı

satırdaki kodun yazılmaması halinde, sürekli listedeki ilk elementi yazan sonsuz bir döngü üretilmiş olur.

## 1.10. Gözle ve Uygula: Fonksiyonlar

Fonksiyonlar sıkılıkla uygulanan bir işlemi gerçekleştirmek adına bir araya getirilmiş kod bloklarıdır. Tanımlı fonksiyonlar kod içinden çağrılarak bu işlevlerin gerçekleşmesi sağlanabilir. Fonksiyonlar parametreler yoluyla dışarıdan veri alabilir ve geriye veri gönderebilir.

```
1  bool ciftMi(int sayı)
2  {
3      return ((sayı % 2 == 0) ? true : false);
4  }
5  void main() {
6      print(ciftMi(4));
7  }
```

Kod 51. Temel bir fonksiyon tanımaması

Kod 51'de 1 ve 4 numaralı satırlar arasında kendisine gönderilen bir değerin çift sayı olup olmadığını kontrol eden basit bir fonksiyon tanımlanmıştır. Fonksiyon tanımamasına, fonksiyonun geri döndüreceği verinin tipinin belirtilmesi ile başlanır. Tip tanımlaması zorunlu değilse de bu tanımlamanın yapılması önerilen bir yaklaşımdır. Fonksiyon geriye değer döndürmeyecekse **void** ile tanımlanması gerekmektedir. Tip tanımlamasının ardından fonksiyonun adı verilir. Fonksiyon adından sonra parantezler içinde fonksiyona gönderilecek parametreler sıralanır. Burada tanımlanan parametreler fonksiyon gövdesinde tanımlı değişkenler olarak kullanılır. Fonksiyonun başlangıcı ve bitisi küme parantezleri ile işaretlenir. Fonksiyon **void** dışında bir veri tipi ile tanımlandıysa (ör: **int**, **bool**) fonksiyon gövdesinde **return** ifadesi ile bu tipe uygun bir değerin döndürülmesi zorunludur. Aksi halde bir hata üretilir (ör: **A non-null value must be returned since the return type 'bool' doesn't allow null**). 3 numaralı satırad hesaplanan değere göre **ciftMi** fonksiyonu **Boolean** tipinde **true** ya da **false** değerlerinden birini döndürmektedir.

### Dikkat

Fonksiyonlar hangi tipte veri döndürecekleri belirtilmese de çalışabilirler. Yine de fonksiyonların döndürecekleri değerlerin veri tiplerinin belirtilerek tanımlanması önerilen bir yaklaşımdır.

Fonksiyonlar verilen isimleri ve tanımlanan parametreleri ile çağrırlar. Kod 51'de 6 numaralı satırda **ciftMi** fonksiyonu **4** parametresi ile çağrılmaktadır. Geriye döndürülen **Boolean** tipindeki değer ise **print** fonksiyonuna gönderilmektedir.

Kod 52'de Kod 51'de tanımlanan **ciftMi** fonksiyonunun tek satırlık tanımlanması gerçekleştirilmektedir. Bir fonksiyon içinde tek bir işlem gerçekleştiriliyorsa kodu kısaltmak adına ok sentaksı tercih edilebilir.

```
1  bool ciftMi(sayı) => (sayı % 2 == 0) ? true : false;
```

Kod 52. Tek ifadeli fonksiyonlarda kısayol yazımı

Fonksiyonların parametre alması ya da değer döndürmeleri zorunlu değildir. Kod 53'te 5 ve 7 numaralı satırlar arasında tanımlanan **sayilarisay** fonksiyonu herhangi bir parametre almaz. Dönüş veri tipi de **void** olarak belirtildiğinden geriye bir değer döndürmez. Bu nedenle fonksiyonun gövdesinde bir **return** ifadesi bulunmamaktadır. **sayilarisay** fonksiyonu çağırıldığında **sayilar** listesindeki element sayısını konsola yazarak işlevini bitirmektedir.

```
1  var sayilar = [1, 2, 3, 5];
2  void main() {
3      sayilarisay();
4  }
5  void sayilarisay() {
6      print(sayilar.length.toString());
7  }
```

Kod 53. Fonksiyona pozisyonel parametre gönderimi

Fonksiyonlar, pozisyonel ve isimli olmak üzere iki tür parametre kabul eder. Pozisyonel parametre kullanımında parametreler fonksiyon tanımlamasındaki sırayla fonksiyona gönderilir. İsimli parametre kullanımında ise değerler parametre isimleriyle birlikte gönderilir.

```
1  void main() {
2      print(alanHesapla(3, 4));
3      print(alanHesapla2(yukseklik: 4, taban: 5));
4  }
5  double alanHesapla(taban, yukseklik) {
6      return (taban * yukseklik / 2);
7  }
8  double alanHesapla2({taban, yukseklik}) {
9      return (taban * yukseklik / 2);
10 }
```

Kod 54. İsimli ve pozisyonel parametre kullanımı

Kod 54'te 5 ve 7 numaralı satırlar arasında tanımlanan **alanHesapla** fonksiyonu pozisyonel parametreler almaktadır. Bu fonksiyon 2 numaralı satırda sırasıyla **3** ve **4** parametreleri ile çağırılmaktadır. Bu durumda **alanHesapla** fonksiyonuna **taban** değeri olarak **3**, **yukseklik** değeri olarsa **4** gönderilmektedir. 8 ve 10 numaralı satırlar arasında tanımlanan **alanHesapla2** fonksiyonu ise isimli parametrelerle tanımlanmıştır. Küme parantezleri içine alınan parametreler isimli parametrelerdir. İsimli parametre kullanımında parametrelerin tanımlanma sırası gözetilmez. İsimli parametre kullanan fonksiyonlar çağrılrken parametre değerleri parametre isimlerinden **:** ile ayrılarak gönderilir. 3 numaralı satırda **alanHesapla2** fonksiyonu **yukseklik** için **4**, **taban** için **5** değeri ile çağırılmaktadır. Bir fonksiyon birkaç

parametre ile tanımlanmışsa, pozisyonel parametrelerin sıralamasının unutulması ihtimaline karşı isimli parametrelerle tanımlanması önerilen bir yaklaşımındır.

İsimli parametreler varsayılan değerlerle tanımlanabilir. Kod 55'te **alanHesapla2** fonksiyonu taban ve yükseklik değerleri varsayılan değerler ile tanımlanmıştır. Bu fonksiyon hiçbir parametre verilmeden çağrırlabilir. Bu çağrıda taban ve yükseklik parametreleri varsayılan değerleriyle kullanılır. Benzer şekilde parametrelerin yalnızca birine değer vererek de fonksiyon (ör: **alanHesapla2(yükseklik: 4);**) çağrırlabilir.

```
1 print(alanHesapla2());
2 double alanHesapla2({double taban=3, double yukseklik=5}) {
3     return (taban * yukseklik / 2);
4 }
```

Kod 55. İsimli parametrelerde varsayılan değer kullanımı

Genellikle isimli fonksiyonlar kullanılsa da Dart anonim (isimsiz) fonksiyonlara da destek verir. Anonim fonksiyonlar genellikle bir değişkene atanır. Örneğin liste değişkenlerinden **forEach** metodu ile çekilen tekil değişkenlere anonim fonksiyonlar atanabilir.

```
1 void main() {
2     var sira=1;
3     var elementler = ["hidrojen", "helyum"];
4     elementler.forEach((elementim){
5         print("$elementim elementi $sira. sıradadır");
6         sira++;
7     });
8 }
```

Kod 56. Anonim fonksiyon kullanımı

Kod 56'da tanımlanan elementler listesine uygulanan **forEach** metodu ile listedeki elementler teker teker taranmaktadır. **forEach** metodu parametre olarak bir anonim fonksiyon alarak bu fonksiyona her seferinde bir element göndermektedir.

### Dikkat

Daha önceki Kod örneklerinde (Ör: Kod 33) tek satırlık fonksiyonlar kullanıldığından => sentaksı kullanılmıştı. Kod 56'daki anonim fonksiyonda birden fazla yönerge kullanıldığından => sentaksı yerine küme parantezleri kullanılmıştır.

## 1.11.Gözle ve Uygula: Etki Alanları

Fonksiyonlar ve döngü blokları gibi küme parantezleri ile kapanan kodlar etki alanları (**lexical scope**) oluşturur. Değişkenler tanımlandıkları etki alanları ve bu alanın kapsadığı alt etki alanlarında geçerlidir. Bir etki alanında tanımlı bulunmayan değişkenlere erişilmeye çalışıldığında hata üretilir (ör: **Error: Getter not found: degisken**).

Döngüler etki alanları oluşturduğundan bu alanlarda tanımlanan değişkenler, alan dışında erişilemez.

```
1 void main() {  
2     for (var i = 0; i < 5; i++) {  
3         print("$i");  
4     }  
5     print("$i");  
6 }
```

Kod 57. Döngü kontrol değişkenlerine erişememe hatası

Kod 57'de 2 numaralı satırda tanımlanan **for** döngüsünde **i** kontrol değişkeni tanımlanmaktadır. Bu değişkenin erişilebilir alanı for döngüsü gövdesiyle sınırlıdır. Yukarıdaki kod çalıştırıldığında 3 numaralı satirdaki **print** fonksiyonu sayesinde **i** değişkeninin aldığı değerler konsola yazılacaktır. Fakat 5 numaralı satirdaki başvuru hata (Ör: **Getter not found: 'i'**) üretecektir.

Benzer bir hata farklı fonksiyonlarda tanımlanan değişkenlere erişmeye çalışırken de gözlenebilir.

```
1 void main() {  
2     var fiyat = 100;  
3     var kdv = 18;  
4     var sonuc = ekle(fiyat, kdv);  
5     print("ekle fonksiyonundan dönen sonuc: $sonuc");  
6     print("fiyat değişkeninin değeri $fiyat");  
7     //print(toplam);  
8 }  
9 int ekle(sayı1, sayı2) {  
10    //print(fiyat);  
11    sayı1 += 10;  
12    var toplam = sayı1 + sayı2;  
13    print("ekle fonksiyonunda sayı1: $sayı1");  
14    return (toplam);  
15 }
```

Kod 58. Fonksiyonların alt alanları

Kod 58'de 9 ve 15 numaralı satırlar arasında **ekle** fonksiyonu tanımlanmıştır. Bu fonksiyon kendisine parametre olarak gönderilen iki değeri toplayarak, sonucu **int** tipinde geri çevirmektedir. **main** fonksiyonu ise Dart uygulamalarının başlangıç noktasıdır. 2 ve 3 numaralı satırlarda tanımlanan **fiyat** ve **kdv** değişkenlerindeki değerler 4 numaralı satırda **ekle** fonksiyonuna parametre olarak gönderilmektedir. Bu çağrı ile **ekle** fonksiyonu çalıştırılarak

kendisine gönderilen parametreleri **sayı1** ve **sayı2** referansları yoluyla toplayıp geriye değer döndürmektedir. Geriye çevrilen değer ise 5 numaralı satırda konsola yazılmaktadır.

Bu iki fonksiyon özgün etki alanları oluşturduğudan, **main** fonksiyonunda tanımlı **fiyat** değişkeni **ekle** fonksiyonunda erişilemez. Bu nedenle 9 numaralı satırda hata üretir. Benzer şekilde **ekle** fonksiyonunda tanımlı **toplam** değişkeni de **main** fonksiyonunda erişilebilir değildir. 6 numaralı satırda hata üretecektir. Hata kodu almamak adına bu satırlar yorum satırına çevrilmiştir. İki farklı fonksiyonun etki alanı arasında değer gönderimi için parametreler ve **return** mekanizmaları kullanılır.

Dikkat edilmesi gereken önemli noktalardan biri de parametre kullanıldığında değişkenlerin değil, değişkenlerdeki değerlerin gönderildiğidir. Dart referans yoluyla parametre gönderimini desteklemez. Bu nedenle 11 numaralı satırda **sayı1** değişkenine yapılan ekleme, **main** fonksiyonundaki **fiyat** değişkenini etkilemez. 13 numaralı satırda **sayı1** değişkeninin değeri konsola yazılmaktadır. 6 numaralı satırda ise **fiyat** değişkeninin değerinin değişmediği gösterilmektedir.

Bir alt etki alanı üzerindeki etki alanındaki değişkenlere erişebilir.

```
1 void main() {  
2     var fiyat = 100;  
3     var kdv = 18;  
4     var toplam = 0;  
5     int hesapla() {  
6         var sonuc = fiyat + kdv;  
7         var otv = 6;  
8         toplam = fiyat + kdv + otv;  
9         return (sonuc);  
10    }  
11    print("fonksiyondan dönen değer: "+hesapla().toString());  
12    print("toplam değişkeninin değeri: $toplam");  
13    //print(sonuc);  
14 }
```

Kod 59. Alt etki alanından, üst etki alanındaki değişkenlere erişim

Kod 59'da 5 ve 9 numaralı satırlar arasında tanımlanan hesapla fonksiyonu herhangi bir parametre almadan **int** tipinde değer çevirmektedir. Bu fonksiyon **main** fonksiyonu etki alanı içinde tanımlandığından, **main** fonksiyonundaki değişkenlere erişimi vardır. Bu sayede 6 numaralı satırda tanımlanan **sonuc** değişkeninin değeri, üst etki alanındaki **fiyat** ve **kdv** değişkenlerine erişilerek hesaplanabilir. Benzer şekilde, 8 numaralı satırda üst etki alanında tanımlanan **toplam** değişkeninin içeriği de **hesapla** fonksiyonu ve **main** fonksiyonu etki alanlarında tanımlanan değişkenlerin toplanması yoluyla hesaplanabilmektedir. Görüldüğü üzere, **hesapla** fonksiyonu üzerindeki etki alanında yer alan değişkenlere erişebilmekte, hatta içeriklerini düzenleyebilmektedir. Fakat tersi geçerli değildir. **main** fonksiyonu altındaki etki

alanında tanımlı **sonuc** değişkenine erişemez. Bu nedenle 13 numaralı satırındaki **print** çağrıları hata üretir.

## 1.12.Dart ile Nesneye Yönelik Programlama

Dart nesne yönelimli bir programlama dili olduğundan, bu programlama yaklaşımının sistematigi konusunda bilgi sahibi olunması gerekmektedir. Şu ana kadar yazılan kodlarda pek çok kez nesneye yönelik programmanın mekanizmaları kullanılmıştır. Bu bölümde kısaca bu mekanizmalardan bahsedilecektir.

### 1.12.1. Nesneler

Nesneler birbirleri ile ilişkili veri yapıları ve kod bloklarını içeren paketlerdir. Nesneye yönelik programlama yaklaşımında çoğunlukla nesnelerin iç işleyişleri ile ilgilenilmez. Nesnelerin dışarıya sundukları arayüzler (Ör: özellikler ve metotlar) kullanılarak gerekli işlemler uygulanır.

```
1 void main() {  
2     var sayilar = [1, 2, 3, 5, 8, 13];  
3     sayilar.add(21);  
4     var elemanSayisi = sayilar.length;  
5 }
```

Kod 60. Nesne üretme, nesnelerin özelliklerine ve metotlarını kullanma

Kod 60'ta 2 numaralı satırda bir liste nesnesi oluşturularak bu nesne **sayilar** isimli değişkene aktarılmaktadır. 3 numaralı satırda ise sayılar değişkeninde tutulan Liste tipindeki nesnenin **add** metodu kullanılmaktadır. Bu metot listenin sonuna bir element eklemek için kullanılır. 4 numaralı satıda ise yine **sayilar** değişkeninde tutulan Liste tipindeki değişkenin **length** özelliğine erişilmektedir. Bu özellik ise listedeki element sayısını vermektedir. Özellikler nesne üzerinde tanımlanmış değişkenlerdir. Nesnelerin özelliklerine, nesne referansının ardından . işlevi ile birleştirilmiş özellik adı ifadesi ile (Ör: **sayilar.length**) erişilir. Metotlar ise nesne üzerinde tanımlanmış fonksiyonlardır. Özelliklerde olduğu gibi bu fonksiyonlar nesne referansının arkasına eklenen . işlevinden sonra isimleri (Ör: **sayilar.add(2)**) ile çağrırlar. Fakat özünde fonksiyon olduklarından metotlar parantezler ve gerekli hallerde parantezler arasındaki parametreler belirtilerek çağrırlar.

### 1.12.2. Sınıflar

Sınıflar nesne taslaklarıdır. Nesnelerde kullanılacak özellikler ve metotlar sınıflar içinde kodlanır. Nesneler sınıflardan üretilir. Kod 61'de bir Hayvan sınıfı tanımlanmıştır. Bu kodlar **hayvan.dart** dosyası içinde saklanmaktadır. Bu sınıf hayvanın yaşı ve cinsiyeti ile ilgili iki özelliğin yanında, hayvanın bilgilerini raporlayan ve hayvanı besleyen iki metot sağlamaktadır. Son olarak hayvanın yaşından düzenlenmesini (**setter**) sağlayan ve yaşın çevrilmesini (**getter**) sağlayan iki metot daha bulunmaktadır.

Hayvan sınıfında kullanılmak üzere iki adet cinsiyet tanımlaması yapılmıştır. 1 numaralı satırda **enum** tipinde cinsiyetler tanımlaması yapılmıştır. **enum** veri tipleri ön tanımlı sabitler için kullanılmaktadır. Bu sayede hayvanın cinsiyetini belirlemek için **cinsiyetler.disi** ya da **cinsiyetler.erkek** tanımlamaları kullanılabilecektir. Bu uygulama zorunlu olmamakla

birlikte kodda bütünlük ve kullanım kolaylığı sağlamaktadır. **enum** tipinin sınıf dışında tanımlandığına dikkat ediniz.

### Dikkat

Kodlarda standartlaşmayı sağlamak için enum tipindeki değişkenler sıkılıkla kullanılır. İleride yazılacak kodlarda, Dart dilinde de simgeler (**Icons.home**) ya da renkleri (**Colors.red**) ifade etmek için enum değerlerin kullanıldığı görülecektir.

2 ile 19. satırlar arasında Hayvan sınıfı tanımlanmıştır. Hayvan sınıfından üretilen nesnelerin özellikleri 3 ve 4 numaralı satırlarda tanımlanmıştır. Her nesnenin yaş ve cinsiyeti belirlenebilecektir. Yaş bilgisi için **int** tipinde **\_yas**, cinsiyet bilgisi içinse **cinsiyetler** tipinde **cinsiyet** özellikleri (değişkenleri tanımlanmıştır). **\_yas** değişkeninin önündeki **\_** karakteri bu değişkenin erişimini sınırlamak için kullanılır. Bu tanımlama sayesinde **hayvan** kütüphanesi dışında (**hayvan.dart** dosyası dışında) **\_yas** değişkeni erişilebilir olmayacağından bir güvenlik önlemi olarak düşünülebilir. Öte yandan bu değişkeni düzenleyen ve değişkeni dışarı çeviren mettotlar kullanılarak sınıf içindeki mekanizmaların otomatikleşmesi sağlanabilecektir. **\_yas** özelliğinin düzenlenmesi için 7-9 numaralı satırlar arasında **yasBelirle** metodu tanımlanmıştır. Bu tür metotlara düzenleyici (**setter**) metotlar denir. Bu kodda yaşı düzenleme dışında bir işlev olmasa da, daha karmaşık sınıflarda yaşı belirlenmesine bağlı olarak pek çok farklı özellik (ör: günlük enerji tüketimi) ve metodun (ör: dış görünüşün güncellenmesi) çalıştırılması sağlanabilirdi. 10-12 numaralı satırlar arasında ise **\_yas** değişkeninde tutulan değeri çeviren (**getter**) **yas0gren** metodu tanımlanmıştır. Bu metotta nesnede tutulan hayvanın yaşıının çeşitli zaman dilimlerine göre (ör: günlük, aylık, yıllık) çevrilmesi sağlanabilirdi.

Tıpkı değişkenlerde olduğu gibi metotlar da kütüphane dışına kapatılabilir. Örneğin hayvanın günlük enerji tüketimini gerçekleştiren (Ör: **\_tuket**) bir metodun dışarıya açılmasına gerek yoktur. Bu metot sınıfın kendi iç işleyişile ilgili olduğundan dışarıdan erişilemez olacak şekilde tanımlanabilirdi.

Hayvan sınıfının iki adet yapıcı metodu (**constructor**) bulunmaktadır. Yeni bir hayvan nesnesi oluşturulurken bu iki metottan biri kullanılabilir. 5 numaralı satırdaki **Hayvan** yapıcı metodu sınıfın standart yapıcı metodu olarak düşünülebilir. Bu metot sırasıyla **yas** ve **cinsiyet** değişkenleri için iki pozisyonel parametre gerektirmektedir. Tanımlama satırında verilen **this.\_yas** kısayol sentaksı sayesinde metodun gövdesinin oluşturulması ve atama yapılmasına gerek yoktur. Kod 62'de 3 numaralı satırda bu yapıcı metot çağrılmaktadır. Bu satırda 2 yaşında dişi bir kuş nesnesi üretilerek **kus** isimli değişkene atanmaktadır.

```

1 enum cinsiyetler { disi, erkek }
2 class Hayvan {
3     int _yas;
4     cinsiyetler cinsiyet;
5     Hayvan(this._yas, this.cinsiyet);
6     Hayvan.isimli({required yas, this.cinsiyet = cinsiyetler.erkek}) : _yas
= yas;
7     set yasBelirle(int yas) {
8         _yas = yas;
9     }
10    int get yas0gren {
11        return (_yas);
12    }
13    besle() {
14        print("yedim");
15    }
16    raporla() {
17        print("$_yas yaşındayım, cinsiyetim: $cinsiyet");
18    }
19 }

```

Kod 61. Sınıf tanımlama, sınıfı metot ve özellik ekleme (hayvan.dart)

**Hayvan** sınıfının ikinci yapıcı metodu **Hayvan.isimli** metodudur. Bu metotta isimli parametreler kullanılmıştır. **Hayvan** sınıfı sadece 2 özellik kullandığında 20 özellikli bir sınıfta pozisyonel parametrelerin kullanılması oldukça zor olacaktır. Bunun gibi durumlarda isimli parametreler kullanılan yapıcı metotlar çok daha işlevseldir. Metodun tanımlama satırında **yas** isimli değişkenin önünde **required** belirteci kullanılmıştır. Bunun anlamı bu isimli parametreye değer verilmesinin zorunlu olduğunu söyler. Bu yolla hiçbir **Hayvan** nesnesinin **yas** bilgisi olmadan üretilmemesi garanti altına alınır. Bir önceki yapıcı metotta pozisyonel parametreler kullanıldığından bu belirtme gereklidir. Cinsiyet parametresinin varsayılan bir değerle tanımlandığı görülebilir. Bu parametreye değer atanmaması halinde, bu metot kullanılarak üretilen tüm **Hayvan** nesneleri **erkek** olacak şekilde düzenlenmiştir. Parametreye değer atanırsa verilen varsayılan değer görmezden gelinecektir. **isimli** yapıcı metodunun arkasında : işaretçi (**initializer**) kullanılmıştır. Bu işaret arkasında gerekli değişkenlere değer ataması yapılmamıştır. Bu alanda değişken değer çiftlerinin arasına virgül konması gereklidir. Bu düzenlemenin yapılmaması halinde **yas** değişkenine değer atanmadığını belirten bir hata mesajı alınacaktır.

```

1 import './hayvan.dart';
2 void main() {
3     var kus = Hayvan(2, cinsiyetler.disi);
4     kus.besle(); // yedim
5     kus.raporla(); // 2 yaşındayım, cinsiyetim: cinsiyetler.disi
6     var kedi = Hayvan.isimli(yas: 5);
7     //print(kedi._yas);
8     kedi.yasBelirle = 3;
9     print(kedi.cinsiyet); // cinsiyetler.erkek
10    print(kedi.yas0gren); // 3
11    kedi.raporla(); // 3 yaşındayım, cinsiyetim: cinsiyetler.erkek
12 }

```

Kod 62. Hazırlanan sınıfın nesne üretme ve metodlarını kullanma (main.dart)

Kod 62'de **main.dart** dosyasının kodları görülmektedir. Bu dosyada **Hayvan** sınıfının kullanılabilmesi için öncelikle bu kütüphanenin **import** edilmesi gereklidir. 1 numaralı satırda bu işlem gerçekleştirilmektedir. 6 numaralı satırda **Hayvan.isimli** yapıcı metodu kullanılarak yeni bir **Hayvan** nesnesi üretilmektedir. Bu üretimde yalnızca **yas** değişkenine değer atandığına dikkat ediniz. Bir **cinsiyet** belirtilmediğinden, üretilen kedinin cinsiyeti varsayılan değer olan **cinsiyetler.erkek** olarak belirlenecektir. 8 numaralı satırda **kedi** değişkenindeki **Hayvan** nesnesinin **yasBelirle** (setter) metodu kullanılarak nesnenin yaşı **3** olarak düzenlenmektedir. 10 numaralı satırda ise nesnedeki **\_yas** değeri **yas0gren** metodu (getter) ile elde edilmektedir. Yorum satırı haline getirilmiş 7 numaralı satırdaki **kedi.\_yas** çağrıları bir hata mesajı üretecektir. Bunun yerine **yas0gren** metodunun kullanılması gereklidir. **cinsiyet** değişkeni dışarıya açık olduğundan bu değişkenin değeri 9 numaralı satırdaki **kedi.cinsiyet** çağrıları ile elde edilebilir. 4 numaralı satırda **Hayvan** sınıfının **besle()** metodu, 5 ve 11 numaralı satırlarda ise **raporla()** metodu çağrılmaktadır. Her iki metodun ekran çıktıları ilgili satırlarda yorum halinde verilmiştir.

### 1.12.3. Kalıtım (Inheritance)

Nesneye yönelik programlama yaklaşımının en önemli mekanizmalarından biri kalıtımıdır. Kalıtım bir sınıf için yazılan kodların yeni sınıflarda kullanılabilir olmasını sağlar. Bu amaçla tanımlama satırında **extends** anahtar sözcüğü ile yeni sınıfın hangi sınıfın kodlarını miras alacağı belirtilir.

Kod 63'te Hayvan sınıfının özellikleri ve metodlarını miras alan **Kedi** sınıfının kodları sunulmuştur. Kediler de özünde hayvan olduklarından bu sınıfın kodlarının kullanılmasında mantıken bir sorun yaşanmayacaktır. Öncelikle **Hayvan** sınıfının kodlarının erişilebilir olması için 1 numaralı satırda **hayvan.dart** dosyası **import** edilmiştir. 2 numaralı satırda kedi nesnesinin cinsini belirmek amacıyla kullanılacak olan **enum** tipindeki **cins** listesi tanımlanmıştır. Üretilerek her **Kedi** nesnesi bu üç cinsten birinde olacaktır. **Kedi** sınıfının **Hayvan** sınıfındaki kodları miras alacağını belirtmek için 3 numaralı satırda **extends Hayvan** ifadesi kullanılmıştır. 4 numaralı satırda **Kedi** nesnesinin cinsini belirtmek amacıyla kullanılacak **cinsi** özelliği tanımlanmıştır. Bu özelliğin değeri 5 numaralı satırdaki pozisyonel

değişkenli yapıçı metotta atanmaktadır. Bu sınıfta tanımlı olmasa dahi, **cinsiyet** ve **yas** bilgilerinin tanımlanması gerekmektedir. Bu amaçla **initialize** alanında **super()** çağrıları ile kalıtım mekanizması uygulanan üst sınıf olan **Hayvan** sınıfının yapıçı metodunu çağrılmıştır. Bu metoda, yapıçı metoda gelen 2. ve 3. pozisyonel parametreler aktarılmıştır.

```
1 import './hayvan.dart';
2 enum cins { van_kedisi, ankara_kedisi, tekir }
3 class Kedi extends Hayvan {
4     cins cinsi;
5     Kedi(this.cinsi, int yas, cinsiyetler cinsiyet) :super(yas, cinsiyet);
6     sesCikar() {
7         print("miyav");
8     }
9     @override
10    raporla() {
11        super.raporla();
12        print("cinsim: $cinsi");
13    }
14 }
```

Kod 63. Kalıtım mekanizmasını kullanma (kedi.dart)

6-8 numaralı satırlar arasında, **Hayvan** sınıfında bulunmayan **sesCikar()** metodunu tanımlanmıştır. Bu metod **Kedi** sınıfına özgü ses çıkarılmasını sağlayacaktır. 9-13 numaralı satırlar arasında ise **Hayvan** sınıfında da yer alan **raporla** metodunun yeni bir versiyonu oluşturulmuştur. Bu metodun **Hayvan** sınıfındaki metodun üzerine yazdığını belirtmek amacıyla **@override** belirtimi yapılmıştır. Bu belirtme zorunlu olmama da önerilen bir uygulamadır. Raporla metodu içinde yine üst sınıfındaki raporla metodunu **super.raporla()** çağrıları ile çalıştırılmıştır. Ardından **Kedi** sınıfına özgü bilgiler rapora eklenmiştir.

```
1 import './kedi.dart';
2 void main() {
3     var kedicim = Kedi(cins.ankara_kedisi, 6, cinsiyetler.erkek);
4     kedicim.raporla(); // 6 yaşındayım, cinsiyetim: cinsiyetler.erkek
5     // cinsim: cins.ankara_kedisi
6     kedicim.sesCikar(); // miyav
7 }
```

Kod 64. Kedi sınıfının kullanımı (main.dart)

Kod 64'te **main.dart** dosyasında Kedi sınıfının kullanımı gösterilmektedir. 1 numaralı satırda **kedi.dart** dosyası **import** edilmektedir. 3 numaralı satırda yeni bir **Kedi** nesnesi üretilerek **kedicim** değişkenine atanmaktadır. 4 ve 6 numaralı satırlarda ise **raporla()** ve **sesCikar()** metodlarının kullanımı gösterilmektedir. Her iki metodun çıktıları yorum satırı halinde koda eklenmiştir.

## 2. TASARLA ve ÜRET

Bir hayvanat bahçesi oyunu ürettiğinizi varsayıarak, en az 3 farklı hayvan sınıfı (Ör: memeliler, sürüngenler, kuşlar) ve her sınıfa ait bir tür (Ör: ev kedisi, kaplumbağa, papağan) içeren sınıf hiyerarşisini tasarlaymentınız. Bu hiyerarşide kalıtım mekanizmasını kullanınız.

## 3. DEĞERLENDİR

Öğrencinin özdeğerlendirme yapabilmesi için Öz Değerlendirme Formu (<https://forms.gle/kts381jqHWcCah2j6>) uygulanır.

### Proje Hazırlama

Dersin sonunda üretilecek öğrenci projesi fikirlerini sınıfla birlikte tartışınız. Öğrencilerin grup çalışmasına verdikleri katkılara yönelik özdeğerlendirme yapabilmeleri için Grup Çalışması Öz Değerlendirme Formu'nu (<https://forms.gle/FWk3AhU2sFX5heVK8>) uygulayınız.

Bir sonraki haftaya kadar öğrencilerden projede kullanacakları veri yapılarını oluşturmalarını isteyiniz.

## 4. İLAVE ETKİNLİK

Bir sayının asal olmadığını kontrol eden Dart uygulamasını hazırlayınız.

```
1 void main() {  
2     var sayı = 7;  
3     var sayac = 2;  
4     var bolundu = false;  
5     while (sayac < sqrt(sayı)) {  
6         if (sayı % sayac == 0) {  
7             bolundu = true;  
8         }  
9         sayac++;  
10    }  
11    if (bolundu) {  
12        print(sayı.toString() + " sayısı asal değildir");  
13    } else {  
14        print(sayı.toString() + " sayısı asaldır");  
15    }  
16}
```

Kod 65. Verilen sayının asal olmadığını inceleyen kod

## **5. KAYNAKÇA**

Google. (t.y.). Dart Language Tour. <https://dart.dev/guides/language/language-tour> adresinden 04.11.2021 tarihinde erişilmiştir.

## 3. Hafta: Bileşenler

---

### Ön Bilgi:

- Temel Dart bilgisi
- Temel internet kullanım bilgisi

### Haftanın Kazanımları:

- Bileşen (Widget) kavramını açıklayabilir
- Bileşen ağaçları (Widget Tree) kavramını açıklayabilir
- Flutter uygulamalarına simge ekleyebilir
- Flutter uygulamalarına metin ekleyebilir
- Flutter uygulamalarındaki metinlerin stillerini düzenleyebilir
- Flutter uygulamalarında tema kullanabilir
- Flutter uygulamalarına resim ekleyebilir

### Haftanın Amacı:

Bu haftanın amacı, Flutter uygulamalarına içerik eklemek ve içeriklerin sunum biçimlerini düzenleyebilmektir.

### Kullanılacak Malzemeler:

Visual Studio Code, Chrome internet tarayıcısı.

### Haftanın İşlenisi:

**Gözle:** Yazı, simge ve resim ekleme bileşenlerini incele

**Uygula:** Uygulamalara yazı, simge ve resim ekle

**Tasarla:** Uygulama için bir tema tasarla

**Üret:** Tasarladığın tema ile bir arayüz üret

**Değerlendir:** Tasarladığın arayüzü değerlendir

# 1. GÖZLE VE UYGULA

## 1.1. Gözle: Bileşen (Widget) Kavramı

Flutter uygulamalarında ekranda gösterilen her nesne bir bileşendir (Widget). Bileşenler kullanıcı ara yüzü oluşturmak için kullanılan tuqlalar gibi düşünülebilir. Uygulamadaki menü barları, yazılar, resimler, listeler, animasyonlar ve boşlukların tümü bileşenlerdir. Flutter uygulamaları birçok bileşenin bir araya getirilmesi yoluyla oluşturulur.

### Bileşen Kütüphanesi

Flutter uygulamalarının arayüzleri bileşenler üzerine kurgulandığından, Flutter ciddi bir bileşen kütüphanesi sunmaktadır. Kullanılabilecek bileşenleri incelemek için <https://docs.flutter.dev/development/ui/widgets> adresini ziyaret edebilirsiniz.

Bileşenler, bir bileşen ağacı (Widget Tree) içinde organize edilerek Flutter uygulamalarının ara yüzleri oluşturulur. Bu ağacın kökünde genellikle kullanılmak tasarım diline özgü bir kapsayıcı bir bileşen yer alır. Daha önceki bölümlerde Flutter'in farklı işletim sistemlerine özgü çıktılar üretebildiğinden bahsedilmiştir. Her bir işletim sisteminin kendine özgü font, görünüm, fizik ve etkileşim karakteristikleri bulunmaktadır. Bu karakteristikler tasarım dili olarak ele alınmaktadır. Flutter ile Android tasarım diline özgü içerikler üretilmek istenirse bileşen ağacının köküne **MaterialApp** bileşeni eklenir. iOS işletim sistemine özgü içerik üretilmek istediğiseyse bileşen ağacının kökünde **CupertinoApp** bileşeni kullanılır.

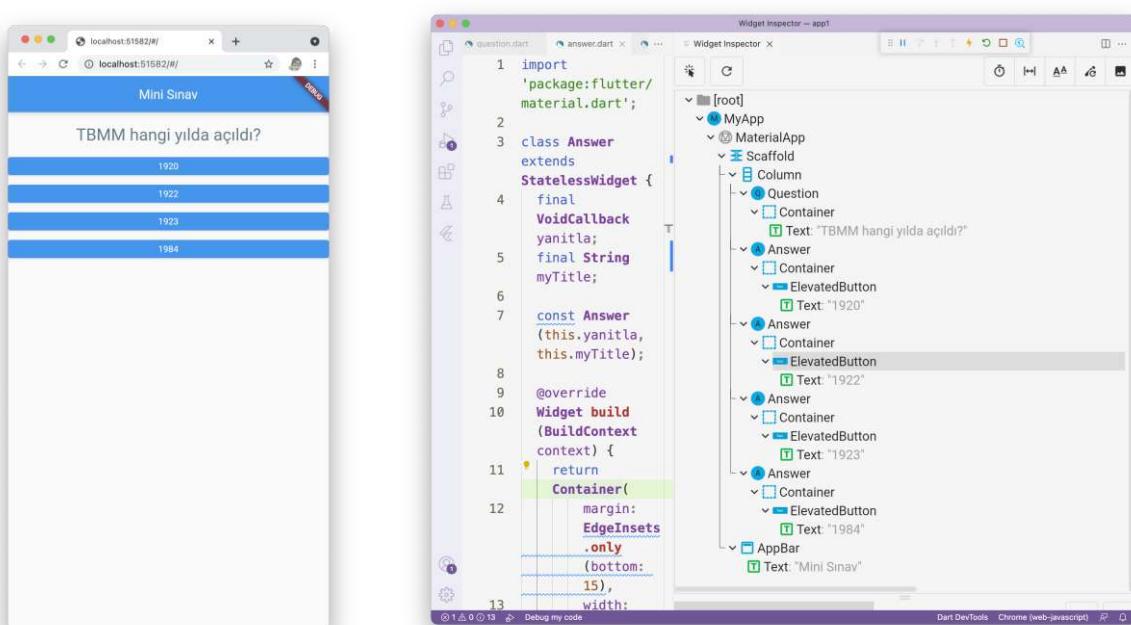
### Cupertino Bileşenleri

Bu kitaptaki örneklerde Google firması tarafından üretilen ve Android işletim sisteminde kullanılan Materyal Tasarım ilkelerine özgü bileşenler kullanılacaktır. Fakat Flutter iOS işletim sistemine özgü bileşenler ve tasarım unsurları da sunmaktadır. Bu bileşenleri incelemek için <https://docs.flutter.dev/development/ui/widgets/cupertino> adresini ziyaret edebilirsiniz.

VS Code programında bir uygulamanın bileşen ağacı görüntülenebilir. Bu amaçla Widget Inspector paneli incelenebilir. Widget Inspector, Flutter kurulumu ile Dart DevTools paketi altında VS Code uygulamasına eklenir. Flutter uygulamaları VS Code uygulamasından çalıştırıldığında bu pencere otomatik olarak açılır. Açılmaması halinde, VS Code uygulaması arayüzünde alt sağda görünen Dart DevTools düğmesine basılarak açılan komutlar penceresinde **Open Widget Inspector** komutu seçilebilir. Benzer şekilde komutlar penceresini **CTRL+SHIFT+P** (MacOS işletim sisteminde **COMMAND+SHIFT+P**) tuş kombinasyonu ile açarak **Flutter: Inspect Widget** komutu ile ile Widget Inspector paneline ulaşılabilir.

Ekran Görüntüsü 7'de basit bir mini sınav uygulaması görülmektedir. Bu uygulamada bir başlık çubuğu (**AppBar**), bir soru (**Text**) ve dört seçenek (**Elevated Button**) kullanıldığı görülebilir. Fakat bu görüntünün oluşturulabilmesi için pek çok bileşen kullanılmıştır. Bu bileşenlerin ağacı sağ taraftaki VS Code penceresinde açık durumda olan **Widget Inspector** panelinde görülebilmektedir.

Flutter uygulamalarını başlatan `runApp` fonksiyonu kendisine parametre olarak gönderilen bileşeni uygulamanın kök bileşeni olarak kullanır. Widget Inspector penceresinde görüldüğü gibi **MaterialApp** bileşeni uygulamanın kök bileşenidir. Bu bileşen altında sayfanın görüntüsünü düzenleyen **Scaffold** bileşeni kullanılmıştır. **Scaffold** bileşenine bir **AppBar** ve bir **Column** bileşeni eklenmiştir. Adından anlaşılacağı üzere **AppBar** bileşeni uygulamanın üzerinde yer alan başlık çubuğu oluşturur. Başlık çubuğu üzerindeki **Mini Sınav** metninin gösterilmesi için de bir **Text** bileşeni kullanılmıştır. **Scaffold** bileşenine eklenen **Column** bileşeni nesnelerin alt alta gösterilmesi için kullanılan bir yayılım bileşenidir. Bu bileşene kullanıcı tarafından tanımlanmış **Question** ve **Answer** bileşenleri eklenmiştir. **Question** bileşeni içinde bir metin (**Text**) ve metnin yayılmasını düzenlemek için kullanılan bir **Container** bileşeni yer almaktadır. Benzer şekilde **Answer** bileşenleri içinde de düğmeler (**ElevatedButton**), düğmeler üzerindeki metinler (**Text**) ve düğmelerin yayılmasını düzenlemek için kullanılan **Container** bileşenleri yer almaktadır.



*Ekran Görüntüsü 7. Basit bir Flutter uygulamasının bileşen ağıacı*

Bu örnekte bileşen ağıacı kavramının tanıtılması için birçok bileşen içeren, görece karmaşık bir arayüz sunulmuştur. Her bir bileşenin kendine özgü görevleri ve yetenekleri bulunmaktadır. Şu an için yalnızca Flutter uygulamalarının arayüzlerinin bileşenlerden oluştuğunu anlaşılması ve bu bileşenlerin bir bileşen ağıacı şeklinde organize edilerek arayüzlerin düzenlendiğinin bilinmesi yeterlidir. İllerleyen başlıklarda bu uygulamada bahsedilen tüm bileşenlerden ve kullanıcılar tarafından oluşturulan kompozit bileşenlerden bahsedilecektir.

## 1.2. Uygula: Temel Flutter Uygulamasındaki Bileşenleri İnceleyelim

Birinci derste oluşturulan Flutter uygulamalarının bileşen ağaçlarını inceleyiniz. Ekrana "Merhaba" yazdırılan uygulama ve ekranındaki düğmeye kaç kez basıldığını gösteren uygulamalarda pek çok farklı bileşen kullanıldığı görülecektir.

### 1.3. Gözle: Simgeler

Flutter çok geniş bir simge kütüphanesi sunmaktadır. Bu simgeler Flutter'ın temel bileşenleri olduğundan projelere eklenmeleri için herhangi bir ek işlem gerekmektedir.

#### Simge Kütüphanesi

Icon sınıfı içinde gelen simge kütüphanesinde binlerce farklı tip ve stilde simge bulunmaktadır. Bu simgeleri incelemek için aynı zamanda Icon sınıfının tanıtım sayfası olan <https://api.flutter.dev/flutter/material/Icons-class.html> adresini ziyaret ediniz.

Uygulamalara simge eklemek için yeni bir **Icon** nesnesi üretilir. **Icon** sınıfının yapıcı metodu (**constructor**) eklenecek ikonun türünü belirleyen bir pozisyonel parametre ve simgenin görünüşünü belirleyen çeşitli isimli parametreler alır. Bu parametrelerin açıklamaları 10 numaralı satırda simgenin boyanacağı renk belirtilmektedir. Bu renk yine **Colors** sınıfı içinde ön tanımlı bir mavi tonudur (**indigo**). Renk (**color**) isimli parametresinin verilmesi zorunlu değildir. Bu durumda simge varsayılan renge boyanır. 11 numaralı satırda boyut bilgisinin verildiği **size** isimli parametresine **50** değeri verilmiştir. Bu parametre **Double** cinsinden sayısal değer alır. Bu parametrenin doldurulması da zorunlu değildir. Bu durumda simge varsayılan boyutunda gösterilir. Son olarak 12 numaralı satırda ekran okuyucular gibi araçların simge hakkında bilgi alabilmesi için yine girilmesi zorunlu olmayan **semanticLabel** parametresine **String** türünde **Hesap ikonu** bilgisi atanmıştır.

Tablo 2'de sunulmuştur.

```
1 import 'package:flutter/material.dart';
2 void main() {
3   runApp(MyApp());
4 }
5 class MyApp extends StatelessWidget {
6   Widget build(BuildContext context) {
7     return MaterialApp(
8       home: const Icon(
9         Icons.account_circle,
10        color: Colors.indigo,
11        size: 50,
12        semanticLabel: 'Hesap ikonu',
13      ),
14    );
15  }
16 }
```



Kod 66. Icon sınıfı yapıcı metodu ile simge tanımlama

Kod 66'da yeni bir simge üretmek için **Icon** sınıfının yapıcı metodu çalıştırılmaktadır. Bu kod ekranın sağında görülen mavi bir hesap ikonunu oluşturur. Kod 66'da 8-13 numaralı satırlar

arasında bir simge tanımlaması yapılmaktadır. Kalan satırlar simgenin ekranda gösterilebilmesi için gerekli temel rutinleri içermektedir. 9 numaralı satırda oluşturulacak simgenin türü belirtilmektedir. Simge türleri **Icons** sınıfı içinde ön tanımlı sabitlerdir. Bu sabitlerin listesine Simge Kütüphanesi kutusunda belirtilen adres ziyaret edilerek ulaşılabilir. Bu parametre pozisyonel olduğundan önünde herhangi bir parametre ismi verilmez. Bu parametrenin verilmesi zorunludur, aksi halde bir hata üretilir (Ör: **1 positional argument(s) expected, but 0 found.**).

10 numaralı satırda simgenin boyanacağı renk belirtilmektedir. Bu renk yine **Colors** sınıfı içinde ön tanımlı bir mavi tonudur (**indigo**). Renk (**color**) isimli parametresinin verilmesi zorunlu değildir. Bu durumda simge varsayılan renge boyanır. 11 numaralı satırda boyut bilgisinin verildiği **size** isimli parametresine **50** değeri verilmiştir. Bu parametre **Double** cinsinden sayısal değer alır. Bu parametrenin doldurulması da zorunlu değildir. Bu durumda simge varsayılan boyutunda gösterilir. Son olarak 12 numaralı satırda ekran okuyucular gibi araçların simge hakkında bilgi alabilmesi için yine girilmesi zorunlu olmayan **semanticLabel** parametresine **String** türünde **Hesap ikonu** bilgisi atanmıştır.

*Tablo 2. Icon sınıfı yapıcı metodunun aldığı parametreler*

Parametre	İşlev	Veri Türü	Örnek Değer
<b>Pozisyonel zorunlu</b>	Simgenin türünü belirler	Icons sınıfı içinde tanımlı sabit değer	<b>Icons.adb</b>
<b>color</b>	Simgenin rengini belirler	Color sınıfı içinde tanımlı sabit değer	<b>Colors.lightblue</b>
<b>size</b>	Mantıksal pikseller cinsinden simgenin boyutunu belirler	Double cinsinden sayısal değer	<b>32</b>
<b>semanticLabel</b>	Ekran okuyucu gibi yazılımların simge hakkında bilgi verebilmesi için kısa açıklama metni	String	“Android işletim sistemi ikonu”

#### 1.4. Uygula: Simge türleri

**Icon** sınıf içinde tanımlı simgelerin dört türü bulunur. Her simgenin bir temel formu, bir dış hat (**outlined**) formu, yuvarlatılmış (**rounded**) formu ve keskin köşeli (**sharp**) formu bulunmaktadır. Öğrencilerden bu simgeleri inceleyerek türleri tanımlamalarını ve aralarındaki farkları açıklamalarını isteyiniz.

#### 1.5. Uygula: semanticLabel parametresi

**semanticLabel** parametresinin işlevi, hangi kullanıcılar için önemli olabileceği, verilmemesi halinde bu kullanıcıların ne tür sorunlarla karşılaşabileceğini konusunda bir tartışma gerçekleştiriniz.

## Bilgi

Semantik, anlam bilimidir. Flutter özelinde, semantik kullanıcı arayüzüne eklenen bileşenlerin makineler tarafından yorumlanabilmesi ile ilgilidir. Bu amaçla oluşturulmuş **Semantic** adında bir bileşen ve bu bileşenle özdeleşmiş 50 kadar özellik bulunabilmektedir. Bu sayede Flutter, kullanıcı arayüzüne görerek yorumlayamayacak görme engelli bireyler (ekran okuyucular yoluyla) ya da makinelerin (Ör: arama motorları ya da sayfaların anlam çıkarabilen yapay zeka uygulamaları) anlam çıkarılabilmesi için ekstra ipuçları üretmektedir. Bu çerçevede **semanticLabel** parametresinin sayfanın erişilebilirliğini artırdığı söylenebilir.

## 1.6. Text Bileşeni

Sayfaya metin eklemek için **Text** bileşeni kullanılır. **Text** bileşeni yapıcı metodu **String** türünde değer alan bir zorunlu pozisyonel parametre alır. Bu parametre bileşen içinde gösterilecek metindir. Bu parametre pozisyonel ve zorunlu olduğundan verilmemesi ya da isimli parametrelerin arkasında verilmesi hata üretir (Ör: **1 positional argument(s) expected, but 0 found**).

## Bilgi

Tasarım çalışmalarında sıkılıkla Lorem Ipsum adındaki Latince metin kullanılmaktadır. Bu metne ulaşmak için <https://www.lipsum.com/> adresini ziyaret edebilirsiniz.

```

3   class MyApp extends StatelessWidget {
4       Widget build(BuildContext context) {
5           return MaterialApp(
6               home: Scaffold(
7                   appBar: AppBar(
8                       title: Text('Metin bileşeni'),
9                   ),
10                  body: Text("Lorem ipsum dolor sit amet, consectetur
11                     adipiscing elit, sed do eiusmod tempor incididunt
12                     ut labore et dolore magna aliqua.",
13                     //maxLines: 4,
14                     //overflow: TextOverflow.ellipsis,
15                     ),
16                 );
17 }

```

Kod 67. Text bileşeni ile sayfaya metin ekleme

Kod 67'de oluşturulan uygulamaya iki adet Text bileşeni ile metin eklenmiştir. Öncelikle 8 numaralı satırda başlık çubuğu kisa bir başlık eklenmektedir. Flutter uygulamalarında ekranda görünen tüm metinler **Text** bileşeni ile üretilmektedir. Diğer **Text** bileşeni ise 10-13 numaralı satırlar arasında eklenmiştir. Bu bileşende isimli parametreler yazılmış fakat yorum satırına çevrilmiştir. Bu parametrelerin alabileceği değerleri değiştirmek etkilerini inceleyiniz.

### Bilgi

Text bileşenleri bunlar dışına pek çok isimli parametre ile düzenlenebilir. Bu bölümde en çok kullanılan özelliklere yer verilmiştir. Diğer parametreleri incelemek için <https://api.flutter.dev/flutter/widgets/Text-class.html> adresini ziyaret ediniz.

Metinler en fazla kullanılan içerikler arasında olduğundan **Text** bileşenleri pek çok düzenleme seçeneği sunar. **Text** bileşeninin yapıcı metodу Tablo 3'te belirtilen isimli parametreleri alabilir. **maxLines** parametresi **Text** bileşeninde gösterilebilecek maksimum satır sayısını belirler. Bileşendeki metnin bu satır sayısını aşması durumunda, metin **overflow** parametresinde belirtilen değere göre gösterilir.

Tablo 3. Text bileşeni yapıcı metodunda kullanılabilen isimli parametreler

Parametre	Açıklama	Veri Tipi	Örnek Değer
<b>maxLines</b>	Gösterilebilecek maksimum satır sayısı	<b>int</b>	<b>1, 4</b>

<b>Overflow</b>	Bileşen boyutlarını aşan metinlerin yorumlanması yöntemi	<b>TextOverflow nesnesi</b>
<b>Style</b>	Bileşende gösterilen metnin stil özellikleri	<b>TextStyle nesnesi</b>
<b>textAlign</b>	Metnin yatay hizalanma yöntemi	<b>TextAlign nesnesi</b>

#### 1.6.1. Uygula: *TextOverflow sınıfı değerleri ve etkileri*

Oluşturacağınız bir **Text** bileşenine uzunca bir metin ekleyerek (ör: Lorem ipsum metni) bu metnin **maxLines** parametresinde vereceğiniz değeri aşmasını sağlayınız. Ardından **overflow** parametresine Tablo 4'teki değerleri uygulayarak sonuçları gözlemleyiniz. **visible** değerinin etkilerini görebilmek için **Text** bileşeninin kendisinden küçük bir **Container** içine yerleştirilmesi gerekecektir.

Tablo 4. *TextOverflow sınıfındaki sabit değerlerin etkileri*

Değer	Açıklama
<b>TextOverflow.clip</b>	Aşan satırları göstermez
<b>TextOverflow.fade</b>	Aşan satırları solma efekti uygularak belirtir. Satırlar gösterilmmez.
<b>TextOverflow.ellipsis</b>	Aşan satırları son satıra üç nokta ekleyerek belirtir.
<b>TextOverflow.visible</b>	Kapsayıcı boyutlarını aşan satırları gösterir.

#### 1.6.2. *TextStyle Sınıfı*

**TextStyle** bileşeni **Text** bileşenlerinin **style** parametresine uygulanarak bu bileşen içindeki metinlerin stil özelliklerini düzenler. Tablo 5'te **TextStyle** bileşeninin sık kullanılan isimli parametreleri, bu parametrelere atanacak veri tipleri ve örnek değerler gösterilmektedir.

Kod 68'de Ekran Görüntüsü 8'deki uygulamada yer alan Lorem Ipsum metninin stil özelliklerini düzenleyen **TextStyle** nesnesi üretilmektedir. **TextStyle** sınıfının yapıcı metodу pek çok isimli parametre almaktadır. 18 numaralı satırda fontFamily parametresine **Roboto** fontu atanmaktadır. Fontların isimleri **String** türünde verilir. Uygun font bulunamaması halinde varsayılan fontlar kullanılacaktır. **fontStyle** parametresi, fontun yatıklığını düzenleyebilmektedir. Bu parametre **FontStyle** sınıfındaki sabit değerleri (**normal/italic**) alır. 19 numaralı satırda **FontStyle.normal** değeri verilerek yazının normal yazılması sağlanmıştır.

Tablo 5. *TextStyle bileşeninde sık kullanılan parametreler ve örnek değerleri*

Parametre	Açıklama	Veri Tipi	Örnek Değer
<b>fontWeight</b>	Yazı kalınlığı	<b>FontWeight sınıfı sabitleri</b>	<b>FontWeight.bold</b>
<b>fontStyle</b>	Yazının yatıklığı	<b>FontStyle sınıfı sabitleri</b>	<b>FontStyle.italic</b>

<b>color</b>	Yazı rengi	<b>Colors sınıfı sabitleri</b>	<b>Color.teal</b>
<b>fontSize</b>	Yazı boyutu (varsayılan 14)	<b>Doble</b>	<b>20</b>
<b>height</b>	Yazı boyutunun katı cinsinden satır yüksekliği	<b>Double</b>	<b>2</b>
17 <b>style: TextStyle(</b> 18 <b>fontFamily: "Roboto",</b> 19 <b>fontWeight: FontWeight.w300,</b> 20 <b>fontSize: 16,</b> 21 <b>color: Colors.grey[800],</b> 22 <b>height: 1.6,</b> 23 <b>)</b>			

Ecran Görüntüsü 8. TextStyle bileşeni kullanmak

Kod 68. TextStyle nesnesi tanımlamak

**fontWeight** parametresi ise yazı kalınlığını düzenler. Bu parametre **FontWeight** sınıfında sunulan sabit değerleri alabilmektedir. **normal** ve **bold** değerleri yanında **w100-w900** arasında kalınlık değerleri sağlanmaktadır. Bu değerler 100'ün katları şeklinde artar. **w100** çok ince, **w900** ise çok kalındır. Varsayılan değer **normal** ya da **w400**'dır. 20 numaralı satırda **FontWeight.w300** değeri ile normalden daha ince bir yazı kalınlığı belirlenmiştir. **fontSize** parametresi **piksel** cinsinden yazı boyutunu belirler. Yazıları büyütmek için bu parametre kullanılır. Bu parametre **Double** cinsinden değerler kabul eder. 21 numaralı satırda yazının boyutu **16** piksel olacak şekilde düzenlenmiştir. Varsayılan yazı boyutu **14** pikseldir. **color** parametresi yazı rengini belirler. Bu parametre **Colors** sınıfı tarafından sağlanan sabit renk değerlerini alabilir. 22 numaralı satırda yazının rengi koyu bir gri tonuna çevrilmiştir. **height** parametresi ise yazıtının satır yüksekliğini belirler. Bu parametre yazı boyutunun katı cinsinden **Double** cinsinden değerler alır. 23 numaralı satırda satır yüksekliği yazı boyutunun **1,6** katı olacak şekilde düzenlenmiştir.

#### 1.6.3. Uygula: TextAlign Bileşeni ile Metinleri Hizalamak

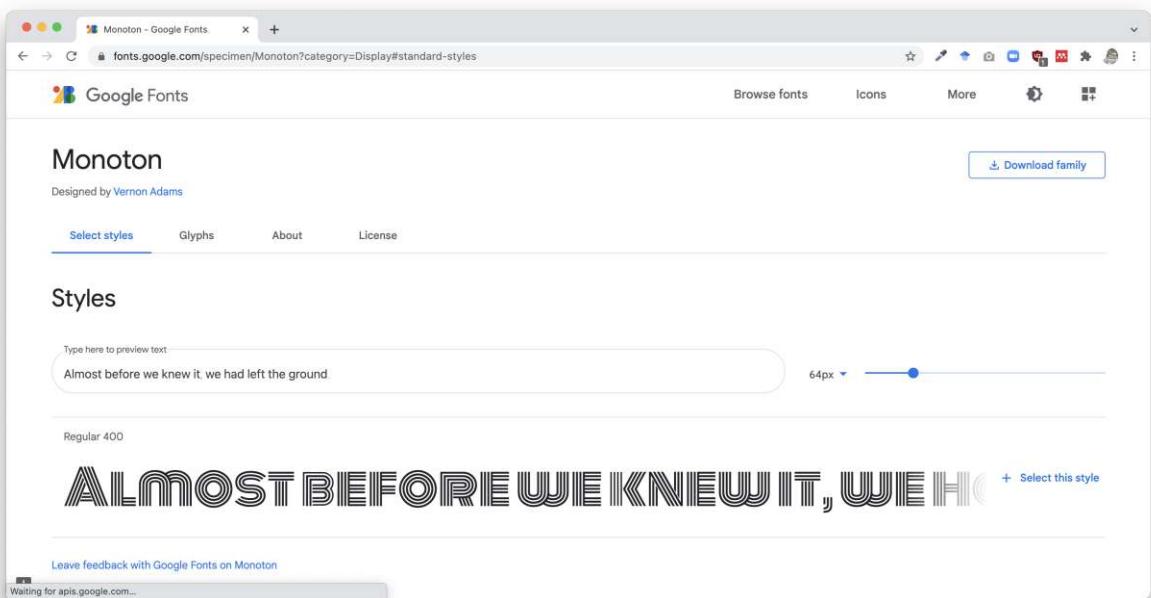
**Text** bileşeninin **textAlign** isimli parametresi metinlerin yatay hizalanmasını düzenler. Bu parametre **TextAlign** sınıfı tarafından sağlanan sabit değerleri alır. Sayfa genişliğini kaplayan bir **Container** içine yerlestireceğiniz **Text** bileşeni üzerinde **TextAlign** sınıfı tarafından sağlanan sabit değerlerin etkilerini gözlemleyiniz.

#### Bilgi

**Text** bileşenleri varsayılan olarak içerik kutuları kadar yer kapladığından, **TextAlign** sınıfı etkilerinin gözlenebilmesi için kutularının büyütülmesi zorunludur.

#### 1.6.4. Uygula: Google Fontlarını Kullanmak

Google firması tasarımlarda kullanılabilecek pek çok fontu ücretsiz olarak sağlamaktadır. Bu fontları incelemek için <https://fonts.google.com> adresini ziyaret ediniz.



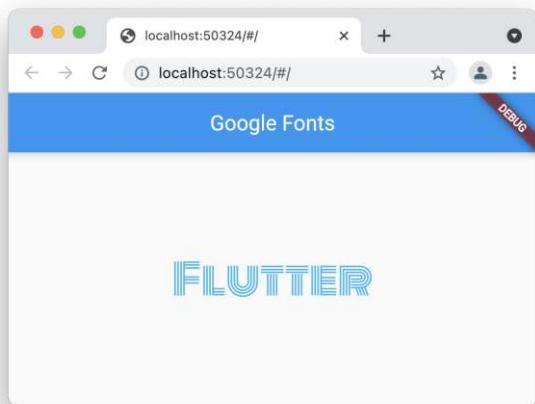
Ekran Görüntüsü 9. Google Fontlarından Monoton fontunun sayfası

Ekran Görüntüsü 9'da Google Fonts sayfasında sunulan görselliği yüksek (display) fontlardan biri olan Monoton gösterilmektedir. Bu fontlar sağ üstte yer alan Download family düğmesi kullanılarak ücretsiz olarak indirilebilmektedir. Burada sunulan fontlarla ilgili önemli bir sınırlılık Türkçe karakter desteği vardır. Her fontta Türkçe karakterler (glyph) bulunmayabilir. Bu nedenle fontların sağladığı karakterler incelenerek indirilmesi önerilmektedir. Bu amaçla arama ekranındaki dil kutusunun kullanılması önerilmektedir.

Ekran Görüntüsü 10'da Google Fonts tarafından sağlanan Monoton fontu projeye eklenderek sayfada kullanılmıştır. Aşağıdaki adımları izleyerek bu projeyi gerçekleştirebilirsiniz.

Bu uygulama için <https://fonts.google.com/specimen/Monoton?category=Display> adresindeki Monoton isimli fontu indiriniz. Bu font, görseliği yüksek (display) yazılar yazabilmek için gerekli karakterleri sağlamaktadır. İndirme bağlantısına tıklandığında **Monoton-Regular.ttf** dosyası bilgisayarınıza indirilecektir.

Dikkat



Ekran Görüntüsü 10. Monoton fontu kullanılan proje

Bazı Font aileleri kalın, ince ve italik font formları için birden fazla ttf dosyası içerebilmektedir. Bu formlardan kullanılacak olanların dosyalarının projeye içeriilmesi gerekecektir.

Bu dosyayı uyuglamanızda açacağınız bir **fontlar** klasörüne kopyalayınız. Dışarıdan eklenecek resimler, videolar, fontlar gibi nesnelerin kendilerine özgü klasörlerde tutulması proje klasörlerini düzenli tutmak için kullanılan bir yaklaşımdır.

Eklenen font dosyalarının projede kullanılabilmesi için bu dosyaların **pubspec.yaml** dosyasında içeriilmesi gerekmektedir. Bu amaçla **pubspec.yaml** dosyasında 77 numaralı satırda yorum satırı halindeki **fonts:** bölümü açılarak Kod 69'deki kodlar eklenir. Yaml dosyalarında içerikler hizalama yolu ile yapılandırıldığından tireler ve girintilere dikkat edilmesi gerekmektedir. Girinti eklemek için klavyedeki TAB tuşu kullanılır.

```
77  fonts:  
78      - family: Monoton  
79          fonts:  
80              - asset: fontlar/Monoton-Regular.ttf
```

Kod 69. pubspec.yaml dosyasında font dosyasını içermek

Kod 70'de 15-19 numaralı satırlar arasında Eklenen **Monoton** fontunun **Text** bileşenine uygulanması için bu bileşenin style parametresine atanacak bir **TextStyle** nesnesi oluşturulmaktadır. 16 numaralı satırda **Monoton** fontu **TextStyle** bileşenine uygulanmaktadır. Gerekli boyut ve renk ayarlamaları yapılarak yazının istenilen şekilde gösterilmesi sağlanabilir.

Kod 70'te 12 numaralı satırda **Center** bileşeni oluşturulmaktadır. Bu bileşen kendisine çocuk olarak verilen bileşenleri yatay ve dikey olarak ortalayan bir kapsayıcıdır. Bu sayede oluşturduğumuz yazı sayfada ortada gösterilmektedir.

```

5   class MyApp extends StatelessWidget {
6       Widget build(BuildContext context) {
7           return MaterialApp(
8               home: Scaffold(
9                   appBar: AppBar(
10                      title: Text("Google Fonts"),
11                  ),
12                  body: Center(
13                      child: Text(
14                          "Flutter",
15                          style: TextStyle(
16                              fontFamily: "Monoton",
17                              fontSize: 40,
18                              color: Colors.lightBlue[400],
19                          ),
20                      ),
21                  ),
22                  ),
23              );
24      }
25  }

```

*Kod 70. Monoton fontunun Text bileşenine uygulanması*

## 1.7. Gözle ve Uygula: GoogleFonts Paketini Kullanmak

Bir önceki örnekte Google Fontlarından biri proje klasörüne indirilerek kullanılmıştır. Bu işlem fontların bulunması, indirilmesi ve yaml dosyası üzerinden projeye eklenmesini gerektirdiğinden görece zahmetlidir. Google Fontları çevrimiçi kullanılabiligidinden, bu işlemi kolaylaştırmak için GoogleFonts paketi üretilmiştir. Bu paket Flutter geliştirme ortamına entegre olmadıktan projelere eklenmesi gerekmektedir. Bu işlem için Kod 71'deki kod satırı proje klasörü içinde çalıştırılır. Bu kod gerekli dosyaları indirerek, **pubspec.yaml** dosyası içine ilgili kodları ekleyecktir.

```
flutter pub add google_fonts
```

*Kod 71. GoogleFonts paketinin projeye eklenmesi*

```

1 import 'package:flutter/material.dart';
2 import 'package:google_fonts/google_fonts.dart';
3 void main() {
4     runApp(MyApp());
5 }
6 class MyApp extends StatelessWidget {
7     Widget build(BuildContext context) {
8         return MaterialApp(
9             title: 'Google Fonts Paketi Kullanımı',
10            home: Scaffold(
11                appBar: AppBar(
12                    title: const Text("Google Fonts Paketi Kullanımı"),
13                ),
14                body: Center(
15                    child: Text(
16                        "Google Fonts",
17                        style: GoogleFonts.monoton(
18                            fontSize: 30,
19                            fontWeight: FontWeight.w300,
20                        ),
21                    ),
22                ),
23            ),
24        );
25    }
26 }

```

Kod 72. Google Fonts paketi ile monoton fontunu uygulamak



Kod 72'de 17-20 numaralı satırlar arasında Monoton fontunun oluşturulan bir **Text** bileşenine uygulanışı görülmektedir. **GoogleFonts** paketi 2 numaralı satırda sayfaya eklenmektedir. Kod 71'de paket komut satırı üzerinden projeye eklense de, **main.dart** dosyası içinde kullanılabilmesi için bu sayfada **import** sözcüğü ile ekleme işleminin gerçekleştirilmesi zorunldur. **GoogleFonts** paketi her bir yazı tipi için bir metot sağlamaktadır (Ör: **GoogleFonts.roboto()**). Bu örnekte **monoton** fontu için sağlanan metot font

yüksekliği **30**, kalınlık değeri de **w300** olacak şekilde çağrılmıştır. **Text** bileşeni **Center** bileşeni içinde olduğundan sayfada ortalanmıştır. **AppBar** kullanılarak da başlık çubuğu eklenmiştir.

## 1.8. Gözle ve Uygula: Temaları Kullanmak

Temalar, uygulamalar içinde yazı tipi ve renk karakteristikleri tanımlamak ve paylaşmak amacıyla kullanılır. Tema üretmek için Theme bileşeni kullanılır. Bu bileşen MaterialApp bileşeninin köküne yerleştirildiğinde tüm uygulamada geçerli olan genel tema işlevi görür. Fakat, giriş sayfası ya da formlar kullanılan sayfalar gibi özel bölümler için yerel temalar da oluşturulabilir. Bu sayede uygulamada görüntü birliği sağlanmış olur.

Her Flutter uygulaması varsayılan tema ile üretilir. Bu temayı kullanmak için **Theme** bileşeninin **of()** metodu kullanılır. Bu metot bileşen ağacında en yakında tanımlı temadaki bilgileri bularak geçerli bileşene uygular. Bileşen ağacında, çağrı yapan bileşene yakında yerel bir **Theme** tanımlı ise bu temadaki bilgiler kullanılır. Aksi halde ağaçta en üstte yer alan genel temaya ulaşılır.

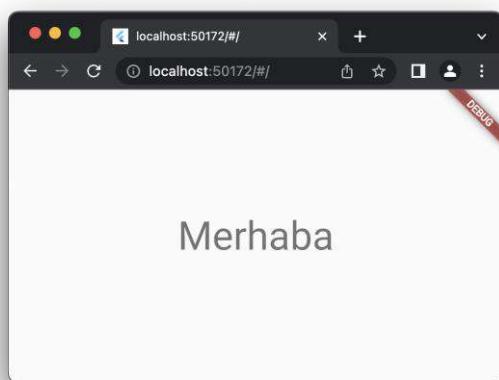
### ThemeData

Her Flutter uygulaması varsayılan bir genel tema ile üretilir. Bu temalarda tanımlanan özellikler tüm uygulama boyunca paylaşıldığından bu temaların kullanılabilir özelliklerini bilmek önemlidir. ThemeData sınıfı uygulamadaki bileşenler için (Ör: düğmeler, barlar, form elementleri) özellikler belirler. ThemeData içinde tanımlanabilen özellikleri incelemek amacıyla <https://api.flutter.dev/flutter/material/ThemeData-class.html> adresini inceleyiniz.

```
3   class MyApp extends StatelessWidget {
4     Widget build(BuildContext context) {
5       return MaterialApp(
6         home: Scaffold(
7           body: Center(
8             child: Text(
9               "Merhaba",
10              style: Theme.of(context).textTheme.headlineLarge,
11            ),
12          ),
13        ),
14      );
15    }
16 }
```

Kod 73. Text bileşenine tema uygulamak

Kod 73'de 8. satırda ekranda Merhaba yazısını göstermek için bir Text bileşeni oluşturulmuştur. Bu bileşen, ekranda ortalanması için bir Center bileşeni içine yerleştirilmiştir. 10 numaralı satırda Text bileşenin görünümünü düzenlemek için style parametresi kullanılmıştır. Bu parametreye daha önceki örneklerde olduğu gibi bir TextStyle bileşeni sağlanmamış; Theme.of() metodu ile kullanılan BuildContext'in metin teması içindeki headlineLarge (büyük başlık) özelliğine ulaşılmıştır. Bu sayede metin büyük bir başlık biçiminde gösterilmiştir (Ekran Görüntüsü 11).



Ecran Görüntüsü 11. headlineLarge teması uygulanmış bir Text bileşeni

## TextTheme

Metinler en çok kullanılan arayüz elementlerindendir. Bu nedenle uygulamalarda kullanılabilcek metin stillerinin bilinmesi gereklidir. <https://api.flutter.dev/flutter/material/TextTheme-class.html> adresinde kullanılabilcek stillerin isimleri ve biçimleri incelenebilir.

### 1.8.1. Varsayılan Temayı Düzenlemek

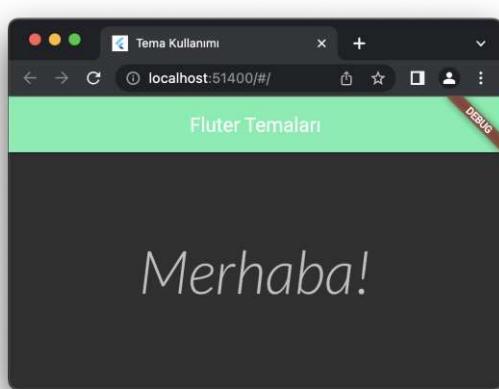
Flutter uygulamaları için üretilen varsayılan tema üzerinde güncelleme yapılmasına izin verilir. Bu amaçla **MaterialApp** bileşenin theme parametresine gönderilen **ThemeData** sınıfı kullanılır. Bu örnekte yine GoogleFonts paketi kullanılcagından örneğe başlamadan önce projeye paketin eklenmesi gereklidir. Ardından kodlar eklenerek genel tema üzerinde düzenleme yapılabilir.

Kod 74'te 8 ve 17. Satırlar arasında tanımlanan **ThemeData** ile uygulama teması düzenlenmektedir. 8. Sayıda **brightness** parametresine verilen **Brightness.dark** değeri ile uygulama koyu temada gösterilmektedir. Flutter uygulamalarında bu değer varsayılan olarak **Brightness.light** şeklinde olduğundan uygulamalar açık artan üzerine koyu yazılarla gösterilmektedir. 9. satırda **primaryColor** parametresi üzerinden uygulamanın birincil rengi **greenAccent** (■) şeklinde belirlenmektedir. 10-15. satırlar arasında ise **textTheme** parametresine atanmış **TextTheme** sınıfı ile uygulamadaki **headlineLarge** stili güncellenmektedir. Bu stilin yazı tipi Google fontlarından lato, yüksekliği 60 birim, kalınlığı 300 birim ve yatıklığı italic olarak güncellenmektedir. Bu düzenlemeler sonunda oluşan arayüz **Hata! Başvuru kaynağı bulunamadı.**'de sunulmuştur.

```

1 import 'package:flutter/material.dart';
2 import 'package:google_fonts/google_fonts.dart';
3 class MyApp extends StatelessWidget {
4     Widget build(BuildContext context) {
5         return MaterialApp(
6             title: "Tema Kullanımı",
7             theme: ThemeData(
8                 brightness: Brightness.dark,
9                 primaryColor: Colors.greenAccent,
10                textTheme: TextTheme(
11                    headlineLarge: GoogleFonts.lato(
12                        fontSize: 60,
13                        fontStyle: FontStyle.italic,
14                        fontWeight: FontWeight.w300,
15                    ),
16                ),
17            ),
18            home: MyHomePage(),
19        );
20    }
21 }
22 void main() {
23     runApp(MyApp());
24 }
```

Kod 74. Varsayılan temayı düzenlemek



Ekran Görüntüsü 12. Varsayılan teması güncellenmiş uygulama arayüzü

Oluşturulan temanın arayüze uygulandığı kodlar Kod 75'te sunulmuştur. Bu kodlarda **StatelessWidget** türündeki **MyHomePage** bileşeni üretilmektedir. Bu sınıfın, Kod 74'te 18 numaralı satırından çağrıldığına dikkat ediniz. Güncellenen temadaki **primaryColor** değeri 33 numaralı satırda **AppBar** bileşenine artalan rengi olarak atanmaktadır. Güncellenen **headlineLarge** stili ise 38 numaralı satırda oluşturulan metin bileşenine uygulanmaktadır. Bu yolla tanımlanan renk ve stiller uygulama boyunca kullanılabilir olmaktadır. Bunun yanında, Kod 74'te gerçekleştirilecek stil güncellemesinin tüm

uygulamaya yansıtılacağı unutulmamalıdır. Böylece uygulamanın görünümü tek bir noktadan düzenlenebilir hale gelmiştir.

```
28 class MyHomePage extends StatelessWidget {  
29     Widget build(BuildContext context) {  
30         return Scaffold(  
31             appBar: AppBar(  
32                 title: Text("Flutter Temaları"),  
33                 backgroundColor: Theme.of(context).primaryColor,  
34             ),  
35             body: Center(  
36                 child: Text(  
37                     'Merhaba!',  
38                     style: Theme.of(context).textTheme.headlineLarge,  
39                 ),  
40             ),  
41         );  
42     }  
43 }
```

*Kod 75. Güncellenen temanın arayüze uygulanması*

## 1.9. Resimler

Uygulamalarda en çok kullanılan içeriklerden biri de resimlerdir. Flutter pek çok kaynaktan (ör: dosyalar, proje varlıkları, ağ kaynakları) resim dosyalarını çekerek ekranda gösterebilir.

### Tartışma

Flutter uygulamalarında resim dosyaları proje varlıkları olarak ya da bir ağ kaynağından eklenebilmektedir. Bu iki yöntemin avantaj ve dezavantajlarını sırala sorunuz.

Proje varlığı olarak resim kullanmanın avantajı bu resimlerin her zaman ulaşılabilir olmasıdır. Resimler projenin parçaları olduğundan, uygulama kurulduğu anda telefona da yüklenmiş olurlar. Bu sayede resimlere erişim sorunu olmaz. Bunun yanında mobil cihaz üzerindeki resimler olduklarından bu resimler uygulamaya oldukça hızlı yüklenirler. Fakat bu resimler sabittir. Bu resimlere yeni resim eklemek için projenin tekrar derlenmesi ve mobil cihaza aktarılması gerekecektir. Ayrıca her resim dosyasının üretilecek uygulama boyutuna ekleneceği unutulmamalıdır. 2MB boyutundaki 50 resim içeren bir projenin boyutu otomatik olarak 100MB artmış olacaktır.

Ağ kaynaklarındaki resimlerinin avantajı projeye sonradan ekleniyor oluşulardır. Bir ağ adresine (URL) konan her resim proje çalışıyorken indirilerek ekranda gösterilebilmektedir. Bunun yanında, bu resimlerde değişiklik yapmak da olasıdır. Fakat ağ kaynaklarına erişim çeşitli nedenlerle sınırlanabilir. Örneğin, kullanıcının internet erişiminin olmaması durumunda

resimler gösterilemez. Benzer şekilde hedef gösterilen resmin ağ kaynağından kaldırılması halinde de bu resim gösterilemeyecektir. Son olarak ağ resimlerinin indirilerek projede gösterildiği unutulmamalıdır. Özellikle yüksek kaliteli resimlerin yoğun kullanılması, kullanıcıların kota problemleri yaşamalarına neden olabilir. Son olarak, düşük hızlı internet bağlantılarında yüksek kaliteli resimlerin gösterilmesi için belirli bir indirme süresinin geçeceği unutulmamalıdır.

Her bir resim gösterme yönteminin avantajları ve dezavantajları olduğundan Flutter çeşitli kaynakların kullanılmasına izin verir. Programcıların bu yöntemleri harmanlayarak çözüm üretmesi beklenir. Örneğin, proje için çok önemli görsellik unsurları olan artalan resimleri, menü resimlerinin proje varlığı olarak tutulması doğru bir karardır. Fakat değişken içeriklere sahip galerilerin ağ kaynaklarından çekilmesi daha doğru olacaktır.

Flutter uygulamalarında çeşitli türlerde resimler kullanılabilir. Flutter uygulamalarında JPG, PNG, GIF, WEBP, BMP ve WBMP türündeki resim dosyaları kullanılabilirmektedir.

### Tartışma

Flutter uygulamalarında farklı türlerde resimler kullanılabilirmektedir. Her resim türünün kendine özgü avantajları bulunmaktadır. Örneğin GIF resimleri hareketli olabilirken, PNG resimlerinde saydamlık kullanılabilirmektedir. Sınıfa bu resim türlerinden hangilerini bildiklerini ve bu türlerin özel avantajlarını sorunuz.

Ağ kaynaklarından resim eklemek oldukça basit bir işlemidir. Bunun için öncelikle bir ağ kaynağı resmini getirebilecek URL adresinin bulunması gerekmektedir. <https://www.pexels.com> adresine giderek beğendiğiniz bir resmin URL adresini bulunuz. Bu amaçla Google Chrome tarayıcısında beğenilen resim açıldıktan sonra (Ör: <https://images.pexels.com/photos/40984/animal-ara-macao-beak-bird-40984.jpeg?cs=srgb&dl=pexels-public-domain-pictures-40984.jpg&fm=jpg>) resme sağ tıklayarak Resim Adresini Kopyala seçeneğini kullanabilirsiniz.

### Pexels.com

Çeşitli internet sitelerinde halka açık resim arşivleri bulunmaktadır. En büyük arşivlerden biri <https://www.pexels.com> adresinden ulaşabilecek olan Pexels sitesidir. Bu sitede resimler ve kısa videolar bulunabilmektedir.

```

1 import 'package:flutter/material.dart';
2 void main() => runApp(MyApp());
3 class MyApp extends StatelessWidget {
4     Widget build(BuildContext context) {
5         return MaterialApp(
6             home: Scaffold(
7                 appBar: AppBar(
8                     title: Text('Resim Bileşeni'),
9                 ),
10                body: Image.network(
11                    'https://images.pexels.com/photos/40984/animal-ara-macao-beak-
bird-40984.jpeg?cs=srgb&dl=pexels-public-domain-pictures-40984.jpg&fm=jpg'),
12                ),
13            );
14        }
15    }

```

Kod 76. Ağ kaynağından resim eklemek

Kod 76'da 10 numaralı satırda projeye bir **Image** bileşeni eklenmektedir. **Image** bileşeninin yapıçı metodlarından biri **network** metodudur. Bu metot pozisyonel bir ağ kaynağı parametresi almaktadır. **Image** bileşeni bu ağ adresindeki görseli otomatik olarak indirerek sayfaya resim olarak eklemektedir.

Ekran Görüntüsü 13'te görüldüğü gibi **Image** bileşeni ile eklenen resim sayfada otomatik olarak gösterilmektedir. **Image** bileşeninin yapıçı metodları bu resmin sayfadaki görüntüsünü değiştirebilecek pek çok isimli parametre sunmaktadır. Bu parametreler, kısa açıklamaları ve kullanılabilecek örnek değerler Tablo 6'de sunulmuştur.

**width** ve **height** parametreleri eklenen görsellerin boyutlarını düzenlemek için kullanılabilir. Bu parametreler **Double** cinsinden değerler alır. **color** ve **colorBlendMode** parametreleri ise resimlerin üzerine renk etkileri uygulamak için kullanılır. **color** parametresi **Colors** sınıfı sabitleri tarafından sağlanan renkleri kabul eder. **colorBlendMode** ise **BlendMode** sınıfı sabitleri tarafından tanımlanan sabitleri kabul eder. Bu sabitler, **color** parametresinde



Ekran Görüntüsü 13. Image bileşeni ile resim eklemek

tanımlanan rengin, **Image** bileşenindeki resmin piksellerine uygulanma şeklini tanımlar.

Tablo 6. Image bileşeninde sık kullanılan parametreler ve örnek değerleri

Parametre	Açıklama	Veri Tipi	Örnek Değer
<b>color</b>	Resim üzerine uygulanacak renk	<b>Colors sınıfı sabitleri</b>	<b>Colors.amber</b>
<b>colorBlendMode</b>	color parametresinde verilen rengin uygulanma metodu	<b>BlendMode sınıfı sabitleri</b>	<b>BlendMode.multiply</b>
<b>fit</b>	Resmin kapsayıcı elemente yerleştirilme metodu	<b> BoxFit sınıfı sabitleri</b>	<b>BoxFit.cover</b>
<b>height</b>	Resim yüksekliği	<b>Double</b>	<b>200</b>
<b>width</b>	Resim genişliği	<b>Double</b>	<b>300</b>

#### 1.9.1. Uygula: Renk etkilerini inceleyelim

Eklenen resimlere Flutter ile gelen renk etkileri otomatik olarak uygulanabilir. Bu uygulamada iki adet flamingonun gösterildiği bir resim dosyası kullanılmaktadır. Öğrenciler Flutter tarafından desteklenen formatlarda olmak kaydıyla istedikleri resim dosyalarını kullanabilirler. Örnekteki dosyaya ulaşmak için <https://www.pexels.com/photo/two-pink-flamingos-1181181/> adresi ziyaret edilebilir. Bu uygulamada resimler varlık olarak ekleneceğinden bu dosyanın indirilerek proje kök klasöründeki bir klasöre (Ör: resimler) kopyalanması gerekmektedir. Daha önceki örneklerde olduğu gibi bu varlıkların **pubspec.yaml** dosyası içinde tanımlanması gerekecektir.

```
47 assets:  
48   - resimler/flamingo.jpg
```

Kod 77. Flamingo görselinin projeye tanıtılması

Kod 77'te projede üretilen resimler klasörü içine yerleştirilen **flamingo.jpg** isimli dosyanın **pubspec.yaml** dosyası içinde projeye tanıtıldığı kodlar yer almaktadır. **assets:** ifadesinin bir TAB, **- resimler/flamingo.jpg** ifadesinin ise iki TAB ile girintilendiridine dikkat ediniz.

Bu tanımlıla birlikte **flamingo.jpg** dosyası projeye varlık olarak eklenmiş olur. Bu sayede **Image** bileşeninin **asset** yapıçı metodu kullanılabilecektir.

```

12 Image.asset(
13   "resimler/flamingo.jpg",
14   color: Colors.lime,
15   colorBlendMode: BlendMode.multiply,
16 )

```

Kod 78. asset yapıcı metodu ile resim eklemek ve renk etkisi uygulamak

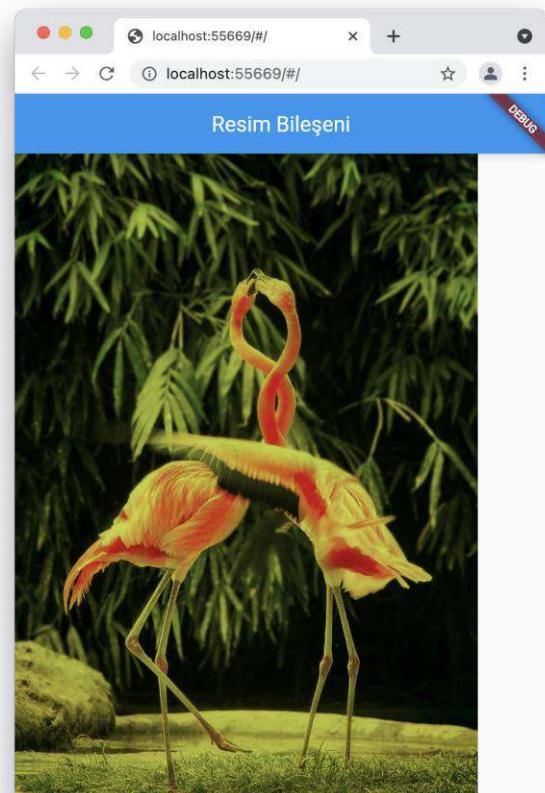
Yer kısıtlarından dolayı Kod 78'de tüm proje verilmemiş; sadece resmin eklenmesi ve renk etkisinin uygulanması gösterilmiştir. 12 numaralı satırda **Image** bileşeninin **asset** yapıcı metodu kullanılarak projeye eklenmiş bir varlık üzerinden resim eklemesi yapılmıştır. Bu metot ilk pozisyonel parametresinde varlığın tanımlanmış yolunu kabul eder. 14 numaralı satırda bu resmin üzerine açık yeşil rengi uygulanmaktadır. Fakat, bir karıştırma metodu verilmemişinde bu renk resmin her bir pikseline uygulandığından tüm resim bu renge boyanacaktır. Tanımlanan rengin resmin piksellerindeki renklere uygulanma metodunu belirlemek için **colorBlendMode** isimli parametresi kullanılır. Bu parametre **BlendMode** sınıfındaki sabit değerleri alır. Örnekte açık yeşil rengi **multiply** metodu ile uygulanarak Ekran Görüntüsü 14'deki görsel elde edilmiştir. **colorBlendMode** parametresinde **BlendMode**. İfadesinden sonra önerilen değerleri uygulayarak etkilerini inceleyiniz.

### 1.9.2. Uygula: Sığdırma yöntemleri

**Image** bileşenin **fit** özelliği resmin içinde oluşturduğu kapsayıcı elemente sığdırılması ile ilgili yöntemi belirler. Bu parametre **BoxFit** sınıfı sabitlerini kabul eder. Bir önceki uygulamadaki resmi 300px genişliğinde, 400px yüksekliğindedeki bir **Container** elementi içine yerleştirerek **BoxFit** sınıfı sabitlerinin etkilerini inceleyiniz.

## 2. TASARLA ve ÜRET

Arayüz tasarımlarında en çok kullanılan içeriklerden biri metinlerdir. Metinlerin kolay algılanması, içeriğin anlaşılabilirliği için oldukça önemlidir. Görsel iletişim tasarımda metinlerin tasarlanması ile ilgili alan tipografidir. Tipografi metinlerin biçim (font), yükseklik, satır aralığı, satır uzunluğu gibi karakteristiklerinin düzenlenmesi ile ilgilenen bir alandır. Geçmişte oldukça eskiye dayanan bu alan hakkında daha fazla bilgi almak için Anadolu Üniversitesi'nin Dijital Ders Platformunda açık ders olarak sunulan Tipografi (<https://ddp.anadolu.edu.tr/#/course/GIT106U>) dersini inceleyebilirsiniz. Bu portalda e-devlet

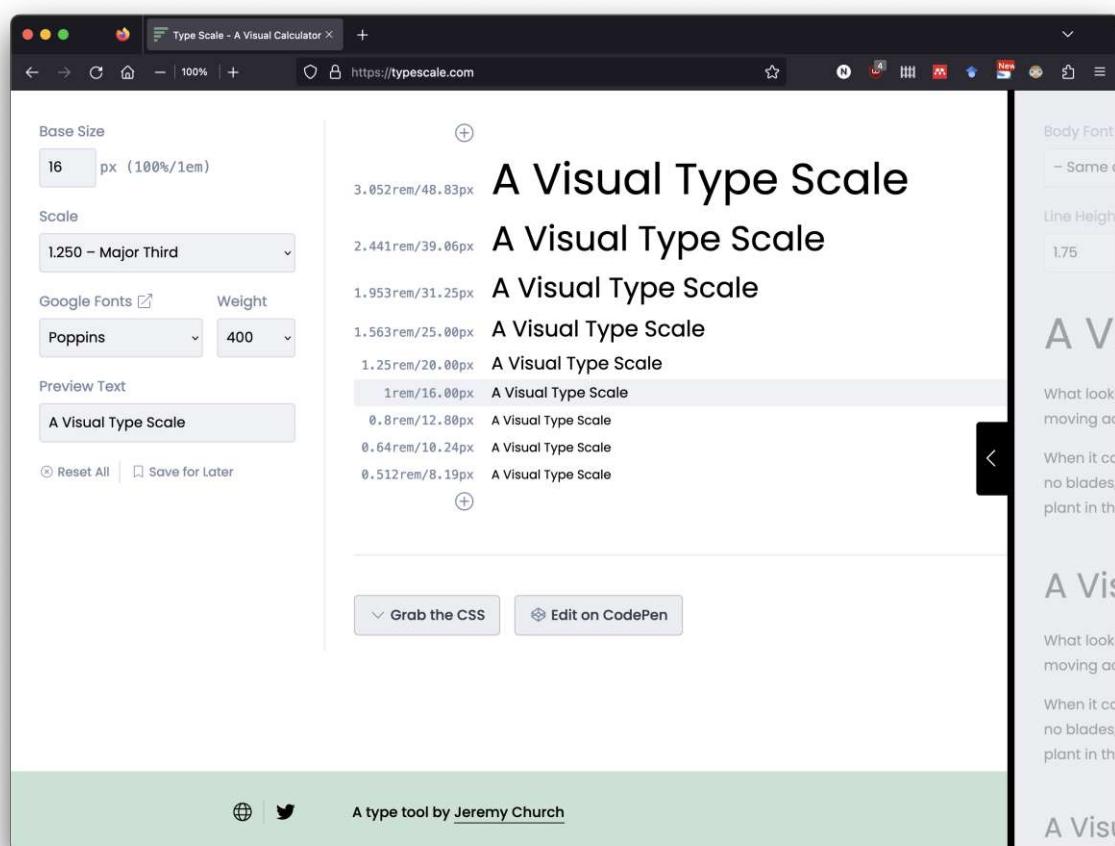


Ekrان Görüntüsü 14. Renk etkisi uygulanmış resim

şifrenizi kullanarak giriş yapabilir, portaldaki binlerce açık dersten ilgilendiğiniz konuları öğrenebilirsiniz.

Ayrıca bu alandaki uzmanların fikirlerini paylaştıkları alanlarda tipografi kurallarını inceleyebilirsiniz. Örneğin <https://medium.com/@tuggrul/her-tasar%C4%B1mc%C4%B1n%C4%B1bilmesi-gereken-tipografik-20-kural-8c89de581ced> adresinde Tuğrul Zengin tarafından hazırlanmış 20 kurallık bir listeye ulaşabilirsiniz. Bu kuralların tasarıma uygulanması ile ilgili videoları (ör: <https://www.youtube.com/watch?v=Uz2RGyK3MyQ>) inceleyebilirsiniz.

Bu alanda farklı tasarımlar gördükçe, uygulamaların başarısında tipografinin önemini kavrayacaksınız. Bu aşamada uygulamanız için bir metin teması oluşturmanız beklenmektedir. Doğrudan kodlamaya geçmeden önce temanızı TypeScale (<https://typescale.com/>) aracı üzerinde oluşturabilirsiniz (Ekran Görüntüsü 15). Bu araç Google Fontlarını kullanır. Bu sayede üitede kullanılan GoogleFonts paketi ile temanızı kolaylıkla üretEBilirsiniz. TypeScale aracı üzerinde görsel olarak oluşturduğunuz temayı, uygulamanızda kodlayınız.



Ekran Görüntüsü 15. TypeScale aracının arayüzü

### 3. DEĞERLENDİR

Uygulama için tema oluşturmak üzere satırlarca kod yazmanız gerekebilir. Bu nedenle oluşturduğunuz temayı öncelikle TypeScle aracı üzerinde değerlendirmeniz önerilmektedir. Bu değerlendirmede aşağıdaki kontrol listesi size yardımcı olabilir:

Ölçüt	Evet	Hayır
Başlıklar ve gövde metni için seçtiğim yazı tipleri birbirleri ile uyumludur. <i>Başlıklar için bir yazı tipi, gövde metni için bir yazı tipi seçtim Başlık ve gövde metni yazı tipleri için doğru renkleri seçtim</i>	<input type="checkbox"/>	<input type="checkbox"/>
Satır hizalamaları ve satır uzunlukları doğru görünmektedir. <i>Masaüstü için satır başına 50-60 karakter, mobil arayüz için satır başına 30-40 karakter</i>	<input type="checkbox"/>	<input type="checkbox"/>
Yazı tipleri ve büyülüklerini çeşitledim <i>Başlıklar, gövde metninden fark edilir düzeyde büyük Birden fazla başlık düzeyi oluşturdum Seçtiğim en küçük yazı boyutu okunabilir düzeyde</i>	<input type="checkbox"/>	<input type="checkbox"/>
Yazılıar artalanlarla karşılık sağlıyor <i>Yazılıar, zeminler üzerinde kolaylıkla okunabiliyor.</i>	<input type="checkbox"/>	<input type="checkbox"/>
Bosluklar okunurluğu destekliyor <i>Farklı düzeylerdeki metinler arasında yeterli boşluk sağladım.</i>	<input type="checkbox"/>	<input type="checkbox"/>

### 4. İLAVE ETKİNLİK

Huy Phan adlı tasarımcı tarafından üretilen mobil uygulama arayüzlerindeki tipografik düzenlemeleri inceleyiniz (<https://dribbble.com/shots/14090962-Masteri-Experimental-Mobile-Layout>). Bu düzenlemeler için uygulamanıza bir tema oluşturunuz. Dribble.com adresinde profesyonel tasarımcılar tarafından üretilen birçok benzer içeriğe ulaşabilirsiniz.

### 5. KAYNAKÇA

Google. (t.y.). <https://docs.flutter.dev> adresinden 18.11.2021 tarihinde erişilmiştir.

<https://medium.com/flutter-community/flutter-layout-cheat-sheet-5363348d037e>

<https://docs.flutter.dev/development/ui/layout>

<https://docs.flutter.dev/development/ui/widgets/layout>

## 4. Hafta: Yayılım Bileşenleri ve Arayüz Tasarımı

---

### Ön Bilgi:

- Temel Dart bilgisi
- Flutter ile bileşen üretme bilgisi
- Flutter ile yayılım düzenleme bilgisi

### Haftanın Kazanımları:

- Statik arayüzler tasarlayabilir
- Flutter uygulamalarının yayılımlarını düzenleyebilir
- Verilen içerikleri etkili bir biçimde sunan arayüzler tasarlayabilir

### Haftanın Amacı:

Bu haftanın amacı, bileşenler ve yayılım mekanizmalarını kullanarak mobil uygulama arayüzlerinin tasarılanmasıdır.

### Kullanılacak Malzemeler:

Visual Studio Code, Chrome internet tarayıcısı.

### Haftanın İşleniği:

**Gözle:** Yayılım bileşenlerinin içine yerleştirilen temel bileşenler ile arayüzlerin tasarılanmasını gözle.

**Uygula:** Kapsayıcı elementleri kullanarak arayüz yayılımlarını düzenle.

**Tasarla:** Verilen içerikleri dikey bir tasarımda sunmak için arayüz tasarla.

**Üret:** Verilen içerikleri dikey bir tasarımda sunan arayüzü üret.

**Değerlendir:** Üretilen dikey arayüzü değerlendir.

# 1. GÖZLE ve UYGULA

Flutter uygulamalarının arayüzlerinin düzenlenmesi için kapsayıcı bileşenler kullanılır. **Container** ve **Center** gibi tek bir bileşeni kapsayabilen bileşenler olduğu gibi, **Row** ve **Flexible** gibi birden fazla bileşeni kapsayabilen elementler de bulunmaktadır. Tek bir bileşeni kapsayabilen bileşenlerin alt elementleri **child** parametresine; birden fazla bileşeni kapsayabilen bileşenlerin alt bileşenleri ise bir liste halinde (**<Widget>List**) **children** parametresine eklenir. Tek elementi kapsayan bileşenler arayüzlerde genellikle elementin kaplayacağı yeri düzenlemek için kullanılır. Çok elementli bileşenlerde ise alt elementlerin arayüzdeki yayılımları ve hizalamaları kontrol edilebilmektedir.

## 1.1. Gözle: Tek Elementli Kapsayıcı Bileşenler

### 1.1.1. Container

**Container** bileşeni kullanıcı arayüzlerinde kutular oluşturmak için kullanılır. Bu kutular kendi başlarına kullanılabileceği gibi, metinler ya da semboller gibi diğer nesnelere artalan ekleme, hızlama, ölçekleme amaçlarıyla da kullanılabilir. Bu nedenle **Container** en çok kullanılan bileşenlerden biridir. Kod 79'da **Container** bileşeni kullanılarak bir simgeye artalan ve kenarlık eklenmiştir. **Container**, kapsayıcı bir bileşen olduğundan içine farklı bir bileşen (Ör: **Text**, **Icon**) eklenmesi gereklidir. Bu bileşen **Container** bileşeninin **child** isimli parametresine atanır. 2 ve 6 numaralı satırlar arasında üretilen beyaz kar simgesi 2 numaralı satırda **Container** bileşeninin **child** parametresine atanmaktadır. Aksi belirtilmediği sürece **Container** bileşeni çocuk bileşeninin boyutları kadar yer kaplar. Fakat **Container** bileşeninin boyutları **width** ve **height** isimli parametreleri yoluyla değiştirilebilir.

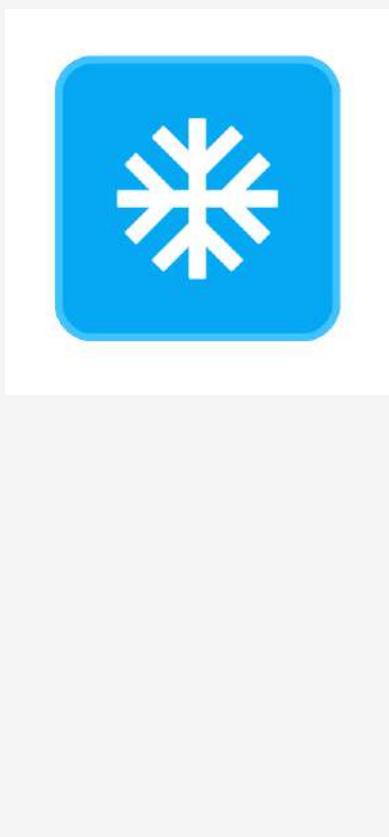
8 numaralı satırda **height** isimli parametresine **100** değeri verilerek bileşenin yüksekliğinin **100** birim boyuttunda olması istenmiştir. Benzer şekilde 7 numaralı satırda **Container** bileşeninin genişliği bulabildiği tüm genişliği kapsayacak şekilde düzenlenmiştir. **double.infinity**, **Double** sınıfı tarafından sağlanan sonsuzluk sabitidir. 7 ve 8 numaralı satırların yorum satırına çevrildiğine dikkat ediniz.

Kullanıcı arayüzü tasarımda nesnelerin yerleştirilmesi kutu modeline göre yapılır. Kutular yan yana ve alt alta getirilerek karmaşık arayüz tasarımları gerçekleştirilebilir. Aksi belirtilmediği sürece kutular yan yana getirilir. Yan yana getirilen kutuların genişliğini kullanılabılır ekran genişliğini aşıyorsa, son kutu otomatik olarak alt satıra alınır. Kutu modelinde bir nesnenin arayüzde ne kadar yer kapladığını dört kutu ile belirler. Öncelikle her bileşen içerik kutusu kadar yer kaplar. Eklenen bir yazı ya da görselin varsayılan boyutları içerik kutusunu oluşturur. Şekil 1'de yer alan görselin boyutları gri ile gösterilen içerik kutusunu oluşturur. İçerik kutusunun boyutları bileşen türüne göre farklı özelliklerce belirlenir. Örneğin metin oluşturmak için kullanılan **Text** bileşenlerinde yazının uzunluğu ve stil özellikleri tarafından belirlenirken, simgelerde **size** parametresi ile belirlenebilir. Kod 79'daki simgenin içerik kutusunun boyutu **size** parametresi ile 50 sanal piksel olacak şekilde belirlenmiştir.

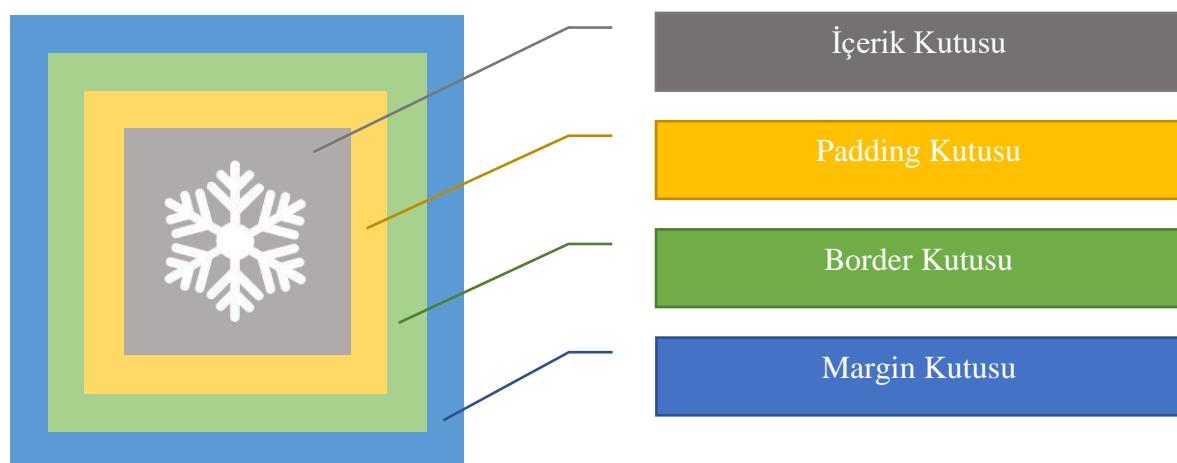
```

1 Container(
2   child: Icon(
3     Icons.ac_unit,
4     color: Colors.white,
5     size: 50,
6   ),
7   //width: double.infinity,
8   //height: 100,
9   margin: EdgeInsets.all(30),
10  padding: EdgeInsets.all(10),
11  decoration: BoxDecoration(
12    color: Colors.lightBlue,
13    border: Border.all(
14      color: Colors.lightBlueAccent,
15      width: 2,
16    ),
17    borderRadius: BorderRadius.circular(10),
18  ),
19 )

```



Kod 79. Container bileşeni ile nesnelere kutu eklemek



Şekil 1. Kutu modeli kavramsal gösterimi

### Dikkat

İçerik kutusu eklenen bileşen tarafından belirlendiğinden zorunlu olarak arayüzde yer kaplar. Fakat içerik kutusu dışındaki kutuların açılması programcinin isteğine bağlıdır. Padding, Border ve Margin kutuları her yönden istenen boyutlarda açılabilir.

Kutu modelinde her bir nesnenin kenarlıkları bulunabilir. Kenarlıklar **Border** sınıfı tarafından oluşturulur ve gerekli noktalara uygulanır. **Border** sınıfı kenarlıkların biçim, renk ve genişliklerini düzenlemek için pek çok özellik sunar. Şekil 1'de kenarlıklar yeşil kutu ile gösterilmiştir. Kod 79'da oluşturulan **Container** bileşeninin biçimsel özellikleri 11-18 numaralı satırlar arasında **decoration** isimli değişkeni yoluyla belirlenmektedir. 13 numaralı satırda **BoxDecoration** sınıfının **border** parametresine bir üretilen bir **Border** nesnesinin atıldığı görülmektedir. Bu nesnede kenarlıkların rengi **lightBlueAccent** (artalan renginin açık hali), genişlikleri de **2** piksel boyutunda olacak şekilde belirlenmiştir. Bu örnekte ayrıca **BoxDecoration** sınıfının **borderRadius** isimli parametresi yoluyla kenarlık yumuşatma uygulanmıştır. 17 numaralı satırda **10** piksellik yarıçap yumuşatması hazırlanarak bu parametreye uygulanmıştır.

Kenarlık ve içerik kutuları arasındaki alan **padding** kutusudur. Bu kutunun boyutu **EdgeInsets** sınıfı tarafından belirlenir. Bu sınıfın **all** yapıçı metodu kullanılarak tüm taraflardan eşit genişlikte kutular açılabilir. Kod 79'da 10 numaralı satırda **Container** bileşeninin boyutları **padding** kutusu büyütülerek her taraftan **10** piksel genişletilmiştir. Bu sayede bir kutunun boyutları **width** ve **height** parametreleri ya da içerik, **padding** ve **border** kutuları yoluyla belirlenebilir.

Kutu modelinin en dışındaki **margin** kutusu ise bir kutunun diğer kutularla arasındaki mesafeyi düzenler. Bir kutunun **margin** kutusunun bittiği noktası diğer kutunun **margin** kutusu başlatılır. Bu yolla sayfa yayılımı düzenlenmiş olur. Kod 79'da 9 numaralı satırda oluşturulan **Container** bileşeninin çevresinde her yönden **30** piksellik boşluk bırakılmıştır.

### 1.1.2. Uygula: Kutu Modeli İncelemesi

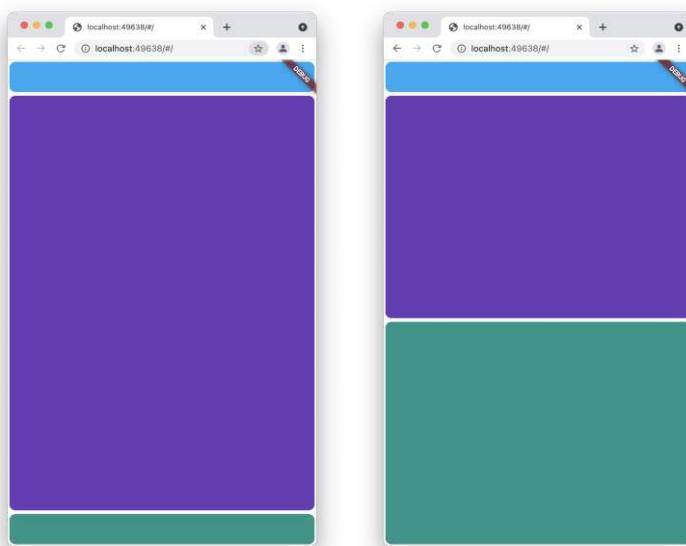
1. Tüm kutular üst üste bindirildiğinden, Kod 79'da oluşturulan Container kutusu arayüzde 134 x 134 piksellik bir alan kaplamaktadır. Bu projeyi çalıştırarak **Widget Inspector** penceresinde kutuların üst üste bindirilmesi yoluyla oluşan boyutlama sistemini inceleyiniz.
2. **mainAxisAlignment** özelliği start olan bir **Row** bileşeni oluşturarak içine iki adet **Container** bileşeni ekleyiniz. Bu kutuların dizilimi üzerinde özellikle **margin** kutusunun etkilerini inceleyiniz.

#### Dikkat

Margin kutusunun etkileri iki kutunun bitişik olduğu senaryolarda geçerlidir. Boşlukların **mainAxisAlignment** özelliği ile değiştirilmesi (Ör: **spaceAround**) halinde **margin** kutusu beklenen etkileri göstermez. Bu nedenle kutuların yayılımının düzenlenmesi için ya kutu modeli ya da hizalama özelliklerini stratejilerinden biri tercih edilmelidir.

### 1.1.3. Expanded Bileşeni

**Expanded** bileşeni arayüzde bulduğu tüm boşluğu kaplar. **Expanded** bileşeleri **Row**, **Column** ya da **Flex** bileşenleri içine yerleştirilirler. Çalışma mantığı **Flexible** bileşeni ile oldukça benzerdir. Birden fazla Expanded bileşeni kullanılması halinde, kalan boşluk bu bileşenler arasında eşit dağıtılır. Ekran Görüntüsü 16'de **Expanded** bileşeni kullanılarak üretilmiş iki adet arayüz bulunmaktadır. İlk arayüzde ilk ve son kutular 50 piksel boyutundadır. Aradaki kutu **Expanded** alanı kaplayacak şekilde ölçeklenmiştir. İkinci arayüzde ise ilk kutu 50 piksel yüksekliğinde bırakılmış; son iki kutu Expanded kullanılarak ölçeklenmiştir.



*Ecran Görüntüsü 16. Expanded bileşeni kullanılarak üretilmiş arayüzler*

```
11 ...
12 children: [
13   Kutu(Colors.lightBlue),
14   Expanded(child: Kutu(Colors.deepPurple)),
15   Expanded(child: Kutu(Colors.teal)),
16 ],
17 ...
```

*Kod 80. Row bileşeni içinde Expanded bileşeninin kullanımı*

Kod 80'de Ekran Görüntüsü 16'deki sağdaki arayüzün kodlarının bir bölümü gösterilmektedir. 13 ve 14 numaralı satırlarda **Column** bileşeni içindeki çocuk elementlerden ikisi **Expanded** bileşeni içine alınarak kalan boşluğun doldurmaları sağlanmıştır.

### 1.1.4. SizedBox Bileşeni

**SizedBox** bileşeni iki amaçla kullanılır. Öncelikle kendisine çocuk olarak verilen bileşeni ölçekleyebilir. Bunun yanında, arayüzde boş kutular üretmek amacıyla da kullanılabilir.

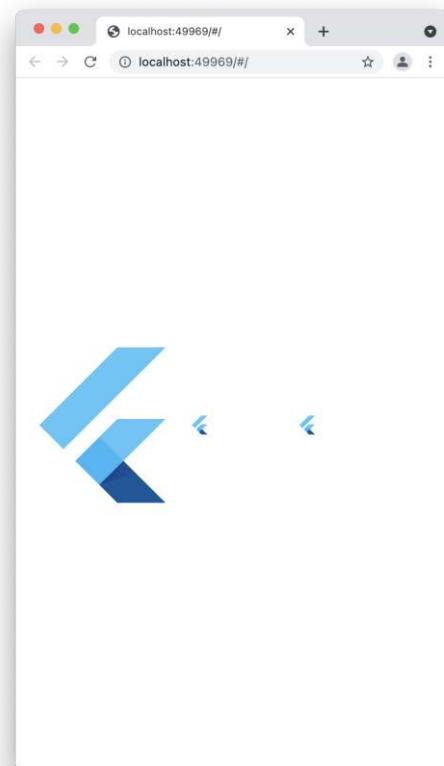
```

13   children: [
14     SizedBox(
15       width: 200,
16       height: 200,
17       child: FlutterLogo(),
18     ),
19     FlutterLogo(),
20     SizedBox(
21       width: 100,
22     ),
23     FlutterLogo(),
24   ]

```

Kod 81. *SizedBox* bileşeni kullanımı

Ekran Görüntüsü 17’de **SizedBox** bileşeni kullanılarak oluşturulmuş basit bir tasarım bulunmaktadır. Kod 81’de Ekran Görüntüsü 17’deki görüntüyü oluşturan kodlar yer almaktadır. Bu görüntüde üç adet **FlutterLogo** bileşeni bir **Row** bileşeninin içine yerleştirilmiştir. Yer kazanmak amacıyla kodun arka kalan rutin bölümleri metne eklenmemiştir. Birinci **FlutterLogo** bileşeni 14-18 numaralı satırlar arasında 200x200 boyutlarında oluşturulan **SizedBox** bileşenine çocuk olarak eklendiğinden, bu bileşen otomatik olarak boyutlandırılmıştır. Ardından eklenen **FlutterLogo** bileşeni varsayılan boyutıyla görüntüye eklendiğinden oldukça küçüktür. 20-22 numaralı satırlar arasında ikinci ve üçüncü **FlutterLogo** bileşenleri arasında 100 piksel genişliğinde bir boşluk oluşturulmuştur. Görüldüğü üzere **SizedBox** bileşeni, arayüzlerde belirli sabit boyutlarda kutular oluşturmak amacıyla kullanılabilmektedir. Dahası, kendisine çocuk olarak eklenen bileşenleri ölçekleyebilmektedir.



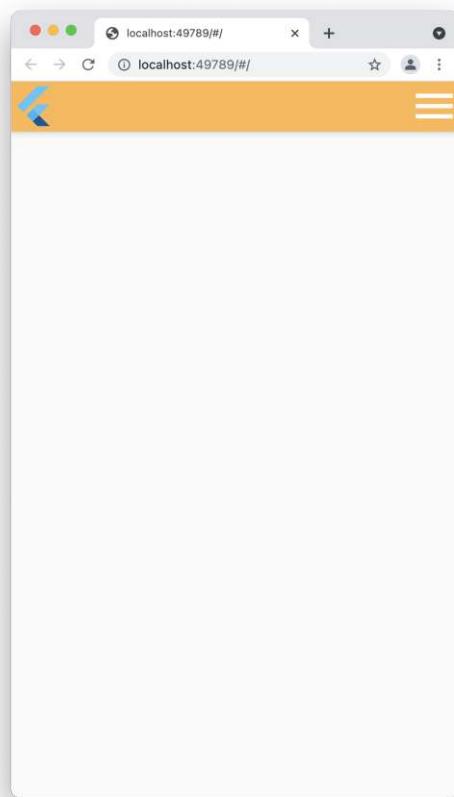
Ecran Görüntüsü 17. *SizedBox* bileşeni kullanımı

### *1.1.5. Spacer Bileşeni*

Bir alandaki nesneleri birbirinden ayırmak için **Spacer** bileşeni kullanılır. **Spacer** bileşeni alandaki boşluğun tümünü kullanacak şekilde tasarlanmıştır. **Spacer** bileşeninin **flex** isimli parametresi yoluyla **flex** alanında üreteceği boşluk alanı belirlenebilir. Fakat bu parametrenin kullanılması zorunlu değildir.

Ekran Görüntüsü 18'de **AppBar** bileşeni içinde iki adet logo bulunmaktadır. Bu logolardan biri kendisi de bir bileşen (**FlutterLogo**) olan Flutter logosudur. Diğer ise **menu** isimli simgedir. Bu iki bileşen **Spacer** bileşeni ile ayrılmıştır.

Kod 82'de, Ekran Görüntüsü 18'deki görüntüyü oluşturan kodlar bulunmaktadır. **import** ifadesi ve giriş fonksiyonu kodları kısa tutmak adına içerikmemiştir. 4 numaralı satırda **MaterialApp** sınıfının **debugShowCheckedModeBanner** isimli parametresine **false** değeri atanmıştır. Bu yolla uygulamanın sağ üstündeki **Debug** etiketi kaldırılmıştır. 5 numaralı satırda bir **Scaffold** bileşeni üretilmiştir. Bu yolla **AppBar** bileşeni için yer açılmıştır. Bu bileşende **flex** alanı üretmek için **flexibleSpace** isimli parametresine bir **Row** atanmıştır. Bu sayede **FlutterLogo**, **Spacer** ve **Icon** bileşenleri oluşturulabilmiştir. **Spacer** bileşeni iki görsel bileşen arasına eklenderek istenilen ayırma etkisi sağlanmıştır. Bu örnekte **Spacer** bileşeninin **flex** parametresi kullanılmamıştır. Bu parametrenin özellikle bir alanda birden fazla **Spacer** kullanılması halinde işlevsel olacağı unutulmamalıdır.



*Ekran Görüntüsü 18. Spacer bileşeni ile bileşenleri ayırmak*

```

1   class MyApp extends StatelessWidget {
2     Widget build(BuildContext context) {
3       return MaterialApp(
4         debugShowCheckedModeBanner: false,
5         home: Scaffold(
6           appBar: AppBar(
7             backgroundColor: Colors.orange[300],
8             flexibleSpace: Row(
9               children: [
10                 FlutterLogo(
11                   size: 50,
12                 ),
13                 Spacer(),
14                 Icon(
15                   Icons.menu,
16                   color: Colors.white,
17                   size: 55,
18                 ),
19               ],
20             ),
21           ),
22         ),
23       );
24     }
25   }

```

Kod 82. *Spacer* bileşeni ile boşluk oluşturmak

## 1.2. Gözle: Çok Elementli Kapsayıcı Bileşenler

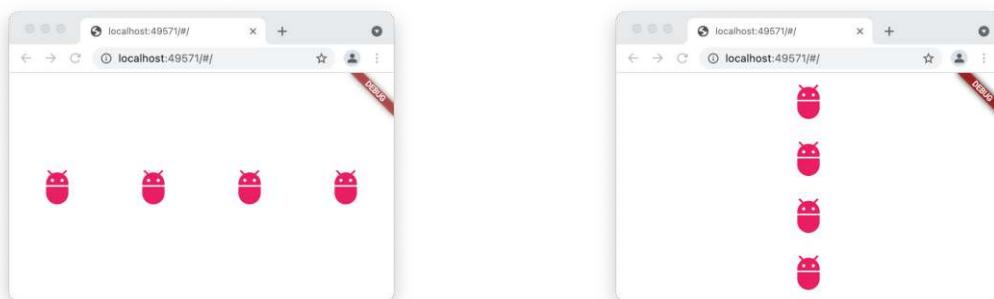
### 1.2.1. Row ve Column

**Row** ve **Column** bileşenleri diğer bileşenleri kapsar ve içindeki bileşenlerin yayılımlarını düzenler. **Row** ve **Column** bileşenlerinin içindeki bileşenler çocuk (**child**), **Row** ve **Column** bileşenlerinin kendileri ise ebeveyn (**parent**) bileşen olarak isimlendirilirler. **Row** bileşenlerinin ana ekseni (**MainAxis**) yataydır. Bu nedenle **Row** bileşenleri içinde yer alan çocuk bileşenler yatay sıralanır. **Column** bileşenlerinin ana ekseni ise dikeydir. Bu sayede **Column** bileşenlerinin çocuk bileşenleri dikey sıralanır. Ekran Görüntüsü 19'da dört adet simgenin bu bileşenler içinde hizalanması gösterilmiştir. Solda simgeler **Row** bileşeni içinde olduğundan simgeler yatay sıralanmıştır; sağda ise bileşenler **Column** bileşeni içinde olduğundan simgeler dikey sıralanmıştır.

```

1 import 'package:flutter/material.dart';
2 void main() => runApp(MyApp());
3 class MyApp extends StatelessWidget {
4     var sembol = Icon(
5         Icons.adb,
6         color: Colors.pink,
7         size: 50,
8     );
9     Widget build(BuildContext context) {
10        return MaterialApp(
11            home: Row(
12                mainAxisAlignment: MainAxisAlignment.spaceAround,
13                crossAxisAlignment: CrossAxisAlignment.center,
14                children: [sembol, sembol, sembol, sembol],
15            ),
16        ),
17    );
18 }
19 }
```

Kod 83. Row bileşeni içindeki bileşenlerin yayılımını düzenlemek



Ekran Görüntüsü 19. Row ve Column bileşenleri

Ekran Görüntüsü 19'daki görüntüyü oluşturmak ve **Row** ve **Column** bileşenlerinin aldığı parametrelerin etkilerini gözlemleyebilmek için Kod 83'teki kodu hazırlayarak sonuçları gözlemleyiniz. Row ve Column arasında geçiş yapmak için 11. numaralı satırda **Row** ifadesinin değiştirilmesi yeterlidir. Bu sayede **Column** ya da **Row** yapıçı metodları çalıştırılmış olacaktır.

Varsayılan olarak **Row** bileşenleri ekran genişliğinin tümünü, **Column** bileşenleri de ekran yüksekliğinin tümünü kaplar. Bu davranışı düzenlemek için **mainAxisSize** isimli parametresinin kullanılır. Bu parametre, **MainAxisSize** sınıfındaki **min** ve **max** sabit değerlerinden birini alır. Varsayılan değeri **MainAxisSize.max** olarak belirnemiştir. **MainAxisSize.min** düzenlemesi yapıldığında bu bileşenler içerdikleri bileşenlerin genişliği

kadar yer kaplayacaktır. 12. satırdaki **MainAxisSize.max** değerini değiştirek etkileri gözlemleyiniz.

#### Dikkat

İsimli parametrelerin küçük harfle başlatıldığına ve alacakları değerlerin aynı isimdeki sınıflardan geldiğine dikkat ediniz. Sınıf isimleri her zaman büyük harfle başlatıldığından yalnızca ilk harfler değişmektedir.

Ör: **mainAxisSize : MainAxisSize.max**

**Row** ve **Column** bileşenleri tüm genişliği kapladığında içeriklerden arta kalan boşluğun kullanımını düzenleyebilir. Bu amaçla **mainAxisAlignment** isimli parametresi kullanılır. Bu parametre **MainAxisAlignment** sınıfı tarafından sağlanan sabit değerler alabilir. Tablo 7'de MainAxisAlignment sınıfı sabitlerinin **Row** bileşeni üzerindeki etkileri belirtilmiştir. Bu değerleri 13 numaralı satırda uygulayarak etkilerini inceleyiniz. **start** ve **end** sabitlerinin etkileri uygulamanın kullandığı dil tarafından belirlenir. **Türkçe** soldan sağa doğru yazıldığından **start** sola, **end** sağa doğru hizalama yapacaktır. Bu etki diğer hizalamalar için de geçerlidir.

#### Dikkat

MainAxisSize.min seçimi yapıldığında Row ya da Column çocuk bileşenlerin toplam boyutu kadar yer kaplayacağından mainAxisAlignment'ların etkileri gözlemlenemez.

Tablo 7. MainAxisAlignment sınıfı sabitlerinin etkileri

Değer	Açıklama
<b>start</b>	Yazım yönüne göre başlangıç (sola) hizalar
<b>end</b>	Yazım yönüne göre son (sağa) hizalar
<b>center</b>	Ortalanmış
<b>spaceBetween</b>	Kalan boşluğu bileşenler arasında eşit dağıt
<b>spaceEvenly</b>	Kalan boşluğu bileşenler çevresinde eşit dağıt
<b>spaceAround</b>	Kalan boşluğu bileşenler çevresinde eşit dağıtırken, ilk ve son bileşenlerin çevresinde yarım boşluk bırak

**Row** ve **Column** bileşenleri çocuk bileşenlerin yayılımlarını ana eksenlerinde gerçekleştirir. Bunun yanında her iki bileşen de üretilen satır ya da sütunun karşı eksendeki (**CrossAxis**) hizalanmasına olanak verir. Bu amaçla **crossAxisAlignment** isimli parametresi kullanılır. Bu parametre hizalama için başlangıç, ortalama ve sona hizalama seçeneklerini sunar. 14 numaralı satırda **CrossAxisAlignment.center** değeri kullanılarak **Row** içinde üretilen satırın dikey olarak ortalanması sağlanmıştır.

Tablo 8. Row bileşeninde CrossAxisAlignmentAlignment değerlerinin etkileri

Değer	Açıklama
<b>start</b>	Satırı üste hizalar
<b>end</b>	Satırı alta hizalar
<b>center</b>	Satırı ortalar

**Row** ve **Column** bileşenlerinin **children** özelliği parametre olarak çocuk bileşenlerinin bir listesini alır. Çocuk bileşenler herhangi bir bileşen (**Widget**) türünde olabilir. Bu örnekte yer kazanmak amacıyla 4-8 numaralı satırlar arasında **sembol** adında bir simge bileşeni üretilmiştir. 15 numaralı satırda 4 adet **sembol** bileşeninden oluşan bir liste **children** parametresine verilerek dört simgeli yapı oluşturulmuştur. Flutter bileşenlerinin yapıcı metotları pek çok parametre aldığından bazı durumlarda bileşenlerin kodun farklı bölmelerinde üretilmesi önerilen bir yaklaşımdır. Hatta her bir Dart dosyasında bir bileşenin üretilmesi; alt bileşenlerin özerk sınıflar halinde farklı bilşenelerde üretilmesi yaygın bir yöntemdir. İlerleyen örneklerde bu yöntem kullanılacaktır.

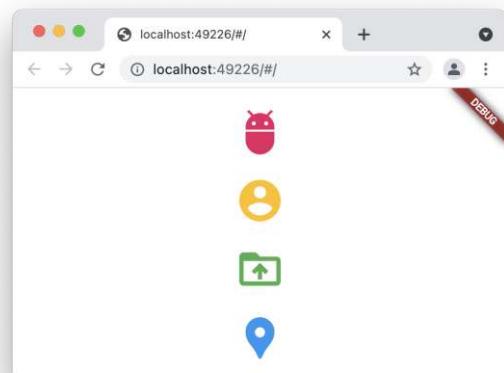
#### 1.2.1.1. Uygula: Column Bileşeni

Kod 83'deki örneği düzenleyerek dört farklı renkteki dört farklı simgeden oluşan bir sütunu dikey olarak ortalanmış ve simgeler arası boşlukları eşit olarak dağıtılmış şekilde oluşturunuz. Bu uygulama için, açıklama amacıyla Ekran Görüntüsü 20 kullanılabilir.

#### 1.2.2. Flexible Bileşeni

**Row** ve **Column** bileşenleri içine yerleştirilen çocuk bileşenlerin boyutları değişken değildir. Fakat gerçek hayat senaryolarında kullanıcıların uygulamaları kullandıkları ekranların boyutları oldukça değişkendir. Örneğin, kullanıcılar bir uygulamayı mobil cihazları yatay ya da dikey tutarak kullanabilir. Benzer şekilde bir uygulama tablet bilgisayar ya da akıllı telefon üzerinde kullanılabilir. Mobil ekranların en-boy oranları arasında dramatik farklar olduğundan, kullanıcı arayüzlerinin bu tür değişimlere tepki verebilmesi beklenir. Bu kavrama elastik tasarım (**Responsive Design**) adı verilir.

**Flexible** bileşeni içine yerleştirilen bileşenlerin boyutlarını elastikleştirir. Bu yolla bir **Container** bileşeninin değişen ekran boyutlarında her zaman ekranın %30'u gibi bir oranını kaplaması sağlanabilir. **Flexible** bileşeninin etkisini görebilmek amacıyla öncelikle boş kutular üreten bir **Kutu** bileşeni hazırlayacağız. Bu yolla hazırlacağımız kodda her seferinde 50x50 boyutlarında kutular oluşturmak için **Container** sınıfının yapıcı metodunu çağırılmak zorunda kalmayacağız. Bu amaçla sadece **Kutu()** çağrısını yapmamız yeterli olacak. Kod 84'da daha önce kullanılmayan bir kod olmadığından bu kodla ilgili açıklama yapılmamıştır. Yalnızca

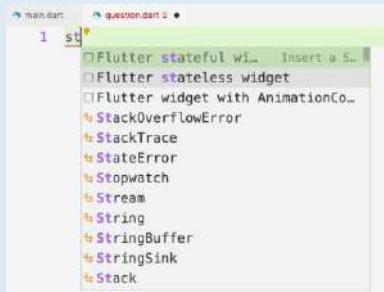


Ekran Görüntüsü 20. 4 farklı simgeden oluşan eşit aralıklı sütun

bu kodların bir sonraki kodörneğinde yazılacak kodlarla aynı dosyada bulunması gerektiğini not ediniz. Bu nedenle satır numaraları 24'ten başlatılmıştır.

### Dikkat

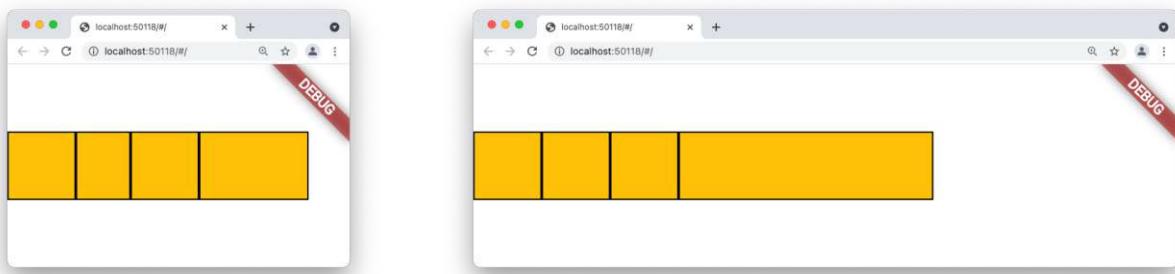
VS Code editörü programcıların iş yüklerini azaltmak için pek çok özelliğe sahiptir. Flutter uygulamaları bileşenler (**Widget**) yoluyla üretildiğinden en çok yazılan kodlardan biri **Widget** sınıflarıdır. VS Code editöründe **st** harfleri basıldığında **Flutter stateful Widget** ve **Flutter stateless Widget** önermeleri yapıldığı görülebilir. Kod 84'da yer alan kodların büyük bir bölümü **Flutter Stateless Widget** seçeneği seçilerek otomatik yazdırılabilir.



```
24 class Kutu extends StatelessWidget {  
25     Widget build(BuildContext context) {  
26         return Container(  
27             width: 50,  
28             height: 50,  
29             decoration: BoxDecoration(  
30                 color: Colors.amber,  
31                 border: Border.all(),  
32             ),  
33         );  
34     }  
35 }
```

Kod 84. Kutu bileşeni kodları

Kod 85'te 10-15 numaralı satırlar arasında bir **Row** bileşeni içinde dört adet kutu oluşturulmaktadır. Bu kutulardan ilki sabit boyutlu kalacak şekilde tanımlanmıştır. Diğer üç kutu **Flexible** bileşeni içine alındığından bu bileşen tarafından ölçeklenir. **Flexible** bileşeninin **child** isimli parametresi ölçeklenecek bileşeni alır. **flex** isimli parametresi ise bileşenin flex alanında kaplayacağı alanı belirler. Aynı alandaki tüm **Flexible** bileşenlerinin **flex** parametrelerindeki değerler toplanarak bir toplam **flex** uzunluğu hesaplanır. Her bir nesnenin **flex** alanında kaplayacağı alan **flex** parametrelerine bakılarak belirlenir. Buna göre, ikinci kutu %20 (1/5), üçüncü kutu %40 (2/5), dördüncü kutu ise %40 (2/5) oranında alan kaplayacaktır. Örneğin, 300 piksellik bir alanda Flexible bileşenine alınmamış sabit boyutlu birinci kutu 50 piksel kaplar. Kalan 250 piksellik alanda Flexible içindeki bileşenler ölçeklenerek yerleştirilir.



*Ekran Görüntüsü 21. Flexible bileşenleri içindeki nesnelerin ölçeklenmesi*

Ekran Görüntüsü 21'de, Kod 85'te yazılan kodun iki farklı tarayıcı boyutundaki etkileri görülmektedir. Kod çalıştırıldığında tarayıcının boyutları el ile değiştirildiğinde yalnızca son kutunun boyutlarının değiştiği gözlenecektir. Bunun nedeni **fit** isimli parametresine verilen değerlerdir.

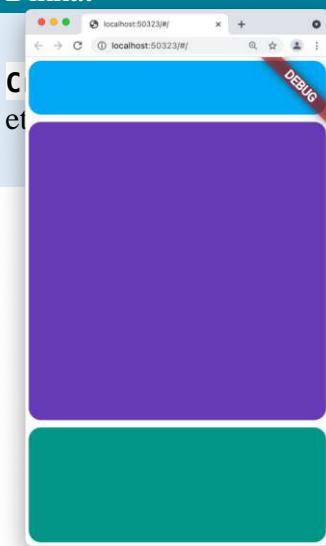
```

1 import 'package:flutter/material.dart';
2 void main() => runApp(MyApp());
3 class MyApp extends StatelessWidget {
4     Widget build(BuildContext context) {
5         return MaterialApp(
6             home: Row(
7                 mainAxisSize: MainAxisSize.max,
8                 mainAxisAlignment: MainAxisAlignment.start,
9                 crossAxisAlignment: CrossAxisAlignment.center,
10                children: [
11                    Kutu(),
12                    Flexible(child: Kutu(), flex: 1, fit: FlexFit.loose),
13                    Flexible(child: Kutu(), flex: 2, fit: FlexFit.loose),
14                    Flexible(child: Kutu(), flex: 2, fit: FlexFit.tight),
15                ],
16            ),
17        );
18    }
19 }
```

*Kod 85. Flexible bileşeni ile kutuları ölçeklemek*

**fit** parametresi **Flexfit** sınıfı tarafından sağlanan iki sabit değerden birini alır. **Flexfit.loose** bileşenin flex alanına girmesini sağlar, fakat ölçeklenmemesine etki etmez. İkonlar gibi nesneler ölçeklendiklerinde bozulabileceklerinden bu tür bileşenler flex alanında bir yer tutacak şekilde **loose** ile işaretlenir. **Flexfit.tight** ise bileşenin flex alanına girmesini ve alanın ebatlarına uyacak şekilde ölçeklenmesini sağlar.

## Dikkat



Çokluğının alabileceği değerlerden biri de **stretch**'dır. Bu değerin 3 numaralı satırda deneyerek gözlemleyiniz.

### 1.2.2.1. Uygula: Flexible ile arayüz tasarımlı

Kutu sınıfını düzenleyerek Kod 85'te yer alan arayüzü tasarlaymentınız.

Bu amaçla **Kutu** sınıfında kenarlıklarını kapatmanız, kenar yumusatmayı açmanız ve kutular arasında boşluk bırakmanız gerekecektir. Ayrıca **Kutu** sınıfını dışarıdan renk kabul edecek şekilde düzenleyerek (Ör: **Kutu(Colors.teal)**) farklı renklerde kutular üretilmesini sağlayabilirsiniz. Örnekteki kutular, **Colors.lightBlue**, **Colors.deepPurple** ve **Colors.teal** renklerine boyanmıştır.

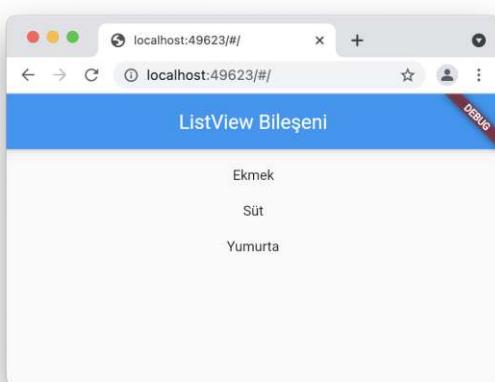
Kutuların boyutlarının ayarlanması için **Flexible** bileşeninin **flex** isimli parametresinin kullanılması ve **fit** isimli parametresine de **Flexfit.tight** değerinin verilmesi gerekecektir.

Kutu genişliklerinin ayarlanması için **width** parametresine **double.infinity** değeri verilebilir

*Ecran Görüntüsü 22. Flexible ile arayüz tasarımlı*

### 1.2.3. ListView Bileşeni

Listeler en çok kullanılan yayılım bileşenleri arasındadır. Listelere eklenen elementler belirtilen kaydırma yönünde hareket ettirilerek küçük ekran alanlarında yüksek miktarda verinin gösterilmesi için kullanılabilir. Liste oluşturmak için **ListView** bileşeni kullanılır. **ListView** bileşeninin **child** isimli parametresine atanmış bileşenler listenin elementleridir.



*Ecran Görüntüsü 23. ListView bileşeni ile oluşturulmuş liste*

boşluğu verilmektedir. 3 numaralı satırda ise **Text**, **Center** ve **Container** bileşenlerinden oluşturulan kompozit bileşenler bir liste halinde **children** isimli parametresine

Kod 86'da Ekran Görüntüsü 23'te görülen liste oluşturulmaktadır. Alan sınırlılıkları nedeniyle **import** ifadesi, **main** fonksiyonu ve uygulama bileşenini oluşturan kodlara yer verilmemiştir. Görülen kodların bir **Scaffold** bileşeninin **body** isimli parametresine aktarılması yeterli olacaktır. **ListView** bileşeninin görevi verilen çocukların dizilimini ve kaydırılabilirliğini sağlamaktr. Elementlerde gösterilen metinlerin stilleri, margin ve paddingler elementler üzerinde gerçekleştirilir. 2 numaralı satırda oluşturulan listeye her yönden 8 piksellik **padding**

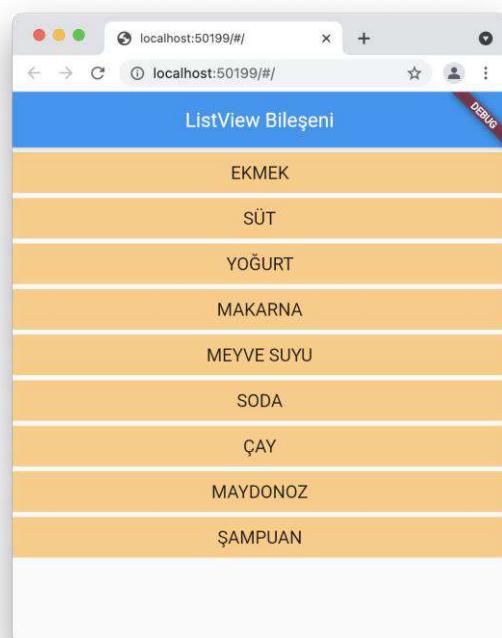
aktarılmaktadır. Bu sayede oluşturulan liste elementlerinde hizalama ve elementler arası boşluklar ayarlanabilmiştir. **ListView** kullanımı oldukça kolay olmakla birlikte, listelerin bu şekilde üretilmesi oldukça zahmetlidir. Bu nedenle **ListView** sınıfı builder ve **separated** isimli iki yapıçı metot sunmaktadır. Bu iki metot arasındaki fark **separated** metodunun listeye istenen özellikle ayrııcı elementler ekleyebilmesidir. Bunun dışında **builder** metodundan farklı olmadığından burada örneklenmeyecektir.

```
1  ListView(
2      padding: EdgeInsets.all(8),
3      children: <Widget>[
4          Container(
5              padding: EdgeInsets.all(10),
6              child: Center(child: Text('Ekmek')),
7          ),
8          Container(
9              padding: EdgeInsets.all(10),
10             child: Center(child: Text('Süt')),
11         ),
12         Container(
13             padding: EdgeInsets.all(10),
14             child: Center(child: Text('Yumurta')),
15         ),
16     ],
17 )
```

Kod 86. *ListView* bileşeni ile liste oluşturmak

Ekran Görüntüsü 24’te 10 adet element içeren bir liste yer almaktadır. Bu uygulamanın kodları Kod 87’de verilmiştir. **builder** yapıcı metodу **itemBuilder** isimli parametresine verilen anonim fonksiyonu **itemCount** isimli parametresinde verilen sayıda çalıştırarak ürettiği elementleri listeye ekler.

Kod 87’de projenün tüm kodları verilmemiştir. Burada verilen kodlar uygulama içinde gerekli yerlere yerleştirilerek proje tamamlanabilir. 6 numaralı satırda tamımlanan liste kodun herhangi bir yerinde yazılabilir. Fakat uygulamanın oluşturulduğu sınıfın başında ya da hemen üzerinde yazılması önerilmektedir. Ekran Görüntüsü 24’teki görüntütü elde etmek için uygulama sınıfında bir **Scaffold** kullanılması, **AppBar** eklenmesi ve **body** isimli parametresine **ListView.builder** metodunun aktarılması gerekmektedir.



*Ecran Görüntüsü 24. ListView bileşeninde stillenmiş elementler*

```

6   var alisverisListem = ["EKMEK", "SÜT", "YOĞURT", "MAKARNA",
7     "MEYVE SUYU", "SODA", "ÇAY", "MAYDONOZ", "ŞAMPUAN"];
8 ...
9   ListView.builder(
10    itemCount: alisverisListem.length,
11    itemBuilder: (BuildContext listem, int index) {
12      return Container(
13        padding: EdgeInsets.all(10),
14        margin: EdgeInsets.fromLTRB(0, 5, 0, 0),
15        decoration: BoxDecoration(
16          color: Colors.orange[200],
17        ),
18        child: Center(
19          child: Text(alisverisListem[index],
20          style: TextStyle(
21            fontFamily: "Roboto",
22            fontSize: 18,
23          ),
24        ),
25      );
26    }
27  )

```

Kod 87. ListView bileşeninin builder yapıcı metodunun kullanımı

Oldukça karmaşık görünse de **ListView.builder** metodunda iki adet paremetre kullanılmıştır. 25 numaralı satırda listeye kaç adet element ekleneceğini belirtmek için **ItemCount** parametresine tanımlanan **alisverisListem** listesinin uzunluğu atanmıştır. Bu sayede **itemBuilder** parametresinde verilen fonksiyon her bir liste elementi için bir kez çalıştırılır. **itemBuilder** isimli parametresine anonim (isimsiz) bir fonksiyon atanmıştır. Bu fonksiyonda **listem** adında bir **BuildContext** parametresi ve **index** adında bir sayıç parametresi verilmiştir. Bu parametrelerin verilmesi zorunludur. Bu fonksiyon bir **Container** bileşeni oluşturarak geri çevirmektedir. 28-32 numaralı satırlar arasında bu **Container** bileşeninin stil özellikleri düzenlenmektedir. 33 numaralı satırda oluşturulan **Center** bileşeni ile oluşturulan **Text** bileşeninin kapsayıç **Container** bileşeni içinde ortaya hizalanması sağlanmaktadır. 34 numaralı satırda **index** üzerinden liste elementlerine ulaşılmaktadır. **index** değişkeninin değeri fonksiyon her çağrıldığında otomatik olarak değiştirilir. Fonksiyon üçüncü kez çağrıldığında **index** değişkeninin değeri otomatik olarak 3 numaralı elementi işaret edebilecek şekilde düzenlenmektedir. 35-37 numaralı satırlar arasında ise oluşturulan **Text** bileşenine uygulanacak stil özellikleri düzenlenmektedir.

### 1.2.3.1. Uygula: Yatay kaydırma listeleri

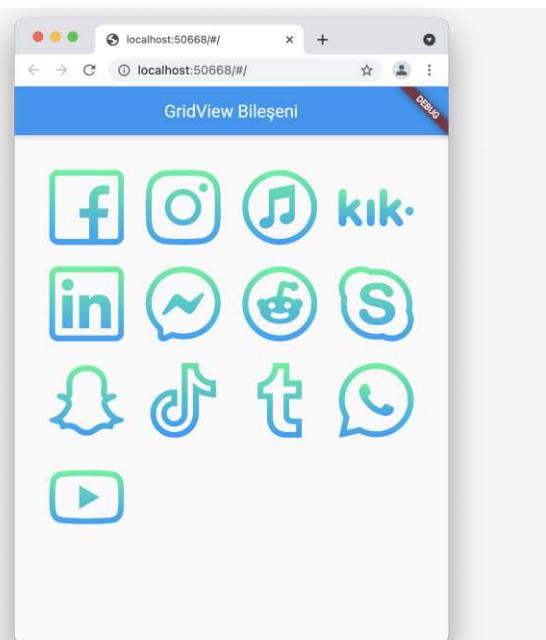
ListView bileşenlerinin varsayılan kaydırma yönü dikeydir. Fakat bu davranış scrollDirection parametresi üzerinden değiştirilebilir. Oluşturacağınız bir listenin yönünü **scrollDirection: Axis.horizontal** düzenlemesi ile yataya çevirerek etkilerini inceleyiniz.

### 1.2.4. GridView Bileşeni

**GridView** bileşeni **ListView** bileşeni ile oldukça benzer. **ListView** bileşenine göre avantajı bir satırda gösterilecek element sayısı ve bu elementler arası boşlukların kolaylıkla düzenlenmesidir. Bu sayede iki boyutlu kaydırılabilir listeler üretilmesine izin verir. Ekran Görüntüsü 25'te 13 adet sosyal medya aracı simgesinin gösterildiği bir ızgara sistemi (**Grid**) gösterilmektedir. Bu ızgara dört sütunlu olacak şekilde düzenlenmiştir. Bu uygulamadaki görsellere <https://www.flaticon.com/packs/social-media-90?k=1637785803829> adresinden erişilmiştir.

```
60 assets:  
61   - img/facebook.png  
62   - img/instagram.png  
63   - img/itunes.png  
64   - img/kik.png  
65   - img/linkedin.png  
66   - img/messenger.png  
67   - img/reddit.png  
68   - img/skype.png  
69   - img/snapchat.png  
70   - img/tik-tok.png  
71   - img/tumblr.png  
72   - img/whatsapp.png  
73   - img/youtube.png
```

Kod 88. Görsellerin projeye varlık olarak eklenmesi



Ecran Görüntüsü 25. GridView bileşeni ile oluşturulan ızgara sistemi

Ekran Görüntüsü 25'teki proje için kullanılan görseller, projeye varlık olarak eklenmiştir. Bu amaçla **pubspec.yaml** dosyasında 60 numaralı satırda açılan **assets:** alanı altında her bir resmin yolu verilmiştir. Resim yollarının TAB ile girintilendiğine ve her yolun tire ile işaretlendiğine dikkat ediniz. Bu yollardan da anlaşılacağı üzere, uygulama kök klasöründe **img** adında bir klasör açılarak **PNG** dosyaları bu klasör içine alınmıştır.

**GridView** bileşenleri **children** isimli parametrelerine bir bileşen (**Widget**) listesi kabul eder. Bu liste yapıçı metot içinde oluşturulabileceği gibi, farklı bir alanda oluşturularak buraya da aktarılabilir. Bu uygulamada bu listeyi oluşturan bir fonksiyon yazılmıştır. Bu fonksiyonun kodları Kod 89'da sunulmuştur.

**olustur** fonksiyonu **<Widget>List** tipinde veri çevirmektedir. Oluşturulan bileşenler bir listeye aktarılıarak bu liste **return** ifadesi ile 47 numaralı satırda geri çevrilmektedir. 25-39 numaralı satırlar arasında kullanılacak varlıklar bir harita içinde isimlerle eşleştirilmektedir.

Örneğin Facebook sosyal ağının adı bu ağın simgesinin yolu ile eşleştirilmektedir. 40 numaralı satırda **Widget** tipinde elementler alan boş bir liste oluşturulmaktadır. Buradaki tip tanımlaması kritiktir. 41-44 numaralı satırlar arasında bir **forEach** döngüsü ile kaynak verileri alınarak bu veriler bir **Container** içine yerleştirilmiş **Image** bileşenine gönderilmektedir. Bu sayede **Container** bileşeni içinde resimler içeren bir kompozit bileşen oluşturulmaktadır. 45 numaralı satırda bu bileşenler geri çevrilecek listeye eklenmektedir. 47 numaralı satırda oluşturulan liste geri çevrilerek fonksiyon sonlanmaktadır.

```
24 List olustur() {  
25   var simgeler = {  
26     "Facebook": "img/facebook.png",  
27     "Instagram": "img/instagram.png",  
28     "iTunes": "img/itunes.png",  
29     "Kik": "img/kik.png",  
30     "LinkedIn": "img/linkedin.png",  
31     "Messenger": "img/messenger.png",  
32     "Reddit": "img/reddit.png",  
33     "Skype": "img/skype.png",  
34     "Snapchat": "img/snapchat.png",  
35     "Tik Tok": "img/tik-tok.png",  
36     "Tumblr": "img/tumblr.png",  
37     "WhatsApp": "img/whatsapp.png",  
38     "YouTube": "img/youtube.png"  
39   };  
40   var simgelerim = <Widget>[];  
41   simgeler.forEach((isim, kaynak) {  
42     var simgem = Container(  
43       child: Image.asset(kaynak),  
44     );  
45     simgelerim.add(simgem);  
46   });  
47   return simgelerim;  
48 }
```

Kod 89. Simgeler listesini oluşturan fonksiyonun kodları

Kod 90'da 12-18 numaralı satırlar arasında **GridView** bileşeni **count** yapıcı metodu yoluyla oluşturulmaktadır. 13 numaralı satırda **padding** isimli parametresi ile nesnenin etrafında 40 piksellik bir boşluk oluşturulmaktadır. Bu bileşen varsayılan olarak dikey yönlü kaydırılmaktadır. Fakat bu davranış **scrollDirection** parametresi ile değiştirilebilir. Bu davranış değiştirildiğinde bileşenin ana eksenin değiştirildiğinden bu ayar önemlidir. 14 ve 15 numaralı satırlarda ana eksen ve ters eksenlerdeki nesneler arası boşluklar 25 piksel olacak şekilde düzenlenmektedir.

```

5   class MyApp extends StatelessWidget {
6       Widget build(BuildContext context) {
7           return MaterialApp(
8               home: Scaffold(
9                   appBar: AppBar(
10                      title: Text("GridView Bileşeni"),
11                  ),
12                  body: GridView.count(
13                      padding: EdgeInsets.all(40),
14                      mainAxisSpacing: 25,
15                      crossAxisSpacing: 25,
16                      crossAxisCount: 4,
17                      children: olustur() as List<Widget>,
18                  ),
19              ),
20          );
21      }
22  }

```

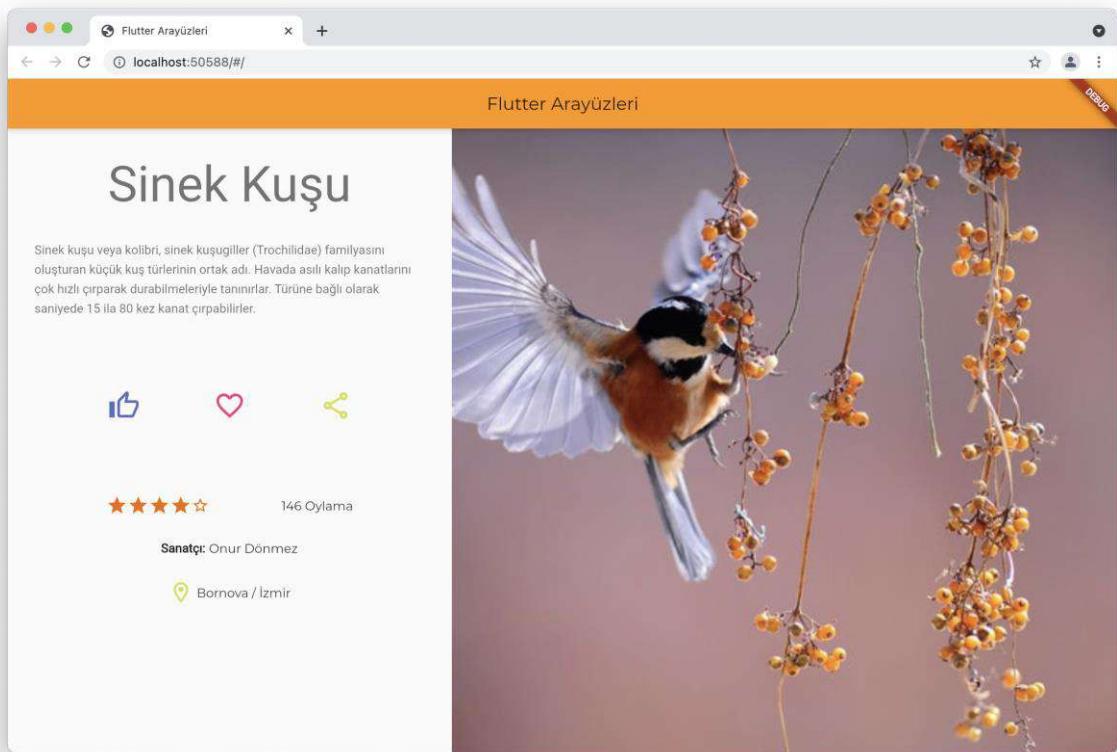
Kod 90. GridView bileşeninin oluşturulması

16 numaralı satırda ızgara sistemindeki sütun sayısı **4** olarak belirlenmiştir. Bileşenin çocuklarını içeren listenin **olustur** fonksiyonu tarafından oluşturulması sağlanmıştır. Bu fonksiyonun çevirdiği listenin bileşenler içeren bir liste olduğu **as** anahtar kelimesi kullanılarak belirtilmiştir.

### 1.3. Gözle: Kapsayıcı Elementlerle Arayüzler Oluşturmak

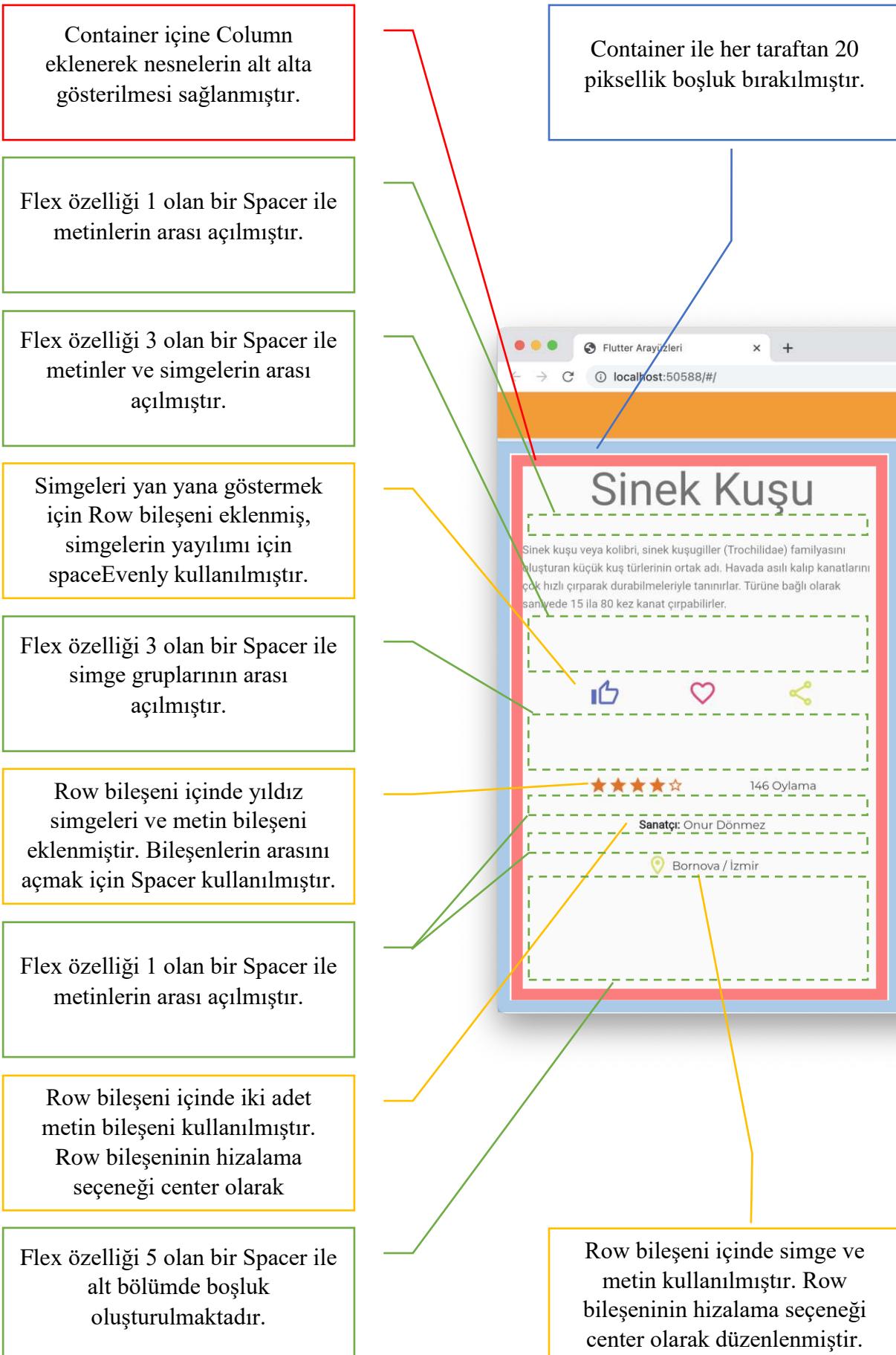
Flutter Ekran Görüntüsü 26'da görülen arayüzler için özel tasarım araçları sunmaz. Bu tür arayüzler, bölümde daha önce bahsedilen kapsayıcı bileşenlerin bir arada kullanılması yoluyla gerçekleştirilir. Örnekte görüntüde temelde iki sütun olduğuna dikkat ediniz. Sağdaki sütunda görsel, soldaki sütunda ise görselle ilgili bilgiler sunulmaktadır. Bu sütunların yan yana gösterilmesi için en üstte bir **Row** bileşeni kullanılmıştır. Ekranı 2/5 ve 3/5 oranlarında bölmek için Row bileşeninin çocukları olarak iki adet **Flexible** bileşeni kullanılmıştır.

Sağdaki **Flexible** elementi içinde bir **Container** elementi oluşturularak bu elementin yüksekliği tüm ekranı kapsayacak şekilde düzenlenmiş; **Boxfit.cover** kullanılarak da resmin tüm kutuyu kaplayacak şekilde gösterilmesi sağlanmıştır.



*Ekran Görüntüsü 26. Örnek yatay ekran arayüzü*

Sol sütündaki görüntüyü sağlamak için birçok bileşen **Flexible** bileşeni altında bir arada kullanılmıştır. Öncelikle içerdeki bileşenlerin çevreden ayrılabilmesi için tüm bileşenler bir **Container** bileşeni içine alınmış ve her taraftan 20 piksellik **padding** alanı bırakılmıştır. Nesneler arası boşlukları bırakmak içinse **Spacer** bileşenleri kullanılmıştır. **Spacer** bileşenlerinin **Flex** özellikleri yoluyla bu bileşenlerin boyutları ayarlanmıştır.



Ecran Görüntüsü 26'da görüldüğü gibi yayılım bileşenleri bir arada kullanılarak karmaşık kullanıcı arayüzleri oluşturulabilmektedir. Flutter, tasarımcıların içerikleri en anlaşılır biçimde aktarabilmesi için pek çok bileşen ve özellik sunmaktadır. Tasarımcıların bu bileşen ve özellik kümelerini iyi analiz ederek en basit, esnek ve anlaşılır arayüzleri oluşturması olasıdır.

#### Proje Kodu

Bu arayüzün kodlarına <https://github.com/onurdonmez/deneyap/blob/main/kuslar.dart> adresinden erişebilirsiniz.

#### *1.3.1. Uygula: Dikey Arayüz Oluşturma*

Ecran Görüntüsü 26'da içerikler yatay bir ekrana sığdırılacak şekilde gösterilmiştir. Fakat mobil uygulamalar genellikle dikey ekranlarda çalışmaktadır. Bu ekranada görünen içerikleri dikey bir ekrana gösterilecek şekilde düzenleyiniz.

## 2. TASARLA ve ÜRET

Hazırlayacağınız bir uygulama için satın alma ekranı tasarlayınız. Örnek satın alma ekranlarını <https://dribbble.com/georgehatzis/collections/1978329-Pricing-Page-Checklist-Design> adresinden inceleyiniz. Bir satın alma ekranında hangi bilgilerin bulunabileceğini arkadaşlarınızla tartışınız.

<https://www.checklist.design/pages/pricing> adresinde yer alan, satın alma ekranlarında bulunabilecek içeriklerin kontrol listesini inceleyiniz.

## 3. DEĞERLENDİR

Tasarladığınız satın alma ekranını değerlendiriniz. Arkadaşlarınızın oluşturduğu arayüzlerle kıyaslayınız. Bu arayüzleri anlaşılırlık (clarity), kısalık (conciseness) ve kullanışlılık (usefulness) boyutlarında değerlendiriniz. Bu terimlerin anlamları ve örnek bir tasarımın şekillenmesi sürecini incelemek için <https://uxplanet.org/ux-writing-how-to-do-it-like-google-with-this-powerful-checklist-e263cc37f5f1> adresini ziyaret ediniz.

Tasarımlarınızı daha etkili hale getirmek için neler yapabileceğinizi tartışınız.

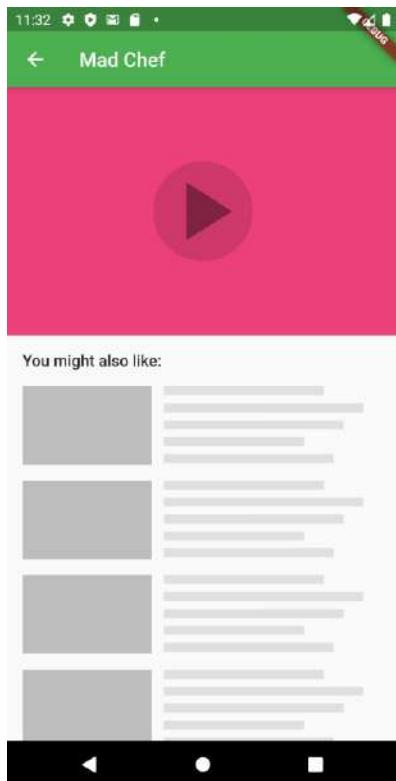
Öğrencinin özdeğerlendirme yapabilmesi için Öz Değerlendirme Formu (<https://forms.gle/kts381jqHWcCah2j6>) uygulanır.

#### Proje Hazırlama

Dersin sonunda üretilen öğrenci projesi için oluşturulan veri yapılarını öğrencilerle birlikte inceleyiniz. Öğrencilerin grup çalışmasına verdikleri katkıları yönelik özdeğerlendirme yapabilmeleri için Grup Çalışması Öz Değerlendirme Formu'nu (<https://forms.gle/FWk3AhU2sFX5heVK8>) uygulayınız.

Bir sonraki haftaya kadar öğrencilerden kullanacakları ekranların arayüzlerini tasarlamalarını isteyiniz.

#### 4. İLAVE ETKİNLİK



Sol tarafta gördüğünüzde benzer örnek bir oynatıcı arayüzü oluşturunuz. Alt tarafta önerilen medyalar için örnek metinler yer

## 5. Hafta: Etkileşimli Uygulamalar

---

### Ön Bilgi:

- Temel Dart bilgisi
- Temel internet kullanım bilgisi
- Flutter ile bileşen üretme bilgisi

### Haftanın Kazanımları:

- Özel bileşen üretebilir
- http istekleri ile veri çekebilir
- Birden fazla sayfadan oluşan uygulamalar üretebilir

### Haftanın Amacı:

Bu haftanın amacı, uzak bir sunucudan veri çekebilen, çok sayfalı ve etkileşimli bir Flutter uygulaması oluşturabilmektir.

### Kullanılacak Malzemeler:

Visual Studio Code, Chrome internet tarayıcısı.

### Haftanın İşlenisi

**Gözle:** Temel bileşenlerin bir araya getirilmesi ile oluşturulan kompozit bileşenlerin hazırlanmasını ve internet üzerinden veri çekilmesini gözle.

**Uygula:** Temel bileşenleri kullanarak kompozit bileşenler üret.

**Tasarla:** Çok sayfalı basit bir mesajlaşma uygulaması tasarla.

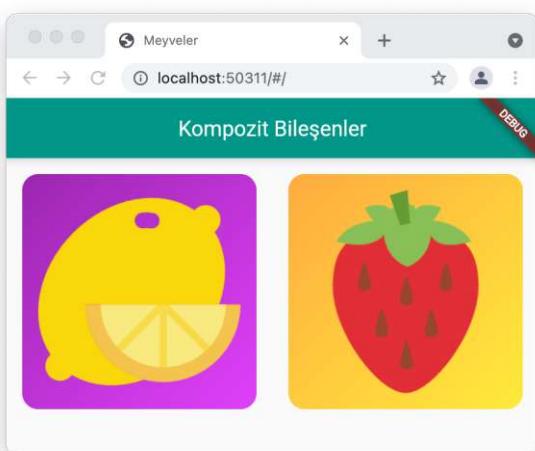
**Üret:** Çok sayfalı mesajlaşma uygulamasını üret.

**Değerlendir:** Ürettiğin mesajlaşma uygulaması arayüzüünü değerlendir.

# 1. GÖZLE VE UYGULA

## 1.1. Gözle: Kompozit Bileşenler

Flutter önemli bir bileşen kütüphanesi sunmaktadır. Bu bileşenler bir araya getirilerek daha karmaşık kompozit bileşenler üretilebilmektedir. Örneğin renkli artalanlı, kenarları yumuşatılmış bir resim kutusu oluşturmak için **Image** ve **Container** bileşenleri birlikte kullanılır. Bu uygulamalar oldukça yaygın olduğundan, programcılar sıkılıkla kendi kompozit bileşenlerini tanımlarlar. Bu sayede, kompozit bilşeneleri uygulamanın her noktasında tekrar tekrar oluşturmak yerine, oluşturdukları kompozit bileşenlerin yapıcı fonksiyonlarına gerekli parametreleri göndererek bileşeni oluştururlar.



Ecran Görüntüsü 27. ResimliKutu kompozit bileşenleri

Ecran Görüntüsü 27'de bir **GridView** bileşeni içine yerleştirilmiş iki adet resimli kutu bulunmaktadır. Kutuların temelinde bir **Container** bileşeni yer almaktadır. Kutuların artalarlarında, sol üst köşeden sağ alt köşeye doğru doğrusan bir gradyan oluşturulmuştur. Ayrıca kutulara projeye eklenen varlıklardan resimler eklenmiştir. Son olarak kutuların kenarları yumuşatılmış; **margin** ve **padding** değerleri 15 piksel olacak şekilde düzenlenmiştir. Bu tür bir ekranı oluşturmak için **GridView** bileşeninin **children** özelliğinde 2 adet **Container** oluşturulabilir. Ardından bu **Container** nesnelerinin her birinde gerekli düzenlemelerin yapılması gerekecektir. Görüldüğü üzere her bir kutu için oldukça yoğun bir kodlama gerekmektedir. İki adet kutu için bu kodlamayı gerçekleştirmek zor olmasa da 20, 30 hatta 100 kutu için böyle bir kodlamayı yapmak oldukça zor olacaktır. Bu iş yükünü azaltmak için kompozit bileşen üretme yolu seçilebilir.

Kod 91'de **ResimKutusu** adlı kompozit bileşenin kodları sunulmuştur. Bu bileşen eklenecek resmin yolu ve gradyandaki iki adet renk değerinden oluşan üç değişken tutmaktadır. 48-50 numaralı satırlar arasında bu değişkenlere bileşen dışından erişimi engellemek için değişkenler yerel olarak tanımlanmıştır (Ör: `_yol`, `_renk1`, `_renk2`). Ayrıca değişkenlerin sabit kalacağı **final** tanımlaması ile belirlenmiştir.

51 numaralı satırda bileşenin değişkenlerine gerekli atamaların gerçekleştirildiği yapıcı fonksiyon tanımlanmıştır. Bu sayede bileşen oluşturulduğunda tüm değişkenlerin değerlerinin verilmiş olması sağlanmıştır.

```

47   class ResimKutusu extends StatelessWidget {
48     final String _yol;
49     final Color _renk;
50     final Color _renk2;
51     ResimKutusu(this._yol, this._renk, this._renk2);
52     @override
53     Widget build(BuildContext context) {
54       return Container(
55         child: Image.asset(_yol),
56         padding: EdgeInsets.all(15),
57         margin: EdgeInsets.all(15),
58         decoration: BoxDecoration(
59           gradient: LinearGradient(
60             colors: [_renk, _renk2],
61             begin: Alignment.topLeft,
62             end: Alignment.bottomRight,
63           ),
64           borderRadius: BorderRadius.circular(15),
65         ),
66       );
67     }
68   }

```

Kod 91. Resim kutusu kompozit bileşeni

**ResimKutusu** kompozit bir bileşen olduğundan bu bileşenin tanımlanması 53-67 numaralı satırlar arasında gerçekleştirilmiştir. Bileşenin temelinde bir **Container** bileşeni yer almaktadır. Bu bileşen oluşturulduğu anda 54 numaralı satırındaki **return** ifadesi ile geri çevrilmektedir. 55 numaralı satırda projeye varlık olarak eklenen resimlerden bir resim nesnesi oluşturularak **Container** bileşeni içine yerleştirilmektedir. 56 ve 57 numaralı satırlarda bileşenin iç ve dış boşlukları 15 piksel olacak şekilde düzenlenmektedir. 58-68 numaralı satırlar arasında ise **Container** bileşeninin art alanının boyanması ve kenarlıklarının yunuşatılması işlemleri gerçekleştirilmektedir. Art alandaki ren geçişik 59-63 numaralı satırlar arasında tanımlanan **LinearGradient** bileşeni ile sağlanmaktadır. Bu bileşenin colors parametresine renk değerleri atanarak renk geçiği sağlanmıştır. Bu geçiş varsayılan olarak yatay eksende gerçekleştirilmektedir. **begin** ve **end** parametrelerinde ise renk geçişinin başlangıç ve bitiş noktaları sol üstten sağ alta olacak şekilde düzenlenmiştir. Görüldüğü üzere **ResimKutusu** bileşeni kendisine parametre olarak verilen değerleri kullanarak bir **Container** nesnesini düzenleyebilmekte ve bu oluşan kompozit bileşeni (**Widget**) geri çevirebilmektedir.

Bu kod sayesinde bir **ResimKutusu** bileşeni oluşturmak için Kod 92'deki çağrı yeterlidir. Bu çağrı ile oluşturulacak **ResimKutusu** kullanıcı arayüzünde istenen yere eklenecektir.

```
ResimKutusu("resimler/meyveler/limon.png", Colors.purple,  
            Colors.purpleAccent)
```

Kod 92. ResimKutusu kompozit bileşenin yapıcı metodunun çağrılmaması

Fakat bu yöntem pratik değildir. Kod 93'te **GridView** gibi birden fazla nesne saklayabilen bir bileşenin **children** parametresine **ResimKutusu** bileşenlerinin eklendiği görülmektedir. Yöntem hem kodları uzatmakta hem de art arda gelecek bileşenlerin yazımını zorlaştırmaktadır.

```
children: [  
  ResimKutusu("resimler/meyveler/limon.png", Colors.purple,  
              Colors.purpleAccent),  
  ResimKutusu("resimler/meyveler/cilek.png", Colors.orangeAccent,  
              Colors.yellow),  
]
```

Kod 93. Bir yayılım bileşenine ResimKutusu bileşenlerinin eklenmesi

Ayrıca bu yazım biçimini dinamik değildir. Sonraki başlıklarda incelenenecek **http** istekleri ile veri çekme gibi yöntemlerle gelen veriler üzerinden, sayısı bilinmeyecek kadar **ResimKutusu** oluşturulmasına izin vermez. Bu nedenle daha dinamik bir yöntem tercih edilmiştir. Bu yöntemin gerçekleştirilmesi için, öncelikle **ResimKutusu** bileşenlerine gönderilebilecek bir nesne modeli oluşturulmuştur. Dart dinamik nesneler üretilmesine izin vermediğinden bir **KutuModeli** sınıfı oluşturulmuştur. Bu sınıf **ResimKutusu** bileşenine gönderilecek parametreleri tutmaktadır. Bu yöntemin kullanılmaması halinde birden fazla listenin kullanılması gibi daha karmaşık yollara başvurulması gerekeceğinden, parametrelerin taşınması için en etkili yol bir model sınıfı olşturmaktr.

```
7  class KutuModeli {  
8    final String _yol;  
9    final Color _renk1;  
10   final Color _renk2;  
11   KutuModeli(this._yol, this._renk1, this._renk2);  
12 }
```

Kod 94. KutuModeli sınıfı

Kod 94'te gösterilen **KutuModeli** sınıfı **ResimKutusu** bileşenine gönderilecek yol ve renk değerlerini tutan 3 değişken içermektedir. Bu değişkenlerin değerleri yapıcı metot üzerinden belirlenmekte, ardından değiştirilememektedir.

Kod 95'te oluşturulacak **ResimKutularına** gönderilecek **KutuModeli** nesnelerini içeren **meyvelerim** listesi oluşturulmaktadır. **List<KutuModeli>** tanımlaması sayesinde Dart bu dizideki elementlerin **KutuModeli** türünde olduğunu anlayabilmektedir.

```

15 final List<KutuModeli> meyvelerim = [
16   KutuModeli("resimler/meyveler/limon.png", Colors.purple,
17     Colors.purpleAccent),
18   KutuModeli("resimler/meyveler/cilek.png", Colors.orangeAccent,
19     Colors.yellow),
20 ];

```

*Kod 95. Oluşturulacak resim kutularının modellerini içeren liste*

Uygulamanın arayüzü oluştururan **MyApp** bileşeninin kodları Kod 96'de verilmiştir. Bu kodların büyük çoğunluğu daha önceki örneklerde incelendiğinden tekrar açıklanmayacaktır. 25-27 numaralı satırlar arasında uygulama temasına yönelik düzenlemeler yapılmaktadır. **MaterialApp** bileşeninin **theme** parametresi üzerinden uygulamanın teması/görünüşü ile ilgili pek çok düzenleme (Ör: renk paleti, koyu/açık tema kullanımı, temel fontlar ...) yapılabilmektedir.

Bu örnekte yalnızca uygulamanın renk paleti bir mavi tonu (teal) şekilde düzenlenmiştir. Bu sayede **AppBar** nesnesi hep gördüğümüz mavi tonunun dışındaki bir renkte gösterilmektedir (Ekran Görüntüsü 27).

32-37 numaralı satırlar arasında bir **GridView** nesnesi oluşturularak **ResimKutuları** bu nesnenin **children** parametresine aktarılmaktadır. 33 numaralı satırda bu ızgara sisteminin 2 sütunlu olması sağlanmıştır. 34 numaralı satırda **children** parametresine **ResimKutusu** bileşenlerinden oluşturulan liste atanmaktadır. Bu listenin oluşturulması için **meyvelerim** adlı listenin **map** metodu kullanılmıştır. Bu metot listedeki her bir elementi içine atanan anonim fonksiyona göndermektedir. Bu metot nesnelere **e** değişkeni üzerinden erişilebilemektedir. 35 numaralı satırda bir **ResimKutusu** bileşeni oluşturulmaktadır. **e** değişkenindeki nesnenin özellikleri **ResimKutusu** bileşeninin yapıçı metoduna gönderilmekteri. Bu sayede **map** metodu her çalıştığında aktif değerlerle yeni bir **ResimKutusu** bileşeni oluşturularak **Iterable** türündeki bir listeye eklenmektedir. Bu nedenle 36 numaralı satırda oluşan liste **toList()** metodu kullanılarak, **children** parametresinin beklediği türde (**List<Widget>**) bir listeye çevrilmektedir.

```

20  class MyApp extends StatelessWidget {
21      @override
22      Widget build(BuildContext context) {
23          return MaterialApp(
24              title: 'Meyveler',
25              theme: ThemeData(
26                  primarySwatch: Colors.teal,
27              ),
28              home: Scaffold(
29                  appBar: AppBar(
30                      title: Text("Kompozit Bileşenler"),
31                  ),
32                  body: GridView.count(
33                      crossAxisCount: 2,
34                      children: meyvelerim.map((e) {
35                          return ResimKutusu(e._yol, e._renk1, e._renk2);
36                      }).toList(),
37                  ),
38              ),
39          );
40      }
41  }

```

Kod 96. Uygulama arayüzünün oluşturulması

#### [1.1.1. Uygula: Kompozit Bileşeninizi Güncelleyin](#)

Bir önceki etkinlikte oluşturduğunuz kompozit bileşeni meyvenin adını alt kısmında ortalanmış biçimde yazacak şekilde güncelleyiniz.

## 1.2. Durum (state) Kavramı

Flutter, diğer programlama dillerinin aksine ekrandaki nesneler üzerinde düzenlemenin nasıl yapılacağını belirten (**imperative**) bir platform değildir. Bu platformlarda gerçekleştirilecek düzenlemeler için (Ör: kullanıcı arayüzüne eklenmiş metin içeriklerini değiştirme) gerekli kodlar programcı tarafından hazırlanır. Örneğin C# programlama dilinde form üzerinde yer alan bir Label nesnesindeki metni değiştirmek için **Label1.Text = "Yeni etiket";** kodu kullanılabilir.

Flutter kullanıcı arayüzüne güncellemek için farklı bir yol benimser. Flutter tanımlayıcı (declarative) bir platformdur. Gerekli görülmesi halinde, kullanıcı arayüzünün tümü ya da bir bölümü alta yatan veri yapısını yansıtacak şekilde sıfırdan üretilerek güncellenebilir. İlk bakışta etkin bir yol gibi görünmese de Flutter uygulamanın gösterileceği her bir karede (Ör: 30 Hz bir ekranda saniyede 30 kare gösterilir.) tüm arayüzü güncelleyebilecek kadar hızlı bir platformdur.

Kullanıcının bir düğmeye dokunması ya da internetten veri çekilmesi gibi durumlarda uygulamadaki veri modelinde bir güncelleme gerçekleşmiş olur. Flutter ortamında bu durum **state** kavramı ile ifade edilir. Flutter uygulamalarında **state** uygulama düzeyinde (**App State**) ya da bileşen düzeyinde (**Widget State**) olabilir. Örneğin bir kullanıcı hesabının giriş formu ile doğrulanması sonucunda uygulama düzeyinde işlemler gerçekleştirebileceğinden (Ör: siparişlerim sayfasının gösterilebilir olması), bu uygulama düzeyinde bir durum güncellemesidir. Kullanıcının bir düğmeye dokunması sonucunda ekranda gösterilen bir bileşenin veri yapısındaki güncelleme (Ör: gösterilen resmin kaynağını değiştirmek) ise bileşen düzeyinde bir durum güncellemesidir.

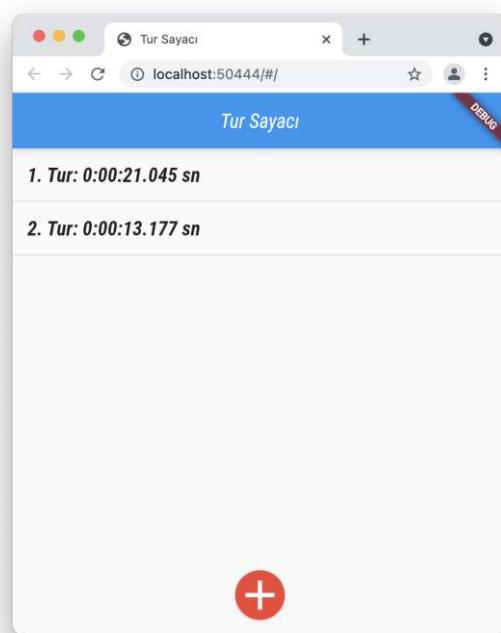
Flutter veri yapısındaki bu güncellemeleri kullanıcı arayüzüne otomatik olarak yansıtır.

Programcının farklılaşan durumlarda arayüzün nasıl görüneceğine ilişkin gerekli tanımlamaları bir kez yapması yeterlidir. Bunun dışında veri yapısındaki değişimleri kontrol eden ve bu değişiklikleri ekrana yansıtan kodları yazmasına gerek yoktur.

Bu mekanizmanın işleyişini gözlemlemek için Ekran Görüntüsü 28'deki basit Tur Sayacı uygulaması hazırlanmıştır. Tur Sayacı yinelenen işlemlerin ne kadar zamanda yapıldığını kaydeden bir uygulamadır. Örneğin bir koşucu, parkurdaki farklı turlarını kaç saniyede attığını bu uygulama ile takip edebilir. Uygulama altta yer alan düğmeye her basıldığında sistem zamanı ile bir önceki turun zamanı arasındaki farkı hesaplayıp **ListView** bileşenine yazmaktadır.

Daha önceki uygulamalarda bileşenlerin içeriği değişmediğinden sürekli **StatelessWidget** sınıfı kullanılmıştır. Bu bileşenler arayüze eklendikten sonra içerikleri değiştirilemez. Örneğin sık sık kullandığımız **Text** bileşeninin içeriği sabittir. Ekranda farklı bir metin göstermek için sıfırdan bir **Text** bileşeni üretilir. İçerikleri değiştirilebilen bileşenler **StatefulWidget** sınıfı üzerinden üretilmektedir. Bu bileşenlerin veri yapıları güncellenebilir. Kendi durum güncellemeleri sayesinde veri yapılarındaki değişimler otomatik olarak kullanıcı arayüzüne yansıtılır.

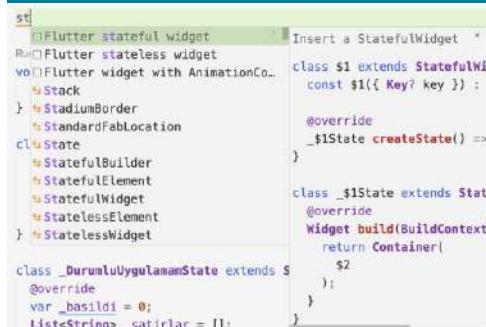
Durumlu bileşenler iki farklı sınıf ile oluşturulur. Bunlardan biri kullanıcı arayüzü, diğer ise bileşenin veri yapısını oluşturur. Kullanıcı arayüzü oluşturulan bileşen **StatefulWidget** sınıfı üzerinden oluşturulur. Bu bileşenin bağlı veri yapısı değiştiğinde bileşen güncellenerek arayüzü günceller. Veri yapısı ise **State** sınıfı üzerinden oluşturulur. Veri yapısı uygulamanın çalıştırıldığı süre boyunca aktif olarak kalır ve içeriği değiştirilmeyez. Burada veri yapısı sadece bileşenin enjekte edilecek verileri ifade etmez. Aslında kendi başına bir bileşen ağacıdır. Fakat bu bileşen ağacı kendi içinde durumunu güncelleyerek **State** üretebilmektedir. Sonuç olarak



Ecran Görüntüsü 28. Tur Sayacı uygulaması arayüzü

**StatefulWidget** bileşeni, **State** sınıfı üzerinden oluşturulan veri yapılarını kullanıcı arayüzüne yansıtır.

### VS Code Kısıyolu



**StatefulWidget** sınıfı ve bağlı sınıfı VS Code tarafından otomatik olarak oluşturulabilir. Bu amaçla düzenleyici içinde **st** tuşlarına basıldığında çıkan yardım panelinden  **StatefulWidget** seçilebilir.

VSCode gerekli sınıfları ekleyerek bağlantıları kurgulayacaktır. Bunun ardından yalnızca **\_state** sınıfı içinde gerekli tanımlamaların yapılması yeterli olacaktır.

Bu iki sınıf arasındaki bağın kurulması için iki işlem gerçekleştirilir. Kullanıcı arayüzüne oluşturan bileşenin adı (**DurumluUygulamam**), veri yapısını oluşturan sınıfta (**\_DurumluUygulamamState**) **State** ifadesinin arkasında belirtilir (**State<DurumluUygulamam>**). Bu ifade Kod 98'deki 16 numaralı satırda görülmektedir. Veri yapısının kullanıcı arayüzü bileşenine bağlanması içinse, kullanıcı arayüzü bileşeni içinde **createState()** metodu çağırılır. Kod 97'de 9-11 numaralı satırlar arasında bu çağrıının yapıldığı görülebilir. Bu iki işlemle kullanıcı arayüzü bileşeni ile veri yapısı sınıfı arasındaki bağ kurulmuş olur. Bu noktadan sonra veri yapısında gerçekleşecek her güncelleme **setState** metodu ile kullanıcı arayüzü bileşenine yansıtılacaktır.

```
1 import "package:flutter/material.dart";
2
3 void main() {
4     runApp(DurumluUygulamam());
5 }
6
7 class DurumluUygulamam extends StatefulWidget {
8     @override
9     _DurumluUygulamamState createState() {
10         return _DurumluUygulamamState();
11     }
12 }
```

Kod 97. DurumluUygulamam sınıfı kodları

Uygulamanın veri yapısının oluşturulduğu kodlar Kod 98'de yer alan **\_DurumluUygulamamState** sınıfı içinde yer almaktadır. Kod 98'de sınıfın tüm kodları bir arada verilmemiştir. Satır numaraları incelendiğinde aradaki kodların silindiği anlaşılabılır. Öncelikle arayüzün **State** ile nasıl oluşturulduğu anlatılacak; ardından her bir bileşenin kodları açıklanacaktır.

```

16   class _DurumluUygulamamState extends State<DurumluUygulamam> {
17
18     var _tur = 0;
19     var _son = DateTime.now();
20     List<String> _satirlar = [];
21
22     void basildi() {
23       setState(() {
24         ...
25       });
26     }
27
28     Widget build(BuildContext context) {
29       return MaterialApp(
30         title: "Tur Sayaci",
31         home: Scaffold(
32           appBar: AppBar(...),
33
34           body: Column(
35             children: [
36               Expanded(
37                 child: ListView.builder(
38                   ...
39
40                   ),
41
42                   );
43
44             ],
45           );
46     }

```

Kod 98. *\_DurumluUygulamamState sınıfı kodları*

**\_DurumluUygulamamState** sınıfı kaçinci turun verilerinin işlendiğini belirten **\_tur**, son turun zamanını içeren **\_son** ve **ListView** bileşenine gönderilecek tur verilerini içeren **\_satirlar** değişkenlerinin tanımlanması ile başlamaktadır. Henüz hiç tur verisi işlenmediğinden **\_tur**

değişkeni **0** ile başlatılmıştır. Uygulama çalışmaya başladığında düğmenin basılmasına kadar geçen süreyi (ilk tur süresi) saklamak için **\_son** değişkenine güncel tarih saklanmıştır. **\_satırlar** ise boş bir liste olarak üretilmiştir.

22-31 numaralı satırlar arasında düğmeye basıldığında uygulama verileri üzerinde gerekli işlemleri yaparak, sonuçlara göre arayüzün tekrar güncellenmesini sağlayan **basıldı** metodu hazırlanmıştır. Bu metot, yeni tur verisini hesaplayarak **\_satırlar** listesine eklemektedir. Metot içinde **setState()** fonksiyonu içine anonim bir fonksiyon yerleştirilerek otomatik olarak çalıştırıldıgına dikkat ediniz. Kullanıcı arayüzüne etki etmesi istenen değişiklikler (ör: Listeye satır eklemek) bu anonim fonksiyon içinde gerçekleştirilir.

33-89 numaralı satırlar arasında uygulamanın kök bileşeni içinde bileşen ağacı tanımlanmaktadır. Bileşen ağacında 48 numaralı satırda tanımlanan **ListView** değişkeni **\_satırlar** dizisindeki değerleri ekrana basmaktadır. 72-81 numaralı satırlar arasında ise yeni tur verilerinin hesaplanması için etkileşim sağlayan düğme oluşturulmaktadır. Bu düğmenin **onPressed** özelliğine **basıldı** metodu bağlanmıştır. Bu sayede düğmeye her basıldığında yeni tur verisi hesaplanarak **\_satırlar** listesine eklenir.

Kod 99'da **\_satırlar** listesindeki zamanları alarak bir **ListView** bileşeni içinde ekrana yazan kodlar yer almaktadır. Listenin satırları **\_satırlar** listesinde tutulduğundan **ListView** bileşeni **builder** yapıcı metodu ile oluşturulmuştur. 49 numaralı satırda listede **\_satırlar** listesindeki kayıt sayısı kadar satır oluşturulacağı belirtilmiştir. **itemBuilder** parametresinde anonim bir fonksiyon tanımlanarak her bir satır için oluşturulacak **Container** ve **Text** bileşenlerinin özellikleri belirtilmiştir. 52-57 numaralı satırlar arasında **Container** bileşeninin alt kenarlığı açık gri renkte boyanmıştır.

58 numaralı satırda **Container** bileşeninin bulunduğu tüm genişliği kaplaması sağlanmıştır. 59 numaralı satırda ise tüm yönlerden 15 piksel **padding** açılmıştır. 60-66 numaralı satırlar arasında **Container** bileşeni içinde bir **Text** bileşeni oluşturularak içine **\_satırlar** dizisindeki bir satır eklenmiştir. Diğer satırlarda bu metnin fontu **Roboto Condensed**, kalın ve italik olacak şekilde düzenlenmiştir.

### Dikkat

Düğmeye fonksiyon bağlanırken yalnızca fonksiyon adının verildiğine, ardından parantezlerin verilmemiğine dikkat ediniz. Bu yolla düğmeye fonksiyonun kodları bağlanmaktadır. Parantezler verildiğinde fonksiyon çalıştırılarak, parametreye fonksiyonun çevirdiği değer atanır.

Yeni tur zamanlarının hesaplanması sağlayan düğmenin kodları Kod 100'de sunulmuştur. Bu düğme bir **Container** içine yerleştirilerek gerekli alan düzenlemelerine olanak verilmiştir. 83 numaralı satırda her yönden 15 piksellik alan bırakıldığına dikkat ediniz. 72-82 numaralı satırlar arasında **ElevatedButton** oluşturulmuştur. Düğmenin kırmızı ve yuvarlak formda olması için 74-77 numaralı satırlar arasındaki stil özellikleri tanımlanmıştır. Düğmenin içinde metin yerine bir artı simgesi kullanılmıştır. Bu simge 78-81 numaralı satırlar arasında tanımlanmıştır. Düğmenin arayüzü değiştiren metodu tetiklemesi 73 numaralı satırındaki **onPressed** parametresine tanımlanan **basıldı** fonksiyonu ile gerçekleştirilmiştir.

```

47   Expanded(
48     child: ListView.builder(
49       itemCount: _satirlar.length,
50       itemBuilder: (BuildContext listem, int index) {
51         return (Container(
52           decoration: const BoxDecoration(
53             border: Border(
54               bottom: BorderSide(
55                 color: Colors.black12,
56                 width: 1,
57               )),
58               width: double.infinity,
59               padding: const EdgeInsets.all(15),
60             child: Text(
61               _satirlar[index],
62               style: const TextStyle(
63                 fontFamily: "Roboto Condensed",
64                 fontSize: 20,
65                 fontStyle: FontStyle.italic,
66                 fontWeight: FontWeight.w700),
67             ),
68           )));
69         })),
70       ),

```

Kod 99. Tur zamanları listesini oluşturan kodlar

Düğmeye basıldığında çağrılan **basıldı** metodunun kodları Kod 101'de sunulmuştur. Bu metot içinde kullanılan **setState** fonksiyonu gerekli hesaplamaların yapılmasından sonra arayüzün güncellenmesi ile ilgili rutinleri otomatik olarak çalıştırır. Bu fonksiyon içine fonksiyon türünde pozisyonel bir değişken verilmesi zorunludur. Bu nedenle 29-31 numaralı satırlar arasında yayılan anonim bir fonksiyon tanımlanmıştır. 22 numaralı satırda **simdi** değişkeni tanımlanarak anlık sistem zamanı bilgisi alınmaktadır. Alınan anlık zaman bilgisi ve son tur arasındaki süre farkı (ilk kez basılması durumunda uygulamanın çalışmaya başlamasından sonra geçen süre) **simdi.difference(\_son)** çağrısı ile hesaplanmıştır. Bu hesaplamanın sonucunun mili saniyeler cinsinden verilmesi için çağrı arkasında **.inMilliseconds** eklenmiştir. Milisaniye cinsinden üretilen bu değerin formatlanması için **Duration** sınıfı yapıcı metodу çağırılmıştır. Bu çağrıda **Duration** sınıfına milisaniye cinsinden değer gönderildiğinin belirtilmesi için **milliseconds** parametresine değer atanmıştır.

```

71 Container(
72   child: ElevatedButton(
73     onPressed: basildi,
74     style: ElevatedButton.styleFrom(
75       shape: const CircleBorder(),
76       primary: Colors.red,
77     ),
78     child: const Icon(
79       Icons.add,
80       size: 50,
81     ),
82   ),
83   padding: const EdgeInsets.all(15),
84 ),

```

Kod 100. Düğme kodları

```

21 void basildi() {
22   var simdi = DateTime.now();
23   var sure = Duration(milliseconds:
24     (simdi.difference(_son).inMilliseconds))
25     .toString()
26     .substring(0, 11);
27   _son = simdi;
28   _tur++;
29   setState(() {
30     _satirlar.add("${_tur}. Tur: $sure sn");
31   })
32 );
33 }

```

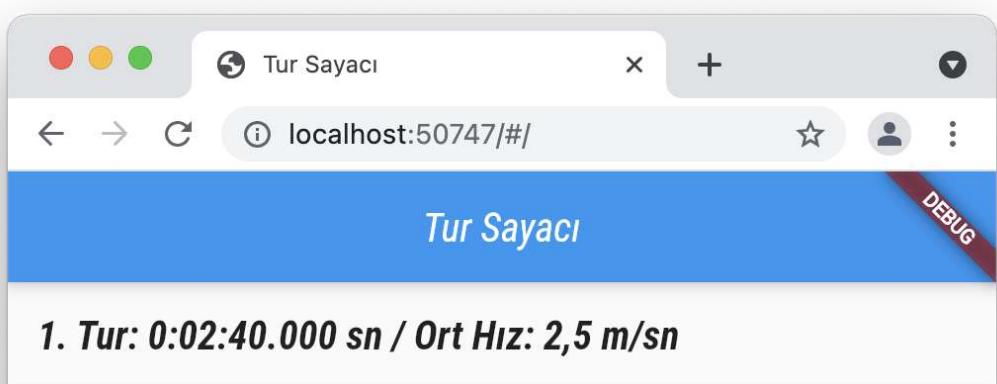
Kod 101. basildi Fonksitonu ve durum güncelleme kodları

**Duration** sınıfı tarafından üretilen değer mikrosaniyeler cinsinden olduğundan (Ör: 00:00:01.000000) okunması oldukça zordur. Bu değerin daha okunur hale getirilmesi için **String**'e çevrilen değerin ilk 11 hanesi alınarak **\_satirlar** listesine aktarılmıştır. 27 numaralı satırda son tur bilgisi güncellenmektedir. 28 numaralı satırda ise atılan tur sayısı bilgisi güncellenmektedir. 30 numaralı satırda hesaplanan değer, tur bilgisi ile birleştirilerek **\_satirlar** listesine eklenmektedir. Bu aktarma işlemi **setState** metodunda gerçekleştirildiğinden arayüz yapılan tanımlamalara göre tekrar oluşturulacaktır. Bunun için **\_satirlar** listesine bağlanmış olan **ListView.builder** metodu tekrar çalıştırılacaktır. Tur

bilgisi ya da zaman bilgisi doğrudan bir değişkene bağlanmadığından bunlar **setState** metodu dışında bırakılmıştır.

#### 1.2.1. Uygula: Tur Sayacı uygulamasında ortalama hız gösterimi

Daha önce oluşturulan Tur Sayacı uygulamasını 400 metre uzunluğundaki bir parkur için her bir turdaki ortalama hızı gösterecek şekilde güncelleyiniz. Bu amaçla metre/saniye ya da kilometre/saat cinsinden hesaplamalar yapabilirsiniz. Örneğin, ortalama 9 kilometre/saat hızında (**9 x 1000 m / 3600 sn**) koşan bir sporcunun hızı metre/saniye cinsinden 2,5'tir. Bu sporcunun 400 metrelik parkuru 160 saniyede (**400m / 2,5 m/sn**) tamamlar. Uygulamanız Ekran Görüntüsü 29'dakine benzer bir sonuç satırı üretebilir.



Ecran Görüntüsü 29. Örnek ortalama hız hesaplaması çıktısı

### 1.3. Gözle: http İstekleri

Günümüzde pek az uygulama kendi üzerindeki verilerle çalışmaktadır. Sosyal ağlar, harita uygulamaları, alışveriş sistemleri gibi uygulamalar düşünüldüğünde bunların çok büyük kısmının sürekli olarak kendileri dışındaki veri kaynaklarından çektiğleri verileri sundukları görülecektir. Bu nedenle mobil uygulamaların kendileri dışındaki kaynaklara erişerek veri elde edebilmeleri oldukça önemli bir yetenektir.

Deneyap Restoran uygulamasında github platformunda tutulan json türündeki veri yapısı ve yine bu platformda tutulan görseller kullanılmaktadır. Bu uygulama için hazırlanmış veri yapısına <https://github.com/onurdonmez/deneyap/blob/main/db.json> adresinden erişilebilir. Öncelikle buradaki veri yapısını inceleyelim.

Json (JavaScript Object Notation) uygulamalar arasında veri transferi için sıkılıkla kullanılan bir notasyondur. Bu yapı Javascript dilinin nesne notasyonunu kullanır. Json dosyaları yalnızca düz yazı içerir. Düz yazılar içinde değişkenler, değerler, listeler ve nesneler kullanılabilir. Listeler köşeli parantezler içinde, nesneler ise küme parantezleri içinde oluşturulur. **String** değerler ve değişken isimleri tırnaklar içinde yazılır. Değişkenlere atanacak değerler, veri türlerine göre yazılır. Örneğin sayısal **16** ya da **Boolean** türündeki **false** değerleri tırnaklar içinde yazılmaz. Değişkenler ve ilişkilendirilen değerler iki nokta : ile birbirinden ayrılır. Json nesne notasyonu olduğundan kökünde küme parantezleri {} bulunur. Kod 102'de 1 numaralı satırda **turlar** adında bir değişken tanımlanmıştır. Bu değişkene bir liste atanmıştır. Bu listenin içeriği 2 ile 15 numaralı satırlar arasında tanımlanmıştır. Bu listedeki ilk nesne 3-8

numaralı satırlar arasında tanımlanmıştır. 9-14 numaralı satırlar arasında tanımlanan ikinci nesne ile ilk nesnenin virgüller birbirinden ayrıldığına dikkat ediniz. Her bir nesnenin kendi için değişken ve değer çiftleri tanımlanmıştır. 4 numaralı satırda ilk nesnenin **id** değişkenine **1** değeri atanmıştır. Bu nesnenin **baslik** değişkenine **String** tipindeki **Balıklar** değeri atanmıştır. Restoranda satılan balık yemeklerini içeren bir liste ise bu nesnenin **yemekler** değişkenine atanmıştır. Son olarak menüde gösterilecek simgenin URL'si **ikon** değişkenine atanmıştır.

```
1  { "turler":  
2    [  
3      {  
4        "id": 1,  
5        "baslik": "Balıklar",  
6        "yemekler": ["Balık Çorbası", "Levrek Izgara", "Balık Kroket"],  
7        "ikon": "https://raw.githubusercontent.com/onurdonmez/deneyap/main/  
8          bolum4/json/resimler/balik.png"  
9      },  
10     {  
11       "id": 2,  
12       "baslik": "Burgerler",  
13       "yemekler": ["Steak Burger", "Mantarlı Burger", "Tavukburger"],  
14       "ikon": "https://raw.githubusercontent.com/onurdonmez/deneyap/main/  
15         bolum4/json/resimler/burger.png"  
16     }  
17   ]  
18 }
```

Kod 102. Deneyap Restoran Uygulaması json veri yapısı

Öncelikle Kod 103'te verilen, menüdeki her bir yemek ürünün görselini oluşturan kodları inceleyelim. Bu amaçla **Tur** adında bir kompozit bileşen üretilmiştir. Bu bileşenin içeriği oluşturulduktan sonra değişmediğinden **StatelessWidget** türündedir. Bu bileşen **\_img** ve **\_baslik** adında iki **String** değişken kullanmaktadır. 23 numaralı satırda bileşenin yapıcısı metodunda bu değişkenlerin belirlendiğine dikkat ediniz. Bu bileşenin temelinde bir **Container** bileşeni yer almaktadır. Bu **Container** içinde **Column** bileşeni sayesinde alt alta gösterilen **Image** ve **Text** bileşenleri bulunmaktadır.

```

20  class Tur extends StatelessWidget {
21      final String _baslik;
22      final String _img;
23      const Tur(this._baslik, this._img);
24
25      @override
26      Widget build(BuildContext context) {
27          return Container(
28              ...
29          );
30      }
31  }

```

Kod 103. Tur sınıfının genel yapısı

Kod 104'te Tur sınıfının arayüzüünü oluşturan bileşenlerin Container içinde yerleşimini düzenleyen kodlar bulunmaktadır. 29 ve 33 numaralı satırlar arasında **Container** bileşeninin artaları boyanmakta, kenarları yumuşatılmakta ve nesnelerin kenarlardan uzaklaştırılması için gerekli boşluklar bırakılmaktadır.

Oluşturulacak **Image** ve **Text** bileşenlerinin alt alta gösterilmesi için child parametresine bir **Column** bileşeni yerleştirilmiştir. Yine gerekli hizalama ve boşlukların sağlanması için **Image** bileşeni **Expanded** bileşeni içine, **Text** bileşeni de farklı bir **Container** bileşeni içine yerleştirilmiştir.

38 numaralı satırda **Image** bileşeninin **network** yapıçı metoduyla uzaktaki bir resim üzerinden görsel oluşturulmaktadır. Bu resmin sağlanan kutu içinde düzgün gösterilebilmesi için resmin kutuya uyum şekli  **BoxFit.scaleDown** olarak belirlenmiştir. **Text** bileşeni ise **Container** bileşeni içine alınarak çevresinde gerekli boşluklar oluşturulmuştur. Gerekli renk ve yazı boyutu düzenlemeleri yapılarak görsel oluşturulmuştur.

Daha önceki örneklerde olduğu gibi kompozit bileşene gönderilecek verileri düzenli tutmak için bir **TurModeli** sınıfı hazırlanmıştır. Bu sınıfın kodları Kod 105'te sunulmuştur. Bu model yalnızca **\_baslik** ve **\_img** değişkenlerini tutmaktadır. Sınıfın yapıçı metodunda bu değişkenlere gerekli atamalar gerçekleştirilmektedir.

Kod 106'da Deneyap Restoranı uygulamasının giriş kodları yer almaktadır. 3 numaralı satırda şu ana kadarki tüm uygulamalarda kullanılan **material.dart** sınıfı projeye eklenmektedir. 1 numaralı satırda **http** isteğin'in gönderilmesi için gerekli **http** sınıfı projeye eklenmektedir. Bu sınıfa **http** ifadesi ile erişebilmek için **as http** kodu kullanılmıştır. 2 numaralı satırda ise gelen Json verisinin nesneye dönüştürülmesi için kullanılacak **convert** kütüphanesi projeye eklenmektedir.

```

28   return Container(
29     padding: const EdgeInsets.all(15),
30     decoration: BoxDecoration(
31       color: Colors.lightBlue,
32       borderRadius: BorderRadius.circular(15),
33     ),
34     child: Column(
35       crossAxisAlignment: CrossAxisAlignment.center,
36       children: [
37         Expanded(
38           child: Image.network(
39             _img,
40             fit: BoxFit.scaleDown,
41           ),
42         ),
43         Container(
44           margin: EdgeInsets.only(top: 15),
45           child: Text(
46             _baslik,
47             style: const TextStyle(
48               fontWeight: FontWeight.bold,
49               fontSize: 20,
50               color: Colors.white,
51             ),
52           ),
53         ),
54       ],
55     ),
56   );

```

*Kod 104. Tur sınıfındaki bileşenlerin yerleşimini düzenleyen kodlar*

```

13   class TurModeli {
14     final String _baslik;
15     final String _img;
16     const TurModeli(this._baslik, this._img);
17   }

```

*Kod 105. TurModeli sınıfı kodları*

```

1 import 'package:http/http.dart' as http;
2 import 'dart:convert';
3 import 'package:flutter/material.dart';
4 void main() {
5     runApp(const MaterialApp(
6         title: "Deneyap Restoranı",
7         home: Menum(),
8     )));
9 }

```

Kod 106. Deneyap Restoranı uygulaması giriş kodları

Kod 106'de 4-8 numaralı satırlar arasında uygulamanın başlangıç noktası olan **main** metodu tanımlanmıştır. Bu metotta **runApp** fonksiyonu içinde yeni bir **MaterialApp** nesnesi oluşturulmuştur. Bu nesnenin başlık özelliği **Deneyap Restoranı** olarak belirlenmiştir. **home** parametresine ise **StatefulWidget** türündeki **Menum()** sınıfının yapıcı metodunu atanmıştır. Bu sayede uygulama başladığında bir **MaterialApp** nesnesi oluşturarak **home** parametresine ise **Menum** sınıfının yapıcı metodunu tarafından çevrilen nesne gönderilecektir.

```

59 class Menum extends StatefulWidget {
60 const Menum({Key? key}) : super(key: key);
61 @override
62     _MenumState createState() {
63         return _MenumState();
64     }
65 }

```

Kod 107. Menum kullanıcı arayüzü sınıfı

Kod 107'de **StatefulWidget** türündeki **Menum** arayüzü sınıfı oluşturulmaktadır. Bu sınıf **createState** metodu ile **\_MenumState()** metodu tarafından üretilerek **State** nesnesinin oluşturulmasını istemektedir. Kullanıcı arayüzünde gösterilecek olan veri yapısı **\_MenumState** sınıfı tarafından oluşturulmaktadır.

Kod 108'de **http** isteğini göndererek, gelen veriler üzerinden uygulamada gösterilen bileşen ağacının veri yapısını oluşturan **\_MenumState** sınıfının kodları bulunmaktadır. 68 numaralı satırda bu sınıf **Menum** sınıf ile ilişkilendirilerek tanımlanmıştır. 69 numaralı satırda **TurModeli** tipindeki elementlerden oluşan **turler** listesi tanımlanmıştır. Bu liste gelen Json verileri ile doldurulacaktır. 70 numaralı satırda ise **http** isteklerinde adres belirlemek için kullanılan **Uri** tipindeki **\_url** bileşeni tanımlanmıştır.

```

68  class _MenumState extends State<Menum> {
69    List<TurModeli> turler = [];
70    final Uri _url = Uri.parse("https://my-json-
server.typicode.com/onurdonmez/deneyap/db");
71    Future<void> menuyuGetir() async {
72      var res = await http.get(_url, headers: {"Accept": "application/json"});
73      List sonuc = json.decode(res.body)['turler'];
74      setState(() {
75        sonuc.forEach((element) {
76          turler.add(TurModeli(element["baslik"], element["ikon"]));
77        });
78      });
79    }
80    @override
81    Widget build(BuildContext context) {
82      ...
83    }
84    @override
85    void initState() {
86      menuyuGetir();
87    }
88  }

```

Kod 108. http isteğiinin gönderilmesi ve Json nesnesinin yorumlanması

81-97 numaralı satırlar arasında bileşen ağacını oluşturan kodlar yer almaktadır. Bu kodlar daha sonra anlatılacaktır. 99-101 numaralı satırlar arasında **initState()** metodu tanımlanmıştır. Bu metot, tanımlanan bileşen bileşen ağacına eklendiği anda çağrılmaktadır. **initState()** tanımlanan bileşenin başlangıç metodу gibi düşünülebilir. Burada çağrılan **menuyuGetir()** metodu ile **http** isteği gönderilmektedir.

Ağ üzerinden veri çekmek, veri tabanlarına yazmak gibi işlemler kullanılan platform dışındaki süreçlerden etkilendiğinden gecikmeler yaşanabilmektedir. Bu gecikmeler nedeniyle başvurulan bir değişkenin içeriği henüz boş olabilmektedir. Bu senaryolarda uygulamalar göçebilmekte ya da yanlış mesajlar gösterebilmektedir. Bu nedenle kullanılan platform dışındaki veri kaynaklarıyla çalışılırken asenkron programlama yapılır.

Flutter asenkron programlama için **Future** sınıfını sağlar. **Future** sınıfı **async** ve **await** anahtar kelimelerini sağlamaktadır. **async** anahtar kelimesi bir fonksiyonun asenkron çalıştığını belirtmek için kullanılır. Bu sayede fonksiyon içinde asenkron işlemler çalıştırılabilecek; uygulama içindeki devam etmekte olan iş akışları aksamayacaktır. Bir işlemin asenkron tamamlanacağını belirtmek için bu işlemin önüne **await** anahtar kelimesi eklenir. **await** anahtar kelimesi yalnızca asenkron fonksiyonlar içinde çalıştırılabilmektedir.

72 numaralı satırda **http** sınıfının **get** metodu kullanılarak Json nesnesi uygulamaya yüklenmeye çalışılmaktadır. Çekilen Json nesnesi **res** değişkenine atanmaktadır. Bu satırda **await** anahtar kelimesinin kullanıldığına dikkat ediniz. Bu anahtar kelime sayesinde, **get** metodu başarıyla tamamlanana kadar beklenecek; çekilen **Json** verileri **res** değişkenine atanacaktır. Asenkron tamamlanması beklenen fonksiyonların önüne **await** anahtar kelimesi eklenerek uygulamada karşılaşılabilen hatalar engellenmiş olur.

73 numaralı satırda **convert** sınıfı tarafından sağlanan **json.decode** metodu kullanılmıştır. Bu metot, **http.get** metodu ile çekilen **String** tipindeki verilerin nesneye dönüştürülmesi için kullanılır. Liste tipindeki **sonuc** isimli değişkenine bu nesnenin **türler** özelliğine bağlanmış olan parametre eklenmiştir. Kod 102'de **türler** parametresine bir nesneler dizisi eklendiğini hatırlayınız. 74-78 numaralı satırlar arasında arayüzde güncelleme yapılacak veriler işlendiğinden **setState** metodu kullanılmıştır. Bu metot içinde **türler** isimli listeye **TurModeli** tipindeki elementler eklenmektedir. Bu elementlerin içeriği, Json verisinden getirilip **sonuc** değişkenine atanmış veriler üzerinden doldurulmaktadır. Bu amaçla **sonuc** listesi üzerinde **forEach** metodu kullanılmıştır. **forEach** metotu içinde **element** değişkeni **print** metodu ile yazdırılırsa Kod 109'ukine benzer bir çıktı oluşacaktır. Buradan anlaşılacağı üzere, **forEach** metodunun her dönüşünde **türler** listesine her seferinde bir türün başlığı ve simgesini içeren bir **TurModeli** nesnesi gönderilmektedir.

```
{  
    id: 11,  
    baslik: Tavuklar,  
    yemekler: [Piliç Graten, Tavuk Sote, Kızarmış Piliç],  
    ikon:  
https://raw.githubusercontent.com/onurdonmez/deneyap/main/bolum4/json/resimler/tavuk.png  
}
```

Kod 109. Json verisinden getirilen bir nesnenin içeriği

Kod 110'da **türler** listesindeki nesnelerin bir **GridView** içinde gösterildiği kodlar yer almaktadır. **GridView** bileşeni **count** yapıçı metodu ile çağrılmıştır. 90 numaralı satırda bileşenin iç çerçevesindeki boşluklar 30 piksel olacak şekilde ayarlanmıştır. 91 ve 92 numaralı satırlarda ise yatay ve dikey boşluklar 20 piksel olacak şekilde ayarlanmıştır. 93 numaralı satırda ızgara sisteminde 2 sütun olması sağlanmıştır. 94 numaralı satırda ise **türler** listesi üzerinden oluşturulan **Tur** tipindeki bileşenler ızgara sistemine eklenmektedir.

```

83  @override
84  Widget build(BuildContext context) {
85      return Scaffold(
86          appBar: AppBar(
87              title: const Text("Deneyap Restoran"),
88          ),
89          body: GridView.count(
90              padding: const EdgeInsets.all(30),
91              mainAxisSpacing: 20,
92              crossAxisSpacing: 20,
93              crossAxisCount: 2,
94              children: [...turler.map((e) => (Tur(e._baslik,e._img))).toList()],
95          ),
96      );
97  }

```

Kod 110. turler Listesindeki nesnelerin izgara sisteminde gösterilmesi

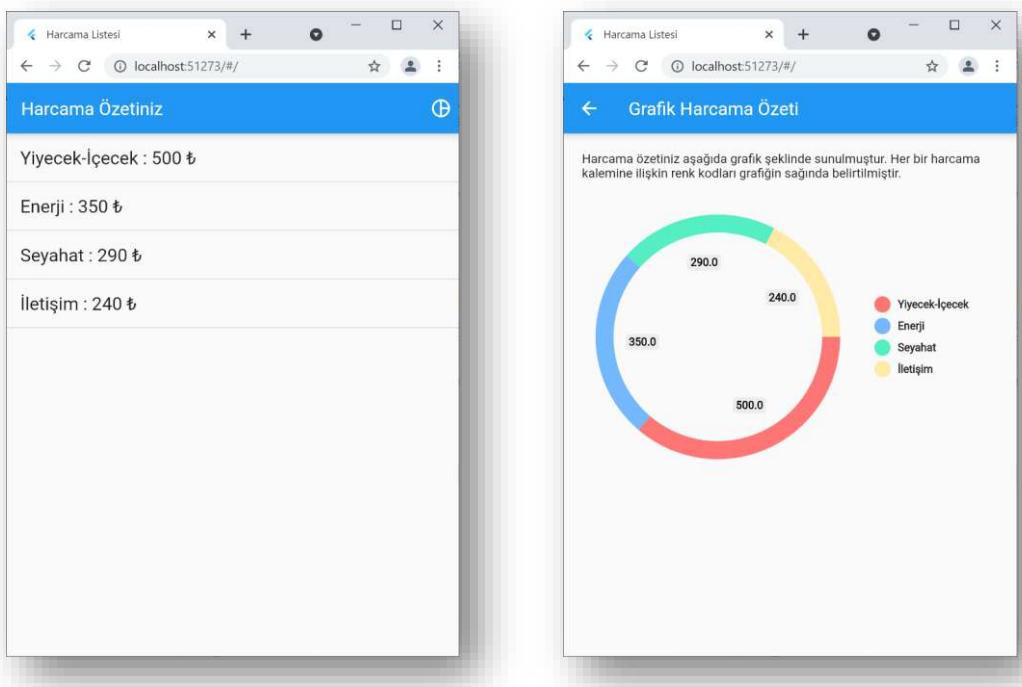
#### 1.4. Gözle: Çok Sayfalı Uygulamalar

Pek çok mobil uygulama farklı amaçlar için üretilmiş birden fazla sayfadan oluşmaktadır. Örneğin bir sayfa kullanıcı girişi için üretilirken, farklı bir sayfada kullanıcının hesap düzenlemeleri yapılabilir, diğer bir sayfda ise kullanıcı kendisine gelen mesajları görüntüleyebilir.

Flutter uygulamalarında bu sayfalara rota (**route**) adı verilir. Kendi başına çalışan her bir sayfa bir rotadır. Flutter bu rotalar arasında gezinim sağlar. Rotalar arası gezinimi incelemek için hayali bir banka uygulaması gerçekleştireceğiz. Uygulamanın arayüzü iki resimle gösterilmiştir. Bu uygulamada kullanıcının aylık harcamalarının dökümünü liste (sol) ya da grafik halinde görülebilmektedir. Harcama Özeti ekranında sağ üstteki grafik simgeseye basıldığında Grafik Harcama Özeti ekranına geçilmektedir. Grafik Harcama Özeti ekranında ise sol üstteki geri simgesine basıldığında Harcama Özeti ekranına geri dönülmektedir.

Bu uygulamada gösterilen pasta garfiğinin oluşturulması için gerekli araçlar Flutter'ın resmi kütüphaneleri arasında yer almaz. Fakat dünyanın çeşitli ülkelerindeki geliştiriciler Flutter uygulamalarında kullanılabilecek araçlar üreterek ücretsiz yayılmaktadır. Bu tür projeleri <https://pub.dev> adresinde bulunabilir. Bu adres Dart ve Flutter uygulamalarında kullanılabilecek paketlerin barındırıldığı bir proje ambarıdır. Ambar Google tarafından desteklenmektedir. Bu uygulamamızda kullanacağımız **pie\_chart** paketi [https://pub.dev/packages/pie\\_chart](https://pub.dev/packages/pie_chart) adresinde sunulmaktadır. Seçilen bir paketin nasıl kullanılabileceği ve bu paketin üretebileceği çıktılar proje sayfalarında İngilizce dilinde anlatılmaktadır. **pie\_chart** paketini kullanabilmek için öncelikle **pubspec.yaml** sayfasında dependencies bölümünde paketin belirtilmesi gerekmektedir. **pie\_chart** ifadesi **pubspec.yaml** belgesinde 32 numaralı satırda eklenmiştir. Paketler birden fazla versiyonda sunulmaktadır. Bu belgenin hazırlandığı dönemde (Ocak 2022) **pie\_chart** projesi 5.10

versiyonundadır. Bir paketin ismi verildikten sonra hangi versiyonun eklenmesi gerektiği iki noktadan sonra belirtilebilir (ör: **pie\_chart: 3.2**). Fakat paketler güncellendikçe yeni yetenekler kazanmakta ve olası hataları düzeltilmektedir. Bu nedenle olası en son versiyonun kullanılması önerilen bir yaklaşımındır. Bu nedenle Kod 111'da belirli bir versiyon numarası yerine aslında yorum satırı olan **#latest version** ifadesi bırakılmıştır.



*Ekrان Görüntüsü 30. pie\_chart eklentisi ile oluşturulan pasta grafiği*

```
29 dependencies:  
30   flutter:  
31     sdk: flutter  
32   pie_chart: #latest version
```

*Kod 111. pubspec.yaml dosyasında pie\_chart paketinin eklenmesi*

### Dikkat

Nadir durumlarda bir paket (Ör: pie\_chart), diğer bir paketin (ör: bir excel dosyasından veri çeken paket) belirli bir versiyonunun kullanılmasını gerektirebilir. Bu hallerde son versiyon yerine, gereksinim duyan paketin sayfasında belirtilen versiyon numarası kullanılır.

```

1 import "package:flutter/material.dart";
2 import 'package:pie_chart/pie_chart.dart';
3 Map<String, double> harcamalar = {
4     "Yiyecek-İçecek": 500,
5     "Enerji": 350,
6     "Seyahat": 290,
7     "İletişim": 240,
8 };
9 var anahtarlar = harcamalar.keys.toList();
10 void main() {
11     runApp(const MaterialApp(
12         debugShowCheckedModeBanner: false,
13         title: "Harcama Listesi",
14         home: GirisSayfam(),
15     ));
16 }
17 ...
18 class GirisSayfam extends StatelessWidget { ... }
19 ...
20 class GrafikSayfam extends StatelessWidget { ... }

```

Kod 112. Çok sayfalı uygulamanın giriş kodları

Kod 112'de çok sayfalı uygulamanın giriş kodları görülmektedir. 2 numaralı satırda **pie\_chart** paketi projeye eklenmektedir. 3-8 numaralı satırlar arasında yapılan harcamaların alanlara göre tutulduğu bir harita yapısı oluşturulmuştur. Bu yapı **pie\_chart** paketi tarafından doğrudan kullanılabildiğinden harita yapısı tercih edilmiştir. Harita yapılarındaki değerlere ulaşmak için anahtarlar kullanıldığından 9 numaralı satırda daha sonra sayfalarda kullanmak üzere haritadaki anahtarlar **anahtarlar** isimli listeye kaydedilmiştir. 10-16 numaralı satırlar arasında uygulamanın **main** metodu hazırlanmıştır. Daha önceki uygulamalarda olduğu gibi **MaterialApp** bileşeni **runApp** fonksiyonu içinde üretilmiş; **home** parametresine oluşturulan rotalardan biri atanmıştır. Uygulamada **GirisSayfam** ve **GrafikSayfam** isimleri verilmiş iki adet rota bulunmaktadır. Bu rotaların çağrıldıklarında oluşturulan **Scaffold** bileşenlerini çevirmektedir. **runApp** metodunda **MaterialApp** bileşeninin **home** parametresine **GirisSayfam** bileşeni atanarak uygulamanın bu sayfa ile başlaması sağlanmıştır.

```

22 class GirisSayfam extends StatelessWidget {
23   const GirisSayfam({Key? key}) : super(key: key);
24   @override
25   Widget build(BuildContext context) {
26     return Scaffold(
27       appBar: AppBar(
28         title: const Text("Harcama Özeti"),
29         actions: <Widget>[
30           IconButton(
31             icon: const Icon(Icons.pie_chart_outline),
32             tooltip: 'Grafik Göster',
33             onPressed: () {
34               Navigator.push(context,
35               MaterialPageRoute(
36                 builder: (context) => const GrafikSayfam()),
37               );
38             },
39           ),
40         ],
41       ),
42       body: ListView.builder( ... ),
43     );
44   }
45 }

```

Kod 113. GirisSayfasi bileşeninde başlık çubuğu ve yönlendirme kodları

Kod 113'te **GirisSayfasi** rotasının kodları görülmektedir. 42-58 numaralı satırlar arasında haritadaki değerlerin **ListView** bileşenine eklendiği kodlar yer kısıtları nedeniyle koddan çıkarılmıştır. Bu kodlar daha sonra sunulacaktır. 27-41 numaralı satırlar arasında rotanın başlık çubuğu üretilerek **appBar** parametresine atanmıştır. 28 numaralı satırda **title** parametresinde çubukta görülecek metin düzenlenmiştir. 29 numaralı satırda kullanılan **actions** parametresi başlık çubuğu sağına yerleştirilecek bileşenleri tutmaktadır. Burada birden fazla bileşen tutulabileceğinden bu parametre **Widget** tipindeki elementlerden oluşan bir liste **<Widget>[]** kabul eder. Uygulamada tek bir düğme kullanılsa da yine de liste içinde verilmesi zorunludur. 30-39 numaralı satırlar arasında bir **IconButton** bileşeni oluşturulmuştur.

**IconButton** bileşenleri simgeleri düğme olarak kullanmak için kullanılır. 31 numaralı satırda düğme olarak gösterilecek simge, **IconButton** bileşeninin **icon** parametresine atanmaktadır. Bu bileşenin üzerine fare getirildiğinde gösterilecek yardım metni 32 numaralı satırda **tooltip** parametresine eklenmiştir. 33 numaralı satırda bu düğmeye basılması halinde çalışacak anonim fonksiyon **onPressed** parametresine atanmıştır.

**Navigator** mobil uygulamada gösterilen rotaların yiğinini tutan bir bileşendir. Gösterilecek yeni sayfalar yiğinin üzerine eklenirken, önceki sayfalara erişmek için yiğinin tepesindeki sayfalar çıkarılır. Bu amaçlar **Navigator** sınıfının **push** ve **pop** metotları kullanılır. **Navigator.push** metodu yiğin üzerine yeni bir sayfa ekler. **Navigator.pop** metodu ise yiğinin en tepesindeki sayfayı çıkarır.

```
42  ListView.builder(
43    itemCount: anahtarlar.length,
44    itemBuilder: (BuildContext listem, int index)
45    {
46      var anahtarim = anahtarlar[index];
47      var degerim = harcamalar[anahtarim];
48      return Container(
49        decoration: const BoxDecoration(
50          border: Border(
51            bottom: BorderSide(
52              color: Colors.black12,
53              width: 1,
54            )),
55            width: double.infinity,
56            padding: const EdgeInsets.all(15),
57            child: Text(
58              "$anahtarim : $degerim ₺",
59              style: const TextStyle(fontSize: 20),
60            ),
61          );
62    }
}
```

Kod 114. ListView bileşeninin içini dolduran kodlar

Kod 114'te 42-62 numaralı satırlar arasında **ListView** bileşenine harcamaları ekleyen kodlar sunulmuştur. 43 numaralı satırda **ListView** bileşenine eklenecek madde sayısı, **anahtarlar** listesindeki element sayısı kadar olacak şekilde belirlenmiştir. 44 numaralı satırda **itemBuilder** parametresine verilen anonim fonksiyon ile **Container** bileşenleri oluşturularak **ListView** nesnesine eklenmektedir. Bu kodlarda kullanılan mekanizma ve bileşenler daha önceden sunulduğundan içerik tekrar anlatılmamıştır.

Kod 115'te **StatelessWidget** türündeki **GrafikSayfam** bileşeninin kodları sunulmuştur. Bu bileşen tüm sayfayı kapladığından bir **Scaffold** bileşeni üzerinde inşa edilmiştir. 69-71 numaralı satırlar arasında oluşturulan **AppBar** bileşeni üzerinde bir geri düğmesi tanımlanmamıştır. Flutter bu tanımlamanın yapılmaması halinde otomatik bir geri düğmesi oluşturarak gerekli alana yerleştirmektedir. Bir önceki sayfaya geçmek için gerekli kodlar (**Navigator.pop()**) bu düğmeye otomatik olarak eklenir.

```

64  class GrafikSayfam extends StatelessWidget {
65      const GrafikSayfam({Key? key}) : super(key: key);
66
67      @override
68      Widget build(BuildContext context) {
69          return Scaffold(
70              appBar: AppBar(
71                  title: const Text("Grafik Harcama Özeti"),
72              ),
73              body: Column(
74                  children: [
75                      Container(
76                          padding: const EdgeInsets.all(20),
77                          margin: const EdgeInsets.only(bottom: 20),
78                          child: const Text(
79                              'Harcama özetiniz aşağıda grafik şeklinde sunulmuştur. Her bir
80                              harcama kalemine ilişkin renk kodları grafiğin sağında
81                              belirtilmiştir.'),
82
83                      PieChart(
84                          dataMap: harcamalar,
85                          chartType: ChartType.ring,
86                          chartRadius: MediaQuery.of(context).size.width / 2,
87
88                      ],
89                  );
90      }

```

Kod 115. GrafikSayfam bileşeninin kodları

**GrafikSayfam** bileşeninde gövde alanında gösterilen bileşenler 73-86 numaralı satırlar arasında bir **Column** bileşeni içine yerleştirilmiştir. Bu sayede eklenen bileşenler alt alta gösterilecektir. Üst satırda altta verilen Grafikle ilgili bir bilgilendirme metni sunulmuştur. 80-84 numaralı satırlar arasında ise **PieChart** bileşeni oluşturularak sayfaya eklenmiştir. **PieChart** yapıcı metodunun **dataMap** parametresi oluşturulacak grafik verilerini alır. **chartType** parametresi ise **ChartType** sınıfındaki öntanımlı değerlerden birini kabul eder. Bu parametre ile oluşturulacak grafiğini tipi belirlenir. **chartRadius** parametresi ise oluşturulacak grafiğin yarı çapını belirler. Burada bulunulan ortam genişliğinin yarısı verilerek grafiğin kullanılabilir alanın tümünü kaplaması sağlanmıştır.

## Dikkat

PieChart bileşenin yapıcı metodundaki parametreler hakkında daha fazla bilgi almak için [https://pub.dev/packages/pie\\_chart](https://pub.dev/packages/pie_chart) adresini ziyaret ediniz.

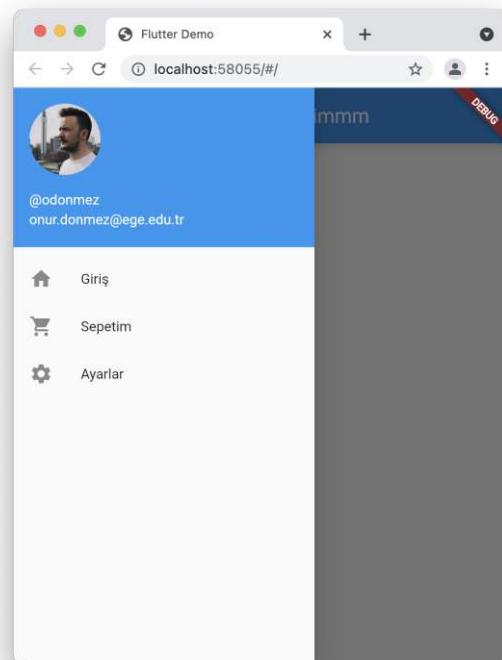
### 1.5. Gözle: Drawer (Çekmece Bileşeni)

Çekmece bileşeni mobil uygulamalarda sıkılıkla kullanılan bir içerik tutucu bileşenidir. Otomatik saklanma özelliği sayesinde uzun menüler bu bileşen altında saklı halde kullanılabilmektedir.

**Drawer** bileşenleri **Scaffold** bileşenlerinin **drawer** parametresine atanır. Bu sayede hangi sayfalarda gösterileceği belirlenmiş olur. Genellikle solda kullanılmakla birlikte, **Drawer** sayfalarda sağ tarafta da kullanılabilir. Soldan sağa yazılan diller kullanılan uygulamalara, **Scaffold** bileşenlerinin **drawer** parametresine atanın **Drawer** bileşenleri solda, **endDrawer** parametresine atanın **Drawer** bileşenleri ise sağda gösterilir. Her iki parametreye de farklı iki bileşen atanabilir.

Drawer bileşenlerinde başlıklar ve düğme listeleri kullanılabilir. Tüm bu bileşenler **ListView** gibi yayılım bileşenleri içine yerleştirilir.

Ekran Görüntüsü 31'de sayfaya eklenmiş **Drawer** bileşeni içinde 4 adet bileşen kullanılmaktadır. Bunlardan ilki kullanıcı hesaplarının gösterilmesi için özelleşmiş başlık alanı olan **UserAccountsDrawerHeader** bileşenidir. Bu bileşen kullanıcı hesabı adı (**accountName**), e-posta (**accountEmail**) ve kullanıcı resmi (**currentAccountPicture**) için özel parametreler sunmaktadır. Diğer bileşenler ise metin ve simge eklenebilen kullanıcı kontrolleri olan **ListTile** bileşenleridir.



Ekran Görüntüsü 31. *Drawer* bileşeni

```

1 import 'package:flutter/material.dart';
2 class Cekmece extends StatelessWidget {
3     const Cekmece({Key? key}) : super(key: key);
4     @override
5     Widget build(BuildContext context) {
6         return Drawer(
7             child: ListView(
8                 children: [
9                     const UserAccountsDrawerHeader(
10                         accountName: Text("@odonmez"),
11                         accountEmail: Text("onur.donmez@ege.edu.tr"),
12                         currentAccountPicture: CircleAvatar(
13                             backgroundImage: AssetImage('img/account.jpg'),
14                             // https://unsplash.com/photos/xBa0a_dd5I
15                         ),
16                     ),
17                     ListTile(
18                         title: const Text("Giriş"),
19                         leading: const Icon(Icons.home),
20                     ),
21                     ListTile(
22                         title: const Text("Sepetim"),
23                         leading: const Icon(Icons.shopping_cart),
24                     ),
25                     ListTile(
26                         title: const Text("Ayarlar"),
27                         leading: const Icon(Icons.settings),
28                     ),
29                 ],
30             ),
31         );
32     }
33 }

```

Kod 116. Drawer bileşeni kodları

Kod 116'da **Cekmece** adında bir **Drawer** bileşeni üretilmektedir. Bu bileşen bir sınıf içine alınarak daha sonra uygulamanın diğer sayfalarındaki **Scaffold** bileşenlerine eklenebilecektir. Görüldüğü üzere, **Drawer** bileşeni içine yerleştirilecek bileşenler bir **ListView** içinde yer almaktadır. 9-16 numaralı satırlar arasında oluşturulan **UserAccountsDrawerHeader** bileşeni, **Drawer** bileşenleri içinde kullanıcı hesaplarına özgü bilgileri listelemek için üretilmiştir. Bu

bileşen bir kullanıcı fotoğrafı, kullanıcı hesap adı ve kullanıcı e-postası gösterebilmektedir. Bu bilgilerin her biri ilgili parametrelere ataranarak hesap bilgileri alanı oluşturulmuştur.

**ListTile** bileşeni, metinler ve simgeleri bir arada göstermek için kullanılabilen ve etkileşim yeteneği katılabilen bir bieşendir. **ListTile** bileşenlerine **leading** (ön) ve **trailing** (arka) simgeleri eklenebilmektedir. Kod 116'da 19, 23 ve 27 numaralı satırlarda bileşenlere ön simgeler eklenmiştir. Arka simgeler ise kullanılmamıştır. **ListTile** bileşenleri bir satır ana metin (**title**) ve iki satır kadar alt metin (**subtitle**) gösterebilmektedir. Kod 116'da 18, 22 ve 26 numaralı satırlarda **title** özelliği ile ana metinler eklenmiştir. **ListTile** bileşenlerine **onTap** parametresine atanmış fonksiyonlar ile kullanıcı etkileşimi eklenebilmektedir. Bir sonraki örnekte bu bileşenlerin uygulamada özel rotaları göstermeleri sağlanacaktır.

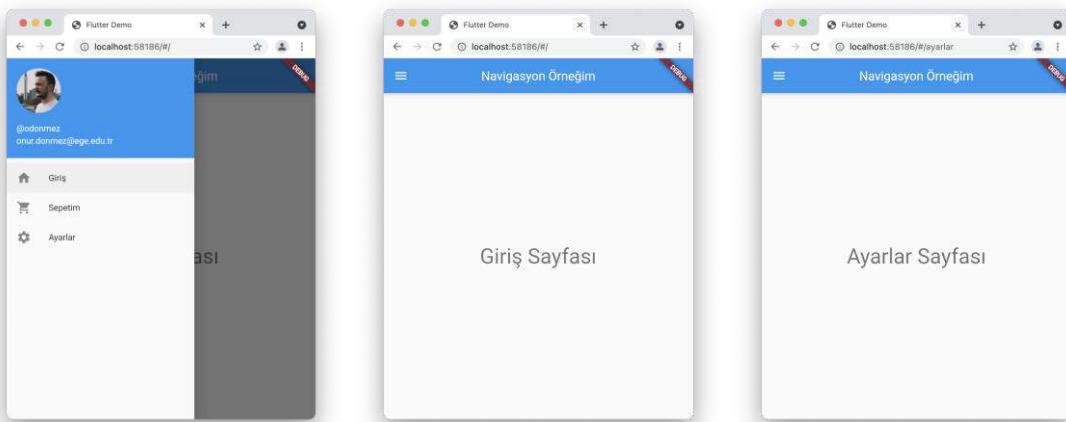
```
1 import 'cekmece.dart';
2 Widget build(BuildContext context) {
3   return Scaffold(
4     appBar: AppBar(
5       title: const Text("Giriş Ekranı"),
6     ),
7     drawer: const Cekmece(),
8     //endDrawer: const Cekmece(),
9   }
}
```

Kod 117. Scaffold bileşenine Drawer bileşeni eklemek

Kod 117'de 1 numaralı satırda, Kod 116'da oluşturulan **Cekmece** bileşeni projeye eklenmektedir. 7 numaralı satırda **drawer** parametresine bir **Cekmece** nesnesi atanarak sayfanın soluna yerleştirilmiştir. Yorum satırı haline getirilen 8 numaralı satırda bir **Cekmece** nesnesi sayfanın sağına yerleştirilmiştir.

## 1.6. Gözle: İsimli Rota Kullanımı

Flutter uygulamalarında pek çok rota kullanılabilir. Örneğin bir rota kullanıcı giriş ekranını gösterirken, diğer bir rota kullanıcıya özgü bir menüyü (Ör: bir restoran uygulamasındaki çocuk menüsü), farklı bir rota ise menüdeki bir yemeğin özelliklerini tanıtabilir. Her uygulamada olduğu gibi kullanıcı profil ayarları içinse ayrı bir rota oluşturulabilir. Bu tür senaryolarda rotaları isimleriyle çağrılmak genel bir uygulamadır. Bu uygulamada, hazırlanacak üç rota bir önceki uygulamada oluşturulan Drawer bileşeni çağrırlacaktır.



*Ekran Görüntüsü 32. Drawer bileşeni ile çağırılan rotalar*

Kod 118'de üç rotalı uygulama ve rotaları tanımlanmaktadır.

```

1 import 'package:flutter/material.dart';
2 import 'cekmecе.dart';
3 void main() {
4   runApp(const MyApp());
5 }
6 class MyApp extends StatelessWidget {
7   const MyApp({Key? key}) : super(key: key);
8   @override
9   Widget build(BuildContext context) {
10     return MaterialApp(
11       title: 'Flutter Demo',
12       initialRoute: '/',
13       routes: {
14         '/ayarlar': (context) => const AyarlarSayfasi(),
15         '/sepetim': (context) => const SepetimSayfasi(),
16         '/': (context) => const GirisSayfasi(),
17       },
18       theme: ThemeData(
19         primarySwatch: Colors.blue,
20       ),
21     );
22   }
23 }
```

*Kod 118. Uygulama ve rotaların tanımlanması*

Kod 118 incelendiğinde 10 ve 21 numaralı satırlar arasında bir **MaterialApp** bileşenin tanımlandığı görülecektir. Daha önceki uygulamaların aksine bu bileşen bir **Scaffold** bileşeni içermemektedir. Bu yaklaşımda ekranda gösterilecek içerikler **Scaffold** bileşenleri içeren harici rotalarda tanımlanmaktadır. **MaterialApp** bileşeni ise bunları yöneten bir kabuk uygulama gibi düşünülebilir.

13 ve 17 numaralı satırlar arasında bu uygulamada gösterilecek sayfalar ve bunlara ilişkin etiketler **routes** parametresi içinde tanımlanmaktadır. Bu tanımlamalar sayesinde uygulama içinde ayarların düzenlenmesi için oluşturulan bileşeni çağrırmak için bu bileşenin adı yerine (Ör: **AyarlarSayfasi**), bu alanda tanımlanan etiket (Ör: **/ayarlar**) kullanılabilir. 12 numaralı satırda ise uygulama açıldığında varsayılan olarak gösterilmesi istenen rotanın (**GirisSayfasi**) etiketi (/) **initialRoute** parametresine atanmıştır. Bu yolla uygulama **GirisSayfasi** rotasını göstererek başlamaktadır.

```
28 class GirisSayfasi extends StatelessWidget {  
29     const GirisSayfasi({Key? key}) : super(key: key);  
30     @override  
31     Widget build(BuildContext context) {  
32         return Scaffold(  
33             appBar: AppBar(  
34                 title: const Text("Navigasyon Örneğim")),  
35                 drawer: Cekmece(),  
36                 body: Center(  
37                     child: Text(  
38                         'Giriş Sayfası',  
39                         style: Theme.of(context).textTheme.headline4,  
40                     ),  
41                 ),  
42             );  
43     }  
44 }
```

Kod 119. GirisSayfasi rotası içeriği

Kod 119'da örnek bir rota içeriği sunulmuştur. Diğer rotalarda yalnızca 38 numaralı satırda hangi sayfanın gösterildiğini belirten **Text** bileşeni içeriği değiştiğinden, yer kısıtları nedeniyle bu rotaların içeriği sunulmamıştır.

Gördüğü üzere GirisSayfasi bileşeni bir **Scaffold** bileşeni içinde düzenlenmiştir. Bu sayede Kod 118'de tanımlanan kabuk **MaterialApp** bileşeni içine **Scaffold** bileşenleri yerleştirilmektedir. 35 numaralı satırda sayfalar arası gezinimi sağlamak için bir **Drawer** bileşeni eklenmektedir. Etkileşim kodları da bu bileşen içinde yer alacaktır. 37-40 numaralı satırlar arasında ise ekranda ortalanmış şekilde dördüncü düzey bir başlık üreten kodlar yer almaktadır.

```

17  ListTile(
18    title: const Text("Giriş"),
19    leading: Icon(Icons.home),
20    onTap: () => Navigator.pushNamed(context, '/'),
21  ),
22  ListTile(
23    title: const Text("Sepetim"),
24    leading: Icon(Icons.shopping_cart),
25    onTap: () => Navigator.pushNamed(context, '/sepetim'),
26  ),
27  ListTile(
28    title: const Text("Ayarlar"),
29    leading: const Icon(Icons.settings),
30    onTap: () => Navigator.pushNamed(context, '/ayarlar'),
31  ),

```

Kod 120. Drawer bileşeninde rotaların düğmelere tanımlanması

Kod 120'de, Kod 116'da tanımlanan **Cekmece** isimli **Drawer** bileşenine etkileşim ekleyen kodlar gösterilmektedir. 20, 25 ve 30 numaralı satırlarda **ListTile** bileşenlerine etkileşim katan **onTap** parametresi kullanılmıştır. Bu parametreye anonim fonksiyonlar atanarak her birinde **Navigator** nesnesinin **pushNamed** metoduna **MaterialApp** bileşeninde **route** parametresi altında tanımlanan rota etiketleri (Ör: /**ayarlar**) gönderilmiştir. Bu sayede rotalar uygulamada gösterilebilmektedir.

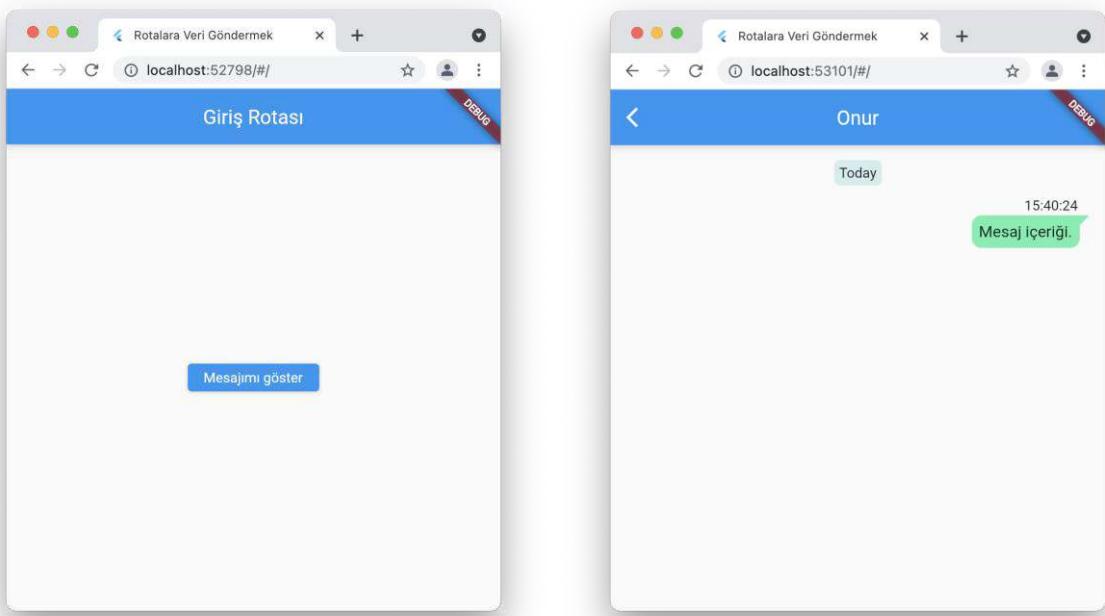
#### 1.6.1. Uygula: Yeni Rota

Üretilen üç rotalı uygulamaya yeni bir rota ekleyerek **Cekmece** bileşeni üzerinden bu rotaya erişim sağlayınız. Örneğin, kullanıcıya vereceğiniz mesajların gösterildiği uygulama içi bildirimleri sunan bir rota oluşturabilirsiniz.

### 1.7. Gözle: Rotalara Veri Aktarmak

Şu ana kadar üretilen uygulamalarda rotalar kendi içinde ürettikleri verileri ya da uygulama düzeyinde tanımlanan değişkenlerdeki verileri kullanmıştır. Fakat büyük uygulamalarda sınıflar arasında bu şekilde veri aktarımı yapmak uygulanabilir bir yöntem değildir. Uygulama verilerinin sınıflar içinde tanımlanması, işlenmesi ve korunması daha güvenilir bir yoldur. Bu veriler gerekli hallerde diğer sınıflara ya da bileşenlere gönderilerek daha temiz bir veri modeli oluşturulabilir. Bu nedenle bileşenler (rotalar) arasında veri aktarımı önemli bir mekanizmadır.

Bir rotaya veri aktarmak için kullanılabilecek ilk yok yapıçı metodunda tanımlı değişkenlere veri göndermektir. Bu amaçla veriyi alacak olan bileşenin yapıçı metodunda girilmesi zorunlu değişkenler tanımlanır. Rota Navigator nesnesine eklenirken bu değişkene veri gönderilerek çağırılır. Yeni rota bu verilerle üretilেceğinden, veri aktarımı gerçekleştirilmiş olur.



*Ekran Görüntüsü 33. Rotalar arasında veri aktarımı için kullanılan ekranlar*

Yapıcı metot yoluyla veri aktarımı basit bir mesajlaşma uygulaması içinde incelenecektir. Bu uygulama için öncelikle iletilecek mesajımıza ilişkin bir mesaj şablonu hazırlanmıştır. Bu şablonun kodları Kod 121'de sunulmuştur. Bu kodlarda mesajın kimden geldiği, geliş zamanı ve mesaj metnini tutmak için üç adet değişken tanımlanmıştır. 31 numaralı satırda bu değişkenlerin değerlerinin yapıcı metot üzerinden belirlenmesi sağlanmıştır.

```

27 class MesajSablonu {
28   final String mesaj;
29   final String kimden;
30   final DateTime zaman;
31   MesajSablonu(this.mesaj, this.kimden, this.zaman);
32 }
```

*Kod 121. Mesaj bilgilerini içeren sınıfın kodları*

Bu uygulama için pub.dev üzerinde sunulan **chat\_bubbles** paketi ve zaman bilgilerinin formatlanması için kullanılan **intl** paketi projeye eklenmiştir. **chat\_bubbles** paketi mesajlaşma uygulamalarında kullanılan mesaj balonlarını otomatik olarak oluşturabilmektedir. Paketin kullanımı ve yetenekleri ile ilgili bilgi almak için 3 numaralı satırda verilen adres incelenebilir.

```

1 import "package:flutter/material.dart";
2 import 'package:chat_bubbles/chat_bubbles.dart';
3 // https://pub.dev/documentation/chat_bubbles/latest/
4 import 'package:intl/intl.dart';
```

*Kod 122. Uygulamada kullanılan paketler*

Uygulamanın kök bileşenini olan **MaterialApp** bileşenini içeren kodlar Kod 123'te sunulmuştur. Bu kodlar çok kez açıklandığından satır satır açıklama yapılmayacaktır.

```
10  class MyApp extends StatelessWidget {
11      const MyApp({Key? key}) : super(key: key);
12
13      @override
14      Widget build(BuildContext context) {
15          return MaterialApp(
16              title: 'Rotalara Veri Göndermek',
17              theme: ThemeData(
18                  primarySwatch: Colors.blue,
19              ),
20              home: const GirisEkrani(),
21          );
22      }
23 }
```

Kod 123. Kök bileşen kodları

Kod 123'te 19 numaralı satırda uygulamanın **GirisEkrani** isimli bir bileşen ile başlatıldığına dikkat ediniz. Bir **Scaffold** içeren bu bileşende ekranda yalnızca ortalanmış bir düğme bulunmaktadır. Bu düğme mesajları gösterecek rotaya gönderilecek verileri hazırlayarak bu rotayı ekrana çağırmaktadır. Bu kodlar da sıkılıkla anlatıldığından Kod 124'te yalnızca düğme bileşenini oluşturan kodlara yer verilmiştir.

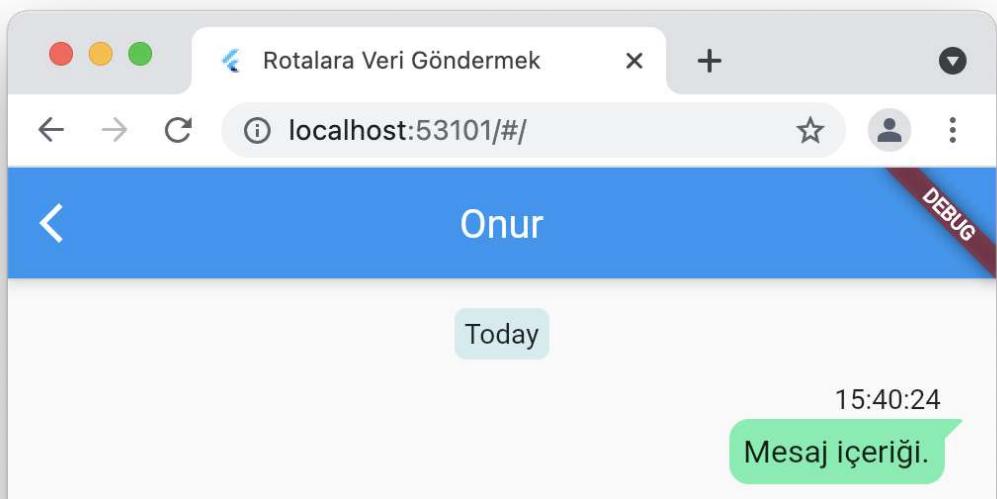
```
43  ...
44  ElevatedButton(
45      onPressed: () {
46          Navigator.push(
47              context,
48              MaterialPageRoute(
49                  builder: (context) => MesajEkrani(
50                      mesaj:
51                          MesajSablonu("Mesaj içeriği.", "Onur", DateTime.now())),
52                  ),
53              );
54      },
55      child: const Text('Mesajımı göster'),
56  ),
57  ...
```

Kod 124. MesajEkrani rotasını ekleyen kodlar

Kod 124'te 55 numaralı satırda düğme üzerinde gösterilen metin bir **Text** bileşeni içinde oluşturulmuştur. Bu düğmeye basılması halinde çalıştırılacak anonim fonksiyon 45-54 numaralı satırlar arasında yayılmaktadır. Bu fonksiyon içinde yalnızca **Navigator** nesnesine **MesajEkranı** rotasını ekleyen bir **push** komutu çalıştırılmaktadır. Bu komut içinde yeni bir **MaterialPageRoute** nesnesi oluşturulmaktadır. Bu nesne içinde de **MesajEkranı** rotasının yapıçı metodu çağrılmaktadır. Bu metot, **MesajSablonu** türünde ve **mesaj** adında bir değişken kabul edecek şekilde düzenlenmiştir. 50 numaralı satırda mesaj parametresine bir **MesajSablonu** nesnesinin atandığına dikkat ediniz. Bu nesnede sırayla, mesajın içeriği, kimden geldiği ve geliş zamanı bilgileri girilmiştir. Bu yolla mesaj değişkeni ekrana eklenen rotaya ilettilmiş olacaktır.

**MesajEkranı** bileşeninin kodları Kod 125'te sunulmuştur. 66 numaralı satırda yapıçı metot içinde **mesaj** değişkeninin **required** olarak işaretlendiğine dikkat ediniz. Bu sayede bileşen oluşturulurken bu değişkenin verilmesi zorunlu kılınmıştır.

Yapıcı metot bir **Scaffold** bileşeni çevirmektedir. Bu **Scaffold**'un başlık barına, **mesaj** değişkeni içinde gelen **kimden** bilgisi basılmıştır. Bu yolla mesajların kimden geldiği uygulama penceresinin üstünde gösterilmiştir. **Scaffold** bileşeninin gövde bölümü içeriği ekran kenarlarından uzaklaştmak için bir **Padding** bileşeni içine alınmıştır.



Gelen mesajların alt alta gösterilebilmesi için oluşturulan tüm bileşenler bir **Column** bileşeni içine alınmıştır. 77-94 numaralı satırlar arasında bu bileşenlerin **Column** bileşeninin **children** parametresine verilen bileşenler listesi içine yerleştirildiği görülebilmektedir. 78-80 numaralı satırlar arasında **chat\_bubbles** sınıfı tarafından sağlanan **DateChip** nesnesi oluşturulmuştur. Bu bileşen popüler mesajlaşma uygulamalarında kullanılan tarih etiketlerini oluşturabilmektedir. Kod 124'te mesaj oluşturulurken zaman bilgisi olarak **DateTime.now()** kullanıldığından, bu uygulamada **DateChip** içinde her zaman Today (bugün) bilgisi yer alacaktır. Daha eski bir tarih verilerek **DateChip** bileşeninin davranışını incelenebilir.

81-89 numaralı satırlar arasında mesaj balonu üzerindeki zaman bilgisinin verildiği **Text** bileşeni oluşturulmaktadır. Bu bilginin hizalanabilmesi ve çevresindeki bileşenlerle gerekli

boslukların verilebilmesi için bir Container bileşeni oluşturulmuştur. 84-88 numaralı satırlar arasında gösterilecek metin oluşturularak stil bilgileri düzenlenmiştir.

```
65  class MesajEkrani extends StatelessWidget {
66      const MesajEkrani({Key? key, required this.mesaj}) : super(key: key);
67      final MesajSablonu mesaj;
68      Widget build(BuildContext context) {
69          return Scaffold(
70              appBar: AppBar(
71                  title: Text(mesaj.kimden),
72              ),
73              body: Padding(
74                  padding: const EdgeInsets.all(8.0),
75                  child: Column(
76                      children: [
77                          DateChip(
78                              date: mesaj.zaman,
79                          ),
80                          Container(
81                              margin: const EdgeInsets.fromLTRB(0, 5, 20, 0),
82                              width: double.infinity,
83                              child: Text(
84                                  DateFormat("HH:mm:ss").format(mesaj.zaman),
85                                  textAlign: TextAlign.end,
86                                  style: TextStyle(color: Colors.grey, fontSize: 10),
87                              ),
88                          ),
89                          BubbleSpecialOne(
90                              text: mesaj.mesaj,
91                              isSender: true,
92                              color: Colors.greenAccent,
93                          ),
94                      ],
95                  ),
96              ),
97          );
98      }
99  }
```

Kod 125. MesajEkrani bileşeni kodları

90-93 numaralı satırlar arasında ise, yine **chat\_bubbles** paketi tarafından sağlanan **BubbleSpecialOne** bileşeni yoluyla yeşil mesaj balonu oluşturulmuştur. Bu balonun rengi **color** parametresi ile düzenlenmiştir. **isSender** parametresi ise mesajın gösterileceği tarafı belirtmektedir. Bu parametreye **true** değer verildiğinde mesaj sağda; **false** değer verildiğinde ise solda gösterilmektedir.

## 1.8. Gözle: İsimli Rotalara Veri Göndermek

İsimli rotalar kullanıldığında yapıcı metodlar yoluyla veri göndermek pratik olmayacağıdır. Bu durumda çağrıya eklenen argümanlar yoluyla veri gönderme işlemi gerçekleştirilebilir. Bu mekanizma bir önceki örneğin isimli rotalarla tekrarlanmasıyla incelenecaktır.

```
10  class MyApp extends StatelessWidget {
11      const MyApp({Key? key}) : super(key: key);
12
13      @override
14      Widget build(BuildContext context) {
15          return MaterialApp(
16              title: 'Rotalara Veri Göndermek',
17              routes: {
18                  "/": (context) => GirisEkranı(),
19                  "/mesajlar": (context) => MesajEkranı()
20              },
21              initialRoute: "/",
22              theme: ThemeData(primarySwatch: Colors.blue),
23      );
24  }
```

Kod 126. Uygulamanın kök bileşeni

Kod 126'da kök bileşen kodlarının isimli rotalarla çalışacak şekilde düzenlenmiş hali yer almaktadır. 16-19 numaralı satırlar arasında **routes** parametresi içinde **GirisEkranı** ve **MesajEkranı** bileşenleri için iki adet isim tanımlaması yapılmıştır. Uygulama başlarken gösterilmesi istenen isimli rota (/) **initialRoute** parametresine atanarak giriş ekranı düzenlenmiştir. Bu kullanımda **home** parametresinin kullanılmasına dikkat ediniz.

Kod 127'de Giriş sayfasında gösterilen ve isimli rotayı çağırın düğmenin sunulmuştur. Bu düğmeye etkileşim katan kodlar 48-58 numaralı satırlar arasında yer almaktadır. Düğmeye basıldığından **Navigator** sınıfının **pushNamed** metodu ile /mesajlar rotası çağrılmaktadır. Bu metodun **arguments** parametresine bir **MesajSablonu** nesnesi eklenmektedir. Bu yolla isimli rotaya veri aktarımı gerçekleştirilmiştir.

```

46 ...
47 ElevatedButton(
48   onPressed: () {
49     Navigator.pushNamed(
50       context,
51       "/mesajlar",
52       arguments: MesajSablonu(
53         "Merhaba",
54         "Onur",
55         DateTime.utc(2022, 1, 20, 16, 05, 40),
56       ),
57     );
58   },
59   child: const Text('Mesajımı göster'),
60 ),
61 ...

```

Kod 127. İsimli rotaya argüman göndermek

Kod 128'de **MesajEkranı** bileşeninin kodlarının ilk bölümünü görülmektedir. 66 numaralı satırda bu bileşenin yapıcısı metodunun kodlarının değiştirilmiştir. **mesaj** değişkeni yapıcısı metot üzerinden gelmediğinden buradaki **required** tanımlaması kaldırılmıştır.

```

65 class MesajEkranı extends StatelessWidget {
66   const MesajEkranı({Key? key}) : super(key: key);
67   @override
68   Widget build(BuildContext context) {
69     final mesaj = ModalRoute.of(context)!.settings.arguments as
      MesajSablonu;
70     return Scaffold(
71       appBar: AppBar(
72         title: Text(mesaj.kimden),

```

Kod 128. İsimli rota içinde gelen argümanların çözümlenmesi

**mesaj** değişkenine 69 numaralı satırda erişilmektedir. Bu satırda **ModalRoute** nesnesinin **arguments** özelliğine erişilmiş; **as** işleci yoluyla bu özellik içinde gelen verinin **MesajSablonu** türünde olduğu ve bu şekilde davranışması gerektiği belirtilmiştir. 72 numaralı satırda **mesaj** nesnesindeki **kimden** özelliğine erişildiğine dikkat ediniz. Kodların geri kalanında bir farklılaşma yoktur.

### 1.8.1. Uygula

Ekran Görüntüsü 30'de görülen pasta grafiği üretme uygulamasını ana ekranдан veri aktaracak şekilde güncelleyiniz. Güncel haliyle bu uygulamada harcamalar değişkeni bileşenler üzerinde

tanımlandığından her iki sayfada da kullanılabilir durumdadır. Bunun yerine harcamalar değişkenini GirişSayfam bileşeni içinde tanımlayarak, GrafikSayfam değişkenine parametre olarak gönderiniz.

## 2. TASARLA ve ÜRET

Bir mesajlaşma uygulaması tasarlaymentınız. Bu uygulamada en az üç farklı mesaj oturumu bulunması için gerekli verileri oluşturunuz. Örneğin, Onur – Beril, Onur – Fırat ve Onur – Ezgi çiftleri arasında mesajlaşmalar bulunabilir. Mesajlaşmalar için her iki tarafında en az 3 mesaj göndermiş olmasını sağlayınız. Örneğin Onur ve Ezgi arasında Onur'un gönderdiği 2, Ezgi'nin gönderdiği 3 mesaj bulunabilir. Bu verileri bir mesaj listesi şeklinde mesajların gösterileceği rotaya iletmeniz gerekecektir. Seçeceğiniz bir tarafı (Ör: Onur) ekran sahibi olarak belirleyiniz. Bu kişinin gönderdiği mesajları yeşil, diğer tarafın gönderdiği mesajları gri olacak şekilde gösterebilirisiniz.

## 3. DEĞERLENDİR

Tasarladığınız mesajlaşma ekranlarını değerlendiriniz. Arkadaşlarınızın oluşturduğu arayüzlerle kıyaslayınız. Bu arayüzleri anlaşılabilirlik (clarity), kısalık (conciseness) ve kullanışlılık (usefulness) boyutlarında değerlendiriniz. Bu terimlerin anlamları ve örnek bir tasarımın şekillenmesi sürecini incelemek için <https://uxplanet.org/ux-writing-how-to-do-it-like-google-with-this-powerful-checklist-e263cc37f5f1> adresini ziyaret ediniz.

Tasarımlarınızı daha etkili hale getirmek için neler yapabileceğinizi tartışınız.

Öğrencinin özdeğerlendirme yapabilmesi için Öz Değerlendirme Formu (<https://forms.gle/kts381jqHWcCah2j6>) uygulanır.

### Proje Hazırlama

Dersin sonunda üretilcek öğrenci projesi için oluşturulan arayüzleri öğrencilerle birlikte inceleyiniz. Öğrencilerin grup çalışmasına verdikleri katkılarla yönelik özdeğerlendirme yapabilmeleri için Grup Çalışması Öz Değerlendirme Formu'nu (<https://forms.gle/FWk3AhU2sFX5heVK8>) uygulayınız.

Bir sonraki haftaya kadar öğrencilerden uygulamalarındaki ekranlar içinde gezinim mekanizmalarını kurgulamalarını isteyiniz.

## 4. İLAVE ETKİNLİK

<https://swapi.dev/api/people/?format=json> adresinde Star Wars film serilerinde rol alan karakterlerin bilgilerinin sunulduğu bir json dosyası bulunmaktadır. Aşağıda bu belgeden bir kod parçası sunulmuştur. Luke Skywalker karakterine ait pek çok özellik bu json dosyası içinde bulunabilmektedir. Karakterlerden seçeceğiniz özellikleri farklı ekranlarda gösteren bir uygulama üretiniz.

```
{  
    "name": "Luke Skywalker",  
    "height": "172",  
    "mass": "77",  
    "hair_color": "blond",  
    "skin_color": "fair",  
    "eye_color": "blue",  
    "birth_year": "19BBY",  
    "gender": "male",  
    "homeworld": "https://swapi.dev/api/planets/1/",  
    "films": [  
        "https://swapi.dev/api/films/1/",  
        "https://swapi.dev/api/films/2/",  
        "https://swapi.dev/api/films/3/",  
        "https://swapi.dev/api/films/6/"  
    ],  
    "species": [],  
    "vehicles": [  
        "https://swapi.dev/api/vehicles/14/",  
        "https://swapi.dev/api/vehicles/30/"  
    ],  
    "starships": [  
        "https://swapi.dev/api/starships/12/",  
        "https://swapi.dev/api/starships/22/"  
    ],  
    "created": "2014-12-09T13:50:51.644000Z",  
    "edited": "2014-12-20T21:17:56.891000Z",  
    "url": "https://swapi.dev/api/people/1/"  
}
```

## 5. KAYNAKÇA

Google. (t.y.). <https://docs.flutter.dev> adresinden 18.11.2021 tarihinde erişilmiştir.

<https://medium.com/flutter-community/flutter-layout-cheat-sheet-5363348d037e>

<https://docs.flutter.dev/development/ui/layout>

<https://docs.flutter.dev/development/ui/widgets/layout>

[https://www.w3schools.com/js/js\\_json\\_intro.asp](https://www.w3schools.com/js/js_json_intro.asp)

<https://docs.flutter.dev/cookbook/navigation/navigation-basics>

## 6. Hafta: Form Uygulamaları

---

### Ön Bilgi:

- Temel Dart bilgisi
- Temel internet kullanım bilgisi
- Flutter ile bileşen üretme bilgisi

### Haftanın Kazanımları:

- Form elementleri oluşturabilir
- Form elementlerinden gelen verileri kullanabilir
- Flutter dokümantasyonunu inceleyerek daha önce kullanmadığı bir sınıfı kullanabilir

### Haftanın Amacı:

Bu haftanın amacı form elementleri içeren bir mobil uygulama oluşturmaktır.

### Kullanılacak Malzemeler:

Visual Studio Code, Chrome internet tarayıcısı.

### Haftanın İşlenişi

*Gözle:* Flutter dokümantasyonu arayüzünün önemli alanlarını incele.

*Uygula:* Basit bir Yapıacaklar Listesi uygulaması hazırla.

*Tasarla:* Bir rehber uygulaması tasarla.

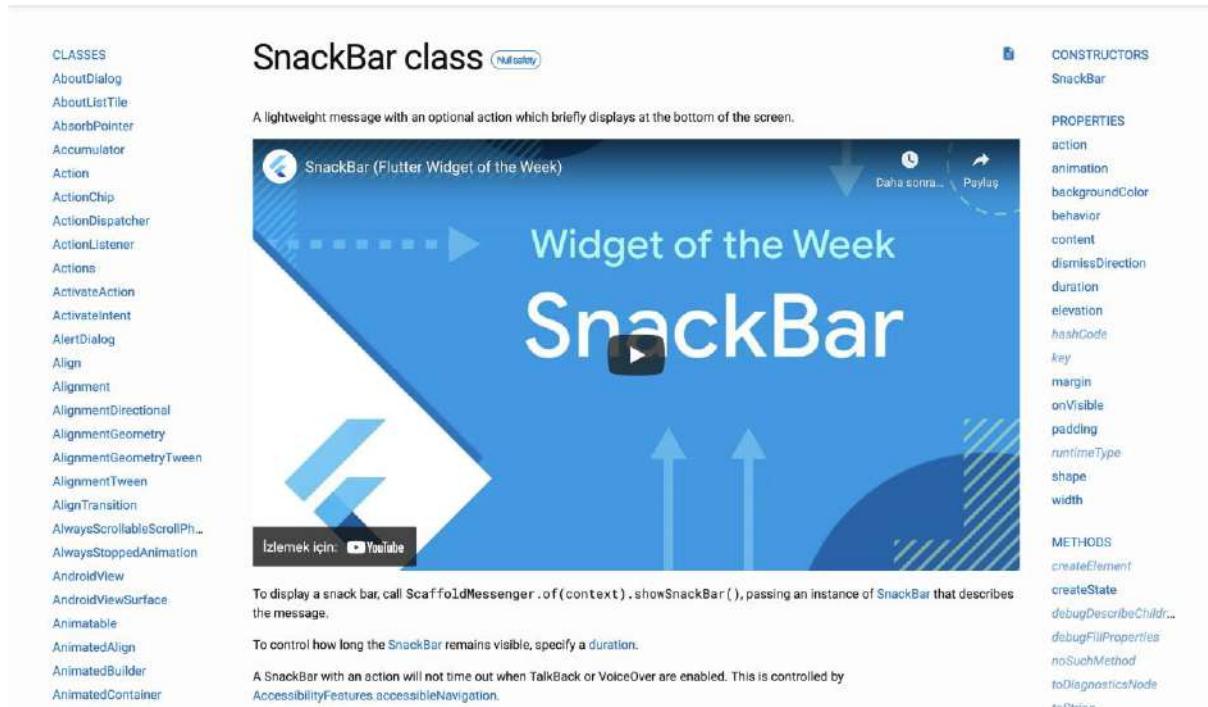
*Üret:* Tasarladığın rehber uygulamasını üret.

*Değerlendir:* Ürettiğin rehber uygulaması arayüzüne değerlendir.

# 1. GÖZLE VE UYGULA

## 1.1. Gözle: Flutter Dokümantasyonu

Flutter, İngilizce dilinde hazırlanmış güçlü bir dokümantasyon sistemine sahiptir. Kullanılacak bileşen ya da mekanizmaların aranması sonucu ulaşılacak sayfaların yapısı anlaşıldığından bu dokümantasyondan kolayca faydalınabilir. Ekran Görüntüsü 34’te sayfaların alt bölümlerinde kısa bilgilendirme metinleri göstermek için kullanılabilcek **SnackBar** bileşeninin dokümantasyon sayfası görülmektedir.



Ecran Görüntüsü 34. SnackBar bileşeni dokümantasyon sayfası

Bu sayfada orta bölümde ilgili bileşenin nasıl kullanılacağı ve neler yapabileceği konusunda bilgilendirici içerikler bulunmaktadır. Çoğu bileşen için kısa video içerikler sunulmaktadır. Bu video içeriklerin altında bileşenlerin nasıl kullanılacağına ilişkin kod örnekleri ve bu kod örneklerinin web ortamında işletilebileceği editörler (Ekran Görüntüsü 35) bulunur.

Bu sayfadaki sağ menünün düzeni anlaşıldığından aranan içeriklere daha kolay ulaşılacaktır. Sağ menüde üstten aşağı doğru bileşenlerin yapıçı metodları (**Constructors**), özellikleri (**Properties**) ve metodları (**Methods**) listeleri yer alır. Sayfaya bir **SnackBar** elementi eklenmek istiyorsa yapıçı metodlarının incelenmesi yeterli olacaktır. **SnackBar** elementinin görünüşünün değiştirilmesi isteniyorsa ilgili özellik (Ör: **backgroundColor**), **Properties** alanında bulunacaktır. Bileşen özel bir metod destekliyorsa, bu metod **Methods** alanında bulunacaktır. **Methods** alanında açık renkli metodlar incelenen bileşenin miras aldığı metodları ifade eder. Bu metodlar desteklense de özellikle **SnackBar** için üretilmiş metodlar değildir. Flutter tanımlayıcı bir dil olduğundan çoğunlukla özellikler ve yapıçı metodların nasıl kullanılacağını bilinmesi yeterli olacaktır.

```

1 import 'package:flutter/material.dart';
2
3 void main() => runApp(const MyApp());
4
5 class MyApp extends StatelessWidget {
6   const MyApp({Key? key}) : super(key: key);
7
8   static const String _title = 'Flutter Code Sample';
9
10  @override
11  Widget build(BuildContext context) {
12    return MaterialApp(
13      title: _title,
14      home: Scaffold(
15        appBar: AppBar(title: const Text(_title)),
16        body: const Center(
17          child: MyStatelessWidget(),
18        ),
19      ),
20    );
21  }

```

To create a local project with this code sample, run:  
flutter create --sample=material.SnackBar.1 mysample

*Ekran Görüntüsü 35. Flutter dokümantasyonu sayfasında yer alan web editörü*

Flutter dokümantasyonunun önemli bir parçası da Cookbook alanıdır. Bu alanda Flutter kullanarak özel bir görevin nasıl gerçekleştirileceği kısa reçeteler şeklinde anlatılmaktadır.

## Cookbook

Cookbook

This cookbook contains recipes that demonstrate how to solve common problems while writing Flutter apps. Each recipe is self-contained and can be used as a reference to help you build up an application.

### Animation

- Animate a page route transition
- Animate a widget using a physics simulation
- Animate the properties of a container
- Fade a widget in and out

### Design

- Add a Drawer to a screen
- Display a snackbar
- Export fonts from a package
- Update the UI based on orientation
- Use a custom font
- Use themes to share colors and font styles
- Work with tabs

*Ekran Görüntüsü 36. Flutter Cookbook (Yemek Kitabı) sayfası*

Örneğin <https://docs.flutter.dev/cookbook/design/snackbars> adresinde bu başlıkta bahsedilen **SnackBar** bileşeni ile ilgili bir reçete bulunabilir. Bu sayfa incelemişinde **SnackBar** bileşeninin gösterilebilmesi için üç adımlı bir reçetenin kodlarıyla birlikte sunulduğu görülecektir.

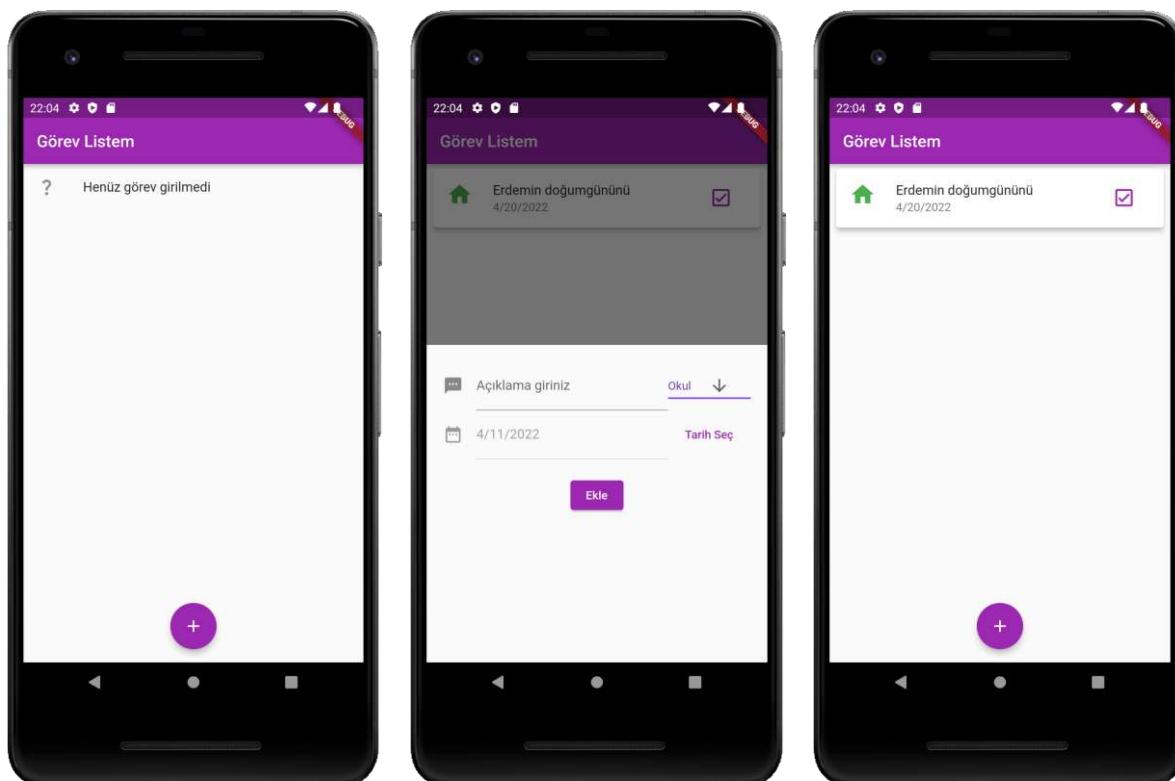
### *1.1.1. Uygula: Snackbar gösterimi*

Hazırlayacağınız boş bir sayfaya ekleyeceğiniz bir düğme ile gösterilen ve 2 saniye sonra kendiliğinden kaybolan bir Snackbar bileşeni tasarlayıınız.

## 1.2. Gözle: Form Uygulaması

Pek çok mobil uygulama kullanıcılarının isteklerini gerçekleştirmek için veri giriş araçlarını kullanır. Kullanıcılarından veri toplamak için genellikle formlar kullanılır. Flutter formlardaki girdi araçları için zengin bir özellik yelpazesi sunar. Bu özellikler girdi elementlerinden gelen verilerin geçerliğinin sağlanmasıından, elemente tıklandığında gösterilecek klavye türüne kadar çeşitlenmektedir.

Kullanıcıların kolayca kavrayabileceği ve az emekle doğru değerleri girebileceği form tasarımları üretmek önemli bir çalışma alanıdır. Bu çerçevede girdi elementlerinin bahsedilen özelliklerinin anlaşılması ve yerinde kullanılması oldukça önemlidir. Örneğin geçerli bir tarih bilgisi almak için metin girdisi yapılabilen boş bir kutu ya da bir takvim gösterilebilir. Benzer şekilde telefon tuşlamak için numerik klavye ya da normal sanal klavye kullanılabilir. Form girdilerinde doğru araçların doğru anlarda kullanılması mobil uygulamaların başarısının ve kabul edilirliğinin en önemli belirleyicileri arasındadır.



*Ekran Görüntüsü 37. Görev Listesi uygulaması arayüzü*

Bu uygulamada form elementlerinin ve kullanıcı girdisinin örneklenmesi için basit bir Görev Listesi uygulaması hazırlanacaktır. Ekran Görüntüsü 37'de bu uygulamanın boş liste ile (solda), veri girişi formu ile (ortada) ve bir adet veri girilmiş (sağda) halleri gösterilmektedir. Bu görüntülerin sağlanması için uygulama Google Pixel 2 Android Telefon emülatörü üzerinde çalıştırılmıştır.

```

1 import 'package:flutter/material.dart';
2 class GorevModeli {
3     num id;
4     String baslik;
5     DateTime tarih;
6     Icon simge;
7     GorevModeli(
8         { required this.id,
9             required this.baslik,
10            required this.tarih,
11            required this.simge});
12 }
```

Kod 129. *GorevModeli* sınıfı kodları

Uygulamadaki görevlerin oluşturulması için bir **GorevModeli** sınıfı oluşturulmuştur. Her bir görevin bir id numarası, bir başlığı, gerçekleştirilmesi gereken tarih bilgisi ve görevlerin yaşamın hangi alanında olduğunu gösteren bir simgesi bulunmaktadır. Yapıcı metotta tüm bu bilgilerin girilmesi gerekli kılmıştır. Hiçbir görevin eksik bilgi ile oluşturulmasına izin verilmeyecektir. Bu kütüphane projenin kök klasöründe **modals** klasörü altına yerleştirilmiştir.

```

1 import 'package:flutter/material.dart';
2 import './modals/gorevmodeli.dart';
3 import './widgets/gorevlistesi.dart';
4 import './widgets/yenigorev.dart';
5 void main() {
6     runApp(MyApp());
7 }
8 class MyApp extends StatelessWidget {
9     @override
10    Widget build(BuildContext context) {
11        return MaterialApp(
12            title: 'Görev Listem',
13            theme: ThemeData(
14                primarySwatch: Colors.purple,
15            ),
16            home: Gorevler(),
17        );
18    }
19 }
```

Kod 130. *main.dart* dosyasında üretilen uygulama kodları

Kod 130'da uygulamanın giriş kodları yer almaktadır. Uygulamanın başında 3 projeye eklenmektedir. Bunların ilki **GorevModeli** sınıfını tutan model dosyasıdır. Diğer ikisi ise yeni bir görevin oluşturulması için üretilen **YeniGrev** bileşeni ve oluşturulmuş görevlerin listelendiği **GrevListesi** bileşenleridir. Kalan kodlarda bir **MaterialApp** nesnesi (**MyApp**) üretilmektedir. Bu nesnenin home parametresinde **Grevler** bileşeni yer almaktadır.

```
23 class Grevler extends StatefulWidget {
24   @override
25   _GrevlerState createState() => _GrevlerState();
26 }
27 class _GrevlerState extends State<Grevler> {
28   List<GorevModeli> _grevler = [];
29   var _grevSayisi = 0;
30   void _grevEkle(
31     ...
32   }
33   void _grevSil(num _grevId) {
34     ...
35   }
36   void _yeniGrevPenceresi(BuildContext ctx) {
37     ...
38   }
39   @override
40   Widget build(BuildContext context) {
41     return Scaffold(
42       appBar: AppBar(
43         title: const Text("Grev Listem"),
44       ),
45       body: GrevListesi(_grevler, _grevSil),
46       floatingActionButtonLocation:
47         FloatingActionButtonLocation.centerFloat,
48       floatingActionButton: FloatingActionButton(
49         child: Icon(Icons.add),
50         onPressed: () => _yeniGrevPenceresi(context),
51       ),
52     );
53   }
54 }
```

Kod 131. StatefulWidget tipindeki Grevler bileşeni kodları

**Gorevler** bileşeninin kodları Kod 131'de sunulmuştur. Bu bileşenin içeriği kendini güncelleyeceğinden bileşen **StatefulWidget** tipindedir. **Gorevler** bileşeni bir **Scaffold** nesnesi çerçevesinde hazırlanmıştır. Bu bileşenin **body** özelliğine **GorevListesi** bileşeni yerleştirilmiştir. Ayrıca ekranın alt bölümünde gösterilen düğmeye (**FloatingActionButton**) yeni görev ekleme panelini açan bir metot bağlanmıştır.

Bileşende görevler **GrevModeli** tipindeki elementlerden oluşan **\_grevler** isimli bir listede tutulmaktadır. Bunun yanında görevlerin **id** numarası olarak kullanılmak üzere üretilen görev sayısını tutan bir **\_grevSayisi** bileşeni tanımlanmıştır. Bileşende yeni bir görevin eklenmesi (**\_grevEkle**) ve görev silme (**\_grevSil**) amaçlarıyla birer metot tanımlanmıştır. Son olarak yeni görev ekleme panelinin gösterilmesi için **\_yeniGrevPenceresi** metodu hazırlanmıştır. Bu yapıda uygulamadaki görevlerin listesi üst düzeyde **StatefulWidget** tipindeki **Gorevler** bileşeninde tutulmaktadır. Bu nedenle görevlerin eklenmesi ve silinmesi ile ilgili rutinler **Gorevler** bileşeni içinde hazırlanmıştır. Görev ekleme ve görev silme ile ilgili istekler hazırlanacak bileşenlerden bu rutinlere yapılacak çağrılarla işletilecektir.

```
32 void _grevEkle(  
33 {required String baslik, required DateTime tarih, required Icon simge})  
34 {  
35     setState(() {  
36         _grevler.add(GrevModeli(  
37             id: _grevSayisi,  
38             baslik: baslik,  
39             tarih: tarih,  
40             simge: simge));  
41     });  
42     _grevSayisi++;  
43 }
```

Kod 132. **\_grevEkle** metodunun kodları

**Gorevler** bileşenindeki **\_grevEkle** metodu tümü zorunlu olan **baslik**, **tarih** ve **simge** parametreleri ile çağrılmaktadır. Metodun gövdesinde **setState** fonksiyonu içinde **\_grevler** listesine yeni bir **GrevModeli** nesnesi eklenmektedir. Bu sayede nesne eklendiği anda gerekli tüm alt bileşenler uyarilarak içeriklerin güncellenmesi sağlanmaktadır. Görevlerin kimlik numarası olarak düşünülebilecek **id** değişkeni her yeni görev üretildiğinde güncellenmektedir.

```
45 void _grevSil(num _grevId) {  
46     setState(() {  
47         _grevler.removeWhere((_grev) => _grev.id == _grevId);  
48     });  
49 }
```

Kod 133. **\_grevSil** metodunun kodları

Kod 133'te `_gorevSil` metodunun kodları sunulmuştur. Bu metot silinmesi istenen görev nesnesinin `id` değeri ile çağrılmaktadır. Listenin `removeWhere` metodu kullanılarak, `id` değeri eşleşen kayıtların silinmesi sağlanmaktadır. Bu işlem `setState` fonksiyonu içinde gerçekleştirildiğinden silme işleminin etkileri arayüze otomatik olarak aktarılmaktadır.

```
51. void _yeniGorevPenceresi(BuildContext ctx) {  
52.   showModalBottomSheet(  
53.     context: ctx,  
54.     builder: (BuildContext ctx) {  
55.       return Container(  
56.         height: 250,  
57.         child: YeniGorev(_gorevEkle),  
58.       );  
59.     },  
60.   );  
61. }
```

Kod 134. `_yeniGorevPenceresi` metodu kodları

Kod 134'te yeni görevlerin oluşturulması için açılan paneli açan kodlar yer almaktadır. **Modal Bottom Sheet** bileşeni açıldığında uygulamanın geri kalanı ile etkileşimi engeller. Bu sayede kullanıcının bir noktaya odaklanması sağlanmış olur. `_yeniGorevPenceresi` metodu çağrıldığında `showModalBottomSheet` metodunu kullanarak **Modal Bottom Sheet** bileşenini aktive eder. Bu metot sadece bir `BuildContext` parametresi almaktadır. Metodun `builder` parametresine verilen anonim fonksiyon içinde yeni bir bileşen ağacı üretilerek **Modal Bottom Sheet** bileşeni içinde gösterilmesi sağlanabilmektedir. Bu örnekte bir `Container` bileşeni üretilmiş, `child` parametresine de `YeniGorev` bileşeni bağlanmıştır. Bu bileşenin yapıçı metoduna, bileşenden toplanacak veriler üzerinden yeni görev üreterek listeye ekleyecek olan `_gorevEkle` metodu gönderilmiştir.

Kod 135'te `GorevListesi` bileşeninin kodları yer almaktadır. `GorevListesi` bileşeni `StatelessWidget` tipindedir, bu nedenle bileşen içinde durum güncellemesi yapılmayacaktır. Durum güncellemesi bileşen ağacında üstte yer alan `Gorevler` bileşeni içinde yapılacak şekilde planlanmıştır. Bu nedenle `GorevListesi` içinde oluşturulan her bir görev satırında görev silme ile ilgili bir düğme eklenecektir. Fakat bu düğme görevleri silmeyecek, üst düzeydeki `Gorevler` bileşeninde görev silme amacıyla hazırlanmış metoda çağrı yapacaktır. Bu çağrıının işletilmesi ardından `Gorevler` bileşeni durum güncellemesi (`setState`) yapacağı için `GorevListesi` bileşeni de otomatik olarak etkilenecektir.

`GorevListesi` bileşenin yapıçı metodu parametre olarak uygulamada tutulan görev listesini (`_gorevlerim`) ve bir görevin silinmesi için `Gorevler` bileşeni içinde tanımlı silme fonksiyonunu (`_gorevSil`) almaktadır. 9. satırda durumsal atama işlevi kullanılarak `_gorevlerim` listesini boş olma durumu kontrol edilmektedir. Bu liste boş gelmişse, geriye tek bir `ListTile` bileşeni çevrilmektedir. Bu bileşeneden henüz görev eklenmediğini belirten bir mesaj iletilmektedir. Fakat `_gorevlerim` listesinde eleman gelmişse, geriye `Listview.builder`

metodu ile hazırlanacak farklı bir liste çevrilmektedir. Bu metodun kodları Kod 136'da verilmiştir.

```
1 import 'package:flutter/material.dart';
2 import 'package:todo/modals/gorevmodeli.dart';
3 import 'package:intl/intl.dart';
4 class GorevListesi extends StatelessWidget {
5   List<GorevModeli> _gorevlerim;
6   Function _gorevSil;
7   GorevListesi(this.gorevlerim, this._gorevSil);
8   Widget build(BuildContext context) {
9     return gorevlerim.isEmpty
10    ? ListTile(
11        title: Text("Henüz görev girilmedi"),
12        leading: Icon(Icons.question_mark),
13      )
14    : ListView.builder(
15        ...
16      );
17  }
18 }
```

Kod 135. GorevListesi bileşeni kodları

**builder** metodunun `_gorevlerim` listesindeki her element için çalıştırılması için **itemCount** parametresine listenin uzunluğu atanmıştır. **itemBuilder** parametresine atanın anonim fonksiyona konteks ve **index** bilgileri gönderilmiştir. Bunlardan **index** parametresi 19 numaralı satırda listeden aktif elementin seçilmesi (**e**) için kullanılmıştır. Anonim fonksiyon her dönüşünde bir **Card** bileşeni üretmektedir. **Card** bileşenleri kenarları yumoşatılmış ve gölge eklenmiş nesneler üretir. Bu yolla satırların artalandan ayrılması ve farkedilebilir olması sağlanır. 21 numaralı satırda **Card** bileşeninin artalandan uzaklışı 5 birim olacak şekilde belirlenmiştir. Bu birim ne kadar arttırılırsa bileşen arkasındaki gölge o kadar büyüyecektir. **Card** nesnelerinin **child** parametresine diğer bileşenler eklenerek içerikler **Card** üzerinde gösterilmektedir. Bu örnekte de **child** parametresine bir **ListView** bileşeni aktarılarak bu bileşendeki slotlara simgeler, metinler ve düğmeler eklenmiştir. 24 numaralı satırda görevle ilgili simge eklenmektedir. 25 numaralı satırda görev açıklaması, 26 numaralı satırda ise görevin gerçekleştirileceği tarih bilgileri bileşen üzerine işlenmektedir. 27-33 numaralı satırlar arasında ise artalanı ve kenarlıklarını olmayan bir düğme üretmek için **TextButton** bileşeni kullanılmıştır. 28-31 numaralı satırlar arasında bu bileşenin içine metin yerine bir simge eklenmektedir. 32 numaralı satırda ise düğmeye basılması halinde görevin silinmesi için **Gorevler** bileşeni içinde hazırlanarak `_gorevSil` parametresine atanın fonksiyona çağrı yapılacaktır. Bu çağrıda ilgili görevin **id** bilgisi aktarılarak (`_gorevSil(3)`) doğru bileşenin silinmesi sağlanmaktadır.

```

16   :ListView.builder(
17     itemCount: _gorevlerim.length,
18     itemBuilder: (ctx, index) {
19       GrevModeli e = _gorevlerim[index];
20       return (Card(
21         elevation: 5,
22         margin: EdgeInsets.symmetric(horizontal:8,vertical:5),
23         child: ListTile(
24           leading: e.simge,
25           title: Text(e.baslik),
26           subtitle: Text.DateFormat.yMd().format(e.tarih)),
27           trailing: TextButton(
28             child: Icon(
29               Icons.check_box_outlined,
30               size: 28,
31             ),
32             onPressed: () => _grevSil(e.id),
33           ),
34         ),
35       ));
36     }
37   );

```

Kod 136. ListView.builder metodu kodları

Kod 137'de **StatefulWidget** türündeki **YeniGrev** bileşeninin kodlarının özeti yer almaktadır. Her bir metot ve bileşenin, bileşen ağacı ilerleyen kodlarda daha detaylı açıklanacaktır. Bu bileşende tarih işlemleri için **intl** kütüphanesi ve arayüzün oluşturulması için **material** kütüphanesi kullanılmaktadır. Bileşende toplanan veriler **Grevler** bileşeni içinde tanımlı **\_grevEkle** metodu ile listeye eklenmektedir. Bu nedenle bu bileşenin bu metoda erişmesi gerekecektir. Bu amaçla bileşenin yapıcı metodunda bu metoda referans veren **\_grevEkleyici** parametresi tanımlanmıştır. Kod 134'teki 57 numaralı satırda **\_grevEkle** metodunun bu bileşene parametre olarak aktarıldığı görülmektedir.

**\_YeniGrevState** sınıfı incelendiğinde bu sınıf içinde **\_tarih** (görev tarihi), **\_baslik** (görev açıklaması), **\_simge** (görev simgesi) için değişkenler tanımlanlığı görülmektedir. Ayrıca bileşen ağacına eklenen metin kutusundaki veriye erişebilmek için **TextEditingController** tipinde **\_baslikKontrolcusu** değişkeni tanımlanmıştır.

Görev tarihinin seçilebilmesi için ilgili bileşeni aktive eden bir **\_tarihSec** metodu, toplanan veriler ile yeni görevin oluşturulabilmesi için gerekli çağrıyı üreten bir **\_grevUret** metodu ve verilerde bir hata bulunması halinde hata bildirimini yapmak amacıyla bir **\_hataBildir** metodu da tanımlanmıştır.

```

1 import 'package:flutter/material.dart';
2 import 'package:intl/intl.dart';

3 class YeniGorev extends StatefulWidget {
4     final Function _gorevEkleyici;
5     YeniGorev(this._gorevEkleyici);
6     @override
7         State<YeniGorev> createState() => _YeniGorevState();
8 }

9 class _YeniGorevState extends State<YeniGorev> {
10     DateTime _tarih = DateTime.now();
11     String? _baslik;
12     String? _simge = "Okul";
13     final _baslikKontrolcusu = TextEditingController();

14     _tarihSec() {
15         ...
16     }

17     _gorevUret() {
18         ...
19     }

20     _hataBildir(String mesaj) {
21         ...
22     }

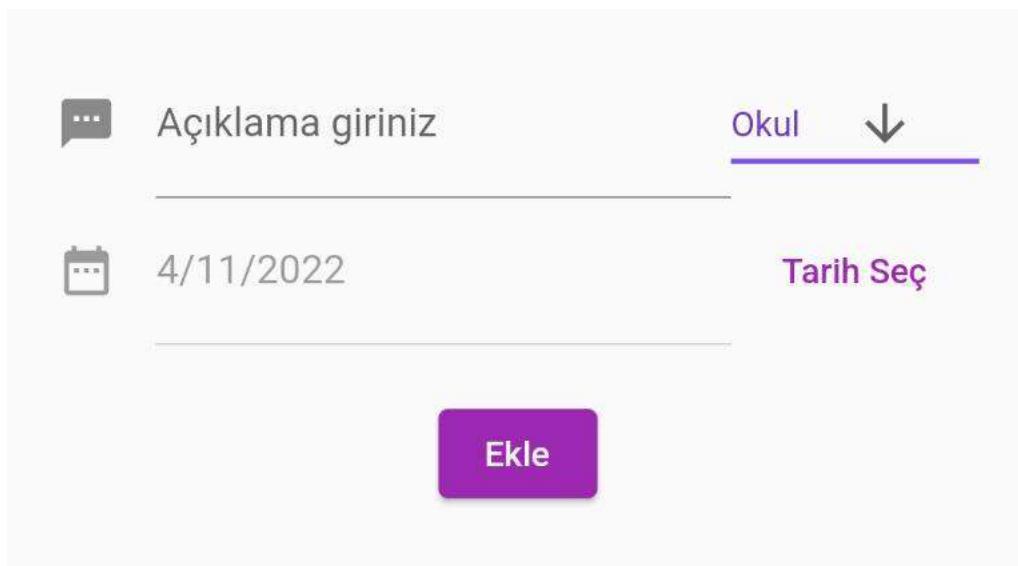
23     @override
24     Widget build(BuildContext context) {
25         return Container(
26             child: Column(
27                 ...
28             );
29     }

```

Kod 137. YeniGorev bileşeninin kodlarının özeti

Öncelikle Ekran Görüntüsü 38'de görülen bileşen ağacının açıklamaları, daha sonra da bu bileşenlerin tanımlanan metodlarla etkileşimleri açıklanacaktır. Bileşen ağacı bir **Container**

bileşeni içine yerleştirilmiştir. Bu bileşenin **padding** ve **alignment** parametreleri kullanılarak nesnelerin hizalanmaları ve yerleşimleri düzenlenmiştir.



*Ekran Görüntüsü 38. YeniGorevPenceresi bileşeni*

Kod 138'de bileşen ağacının ilk satırındaki **TextField** ve **DropdownButton** bileşenlerinin kodları yer almaktadır. Bu bileşenlerin yan yana gösterilmesi için bileşenler bir **Row** nesnesi içine alınmıştır. **DropdownButton** bileşeni bir **SizedBox** bileşeni içine alınarak boyutları ayarlanmıştır. **TextField** bileşeni **Expanded** bileşeni içine alınarak kalan tüm boşluğu kaplaması sağlanmıştır.

**TextField** bileşeni içindeki değere ulaşabilmek için bileşene **TextEditingController** tipinde bir kontrolcü (**\_baslikKontrolcusu**) atanmıştır. Yeni görevi oluşturan kodlarda metin kutusu içindeki değer bu kontrolcü üzerinden kontrol edilecektir. Ayrıca kullanıcıları görsel olarak yönlendirmek için **decoration** parametresine bir **InputDecoration** bileşeni eklenmiştir. Bu bileşenin yapıcı metodunda **icon** parametresine bir simge, bileşende girilmiş bir metin olmadığından gösterilecek metin (**labelText**) parametresine de açıklama eklenmiştir.

115 numaralı satırdan itibaren **DropdownButton** bileşeninin kodları yer almaktadır. Bu bileşen **String** tipinde değerler almaktadır. Nesnenin giriş değeri **\_simge** bileşenindeki değer olarak ayarlanmıştır. Bu değerin **setState** fonksiyonu içinde değiştirilmesi halinde bileşende gösterilen değer de değiştirilmiş olacaktır. **icon** parametresi ile metnin sağında gösterilen aşağı ok simgesi eklenmiştir. **elevation** parametresi ise açılacak değerler kutusunun artalandan ne kadar yüksek olacağını belirlemektedir. **style** parametresindeki renk belirlemesi ile bileşen içindeki yazıların rengi, **underline** parametresi ile de bileşenin alt kenarlığının kalınlığı ayarlanmıştır. **onChanged** parametresine atanın anonim fonksiyonda, bileşen üzerindeki değerin değiştirilmesi halinde **setState** fonksiyonu çağrılarak arayüzün güncellenmesi sağlanmıştır. **value** parametresine **\_simge** değişkeninin değeri atandığından, burada **\_simge** değişkeninin içeriğinin güncellenmesi yeterli olacaktır. **items** parametresine ise **String** tipinde nesnelerden oluşan bir **DropdownMenuItem** listesi eklenmiştir. Bu listenin oluşturulması için verilen **String** listesinin **map** metodu kullanılmıştır.

```

103. Row(
104.   children: [
105.     Expanded(
106.       child: TextField(
107.         controller: _baslikKontrolcusu,
108.         decoration: InputDecoration(
109.           labelText: "Açıklama giriniz",
110.           icon: Icon(Icons.textsms),
111.         ),
112.       ),
113.     SizedBox(
114.       width: 100,
115.       child: DropdownButton<String>(
116.         value: _simge,
117.         icon: const Icon(Icons.arrow_downward),
118.         elevation: 8,
119.         style: const TextStyle(color: Colors.deepPurple),
120.         underline: Container(
121.           height: 2,
122.           color: Colors.deepPurpleAccent),
123.         onChanged: (String? newValue) {
124.           setState(() {
125.             _simge = newValue!;
126.           });
127.         },
128.         items: <String>['Okul', 'Ev', 'İş', 'Eğlence']
129.           .map<DropdownMenuItem<String>>((String value) {
130.             return DropdownMenuItem<String>(
131.               value: value,
132.               child: Text(value),
133.             );
134.           }).toList(),
135.         ),
136.       ],
137.     )

```

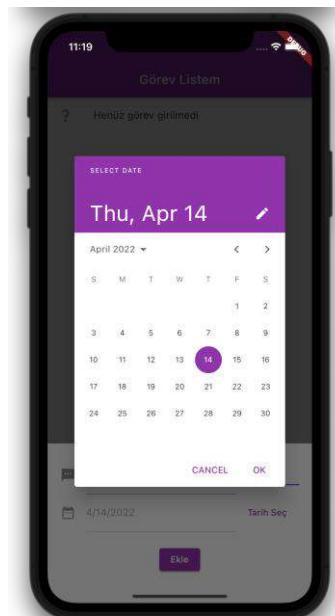
Kod 138. Metin kutusu ve görev türü seçici kodları

```

141 Row(
142   children: [
143     Expanded(
144       child: TextField(
145         enabled: false,
146         decoration: InputDecoration(
147           labelText: (_tarih == null
148             ? "Tarih seçiniz"
149             : DateFormat.yMd().format(_tarih.toLocal())),
150           icon: Icon(Icons.date_range),
151         ),
152       ),
153     ),
154     SizedBox(
155       width: 100,
156       child: TextButton(
157         onPressed: _tarihSec,
158         child: Text("Tarih Seç"),
159       ),
160     )
161   ],
162 )

```

Kod 139. Tarih seçici kodları



Ekran Görüntüsü 39. Tarih seçici bileşeni

Kod 139'da seçilen tarihin gösterildiği **TextField** bileşeni ve **material** kütüphanesinin tarih seçici bileşenini çağırın düğmenin kodları yer almaktadır. Bu iki bileşenin yan yana gösterilmesi için bileşenler bir **Row** bileşeni içine alınmıştır. Boyutlarının ayarlanması içinse yine **Expanded** ve **SizedBox** bileşenleri içine yerleştirilmiştir.

Tarih verileri kullanılan dile göre farklı şekillerde ifade edilebileceğinden doğrudan **TextField** bileşeni içine girilmesine izin verilmemiştir. Bu amaçla **TextField** bileşeninin **enabled** parametresine **false** değeri atanmıştır. **TextField** nesnesinin **decoration** parametresine tarih seçimi ile ilgili olduğunu belirtmek için bir takvim simgesi eklenmiştir. Ayrıca **labelText** parametresine **\_tarih** değişkeninde bir değer bulunmaması halinde “**Tarih seçiniz**” açıklaması, aksi halde seçilen tarihin yazılması sağlanmıştır. 156 numaralı satırda **TextButton** ile artaları ve kenarlıklarını görünmeyen bir düğme üretilmiştir. Bu düğmeye dokunulması halinde **\_tarihSec** metodu çağrılmıştır.

Kod 140'ta, Ekran Görüntüsü 39'da görülen tarih seçici diyalog penceresi bileşenini çağrıdan kodlar yer almaktadır. Bu diyalog penceresinin gösterilmesi için showDatePicker metodu kullanılmaktadır. Bu metotta seçili tarih (initialDate), seçilebilecek ilk tarih (firstDate) ve seçilebilecek son tarih (lastDate) parametrelerine sırasıyla, günün tarihi, içinde bulunulan yıl ve içinde bulunulan yıldan sonraki 3 yıl değerleri girilmiştir. Tarih seçici nesnesinde işlem yapılması haline işletilecek kodlar then metodu ile bağlanmıştır. Bu metot içine yerleştirilen anonim fonksiyona seçilen tarih parametresi gönderilmektedir. Bu parametrede gelen değer null ise (Ör: kullanıcı İptal ya da Geri düğmesine bastı ise) return çalıştırılarak herhangi bir işlem yapılmaz. Fakat bir tarih seçildi ise bu değer \_tarih değişkenine setState metodu içinde aktarılmıştır. Bu sayede seçilen tarihin TextField bileşenine işlenmesi sağlanmıştır.

### Dikkat

Seçilen tarih değeri **DateTime** veri tipindedir. Bu değerin doğrudan **TextField** bileşenine yazdırılması anlaşılırlığı güçlendirilmesi gerekmektedir. Bu nedenle Kod 140'da 149 numaralı satırda biçimlendirilerek yazıldığına dikkat ediniz. **DateFormat** sınıfının kullanılabilmesi için intl kütüphanesinin projeye eklenmesi gerekmektedir.

```
18  _tarihSec() {
19      showDatePicker(
20          context: context,
21          initialDate: DateTime.now(),
22          firstDate: DateTime(DateTime.now().year),
23          lastDate: DateTime(DateTime.now().year + 3),
24      ).then((pickedDate) {
25          if (pickedDate == null) {
26              return;
27          }
28          setState(() {
29              _tarih = pickedDate;
30          });
31      });
32  }
```

Kod 140. \_tarihSec metodu kodları

Kod 141'de formda toplanan veriler üzerinden yeni bir görevin eklenmesi için gerekli rutinleri çağrıran **Ekle** düğmesinin kodları görülmektedir. Bu amaçla **ElevatedButton** bileşeni kullanılmıştır. Düğmeye dokunulduğunda **\_gorevUret** metodu çağrılmaktadır.

```

163 Padding(
164   padding: EdgeInsets.fromLTRB(0, 20, 0, 20),
165   child: ElevatedButton(
166     onPressed: _gorevUret,
167     child: Text('Ekle'),
168   ),
169 )

```

Kod 141. Ekle düğmesi kodları

Kod 142'de yeni bir görev eklenmesi için **Grevler** bileşenindeki **\_gorevEkle** metoduna çağrı yapan **\_gorevUret** metodunun kodları yer almaktadır. Bu kodlarda başlık, tarih ve simge nesneleri hazırlanarak ilgili metoda parametre olarak gönderilmektedir. **\_gorevUret** metodu öncelikle yeni görevle ilgili açıklama metnin girilme durumunu kontrol etmektedir. Bu amaçla **\_baslikKontrolcusu** nesnesindeki metnin boş olma durumu incelenir. Metin boşsa ekrana hata mesajı çikaran **\_hataBildir** metoduna çağrı yapılır. Aksi takdirde **\_kontrolcudeki** içerik **\_baslik** değişkenine aktarılır.

Görev başlığının hazırlanmasının ardından görev ile ilgili simge bileşeninin hazırlanmasına geçilir. Simge bileşeni içeriği **DropdownButton** nesnesinden gelecek değere göre belirleneceğinden bir **switch case** bloğu tanımlanmıştır. **DropdownButton** bileşeninden gelecek değer temel alınarak simgenin görüntüsü (Ör: **Icons.school**), rengi (Ör: **Colors.amber**) ve boyutu (Ör: **32**) ayarlanmaktadır. Okul, ev ve iş için **case** durumları tanımlanmıştır. **DropdownButton** nesnesinden gelebilecek dördüncü değer eğlence olduğundan bu değer için **default** bloğu kullanılmıştır. Ekran görüntüleri incelendiğinde simge yüzlerinin ve renklerinin değiştiği görülebilir. Okul için **Icons.school**, **Colors.red**; iş için **Icons.work**, **Colors.amber**; ev için **Icons.home**, **Colors.green**; eğlence içinse **Icons.videogame\_asset**, **Colors.blueAccent** değerleri kullanılmıştır.

Tüm içerikler **widget.\_gorevEkleyici** referansı yoluyla **Grevler** bileşeni altında tanımlı **\_gorevEkle** metoduna gönderilmektedir. **widget** referansı **StatefulWidget** nesnelerindeki sınıfı işaret etmektedir. Burada kodların **\_YeniGorevState** sınıfı içinde hazırlandığı unutulmamalıdır. **\_gorevEkleyici**, **YeniGorev** sınıfı altında tanımlanmış bir değişken olduğundan bu değişkene erişebilemek için **widget.\_gorevEkleyici** referansının kullanılması gerekmektedir. Bu çağrı ile **Grevler** bileşeni altındaki **\_gorevEkle** metoduna gerekli veri türlerinde parametreler aktarılırak yeni bir görevin üretilmesi sağlanmaktadır. Bu görev ilgili metod altında **setState** fonksiyonu içinde eklendiğinden arayüzde gerekli değişiklikler otomatik olarak yapılacaktır.

Yeni görev paneli ekrana **showModalBottomSheet** metodu ile eklenmiştir. Bu yolla ekran örtülerek yalnızca panel ile etkileşim sağlanması amaçlanmıştır. Metottaki son satırda, ekleme görevi sonunda panelin **Navigator** nesnesinin **pop()** metodu ile (son eklenen nesnenin kaldırılması) panelin ekranдан kaldırılması sağlanmaktadır.

```

34  _gorevUret() {
35      if (_baslikKontrolcusu.text.isEmpty) {
36          _hataBildir("Lütfen başlık bilgisi giriniz.");
37          return;
38      } else {
39          _baslik = _baslikKontrolcusu.text;
40      }
41      Icon? simgem;
42      switch (_simge) {
43          case "Okul":
44              simgem = Icon(
45                  Icons.school,
46                  color: Colors.red,
47                  size: 32,
48              );
49              break;
50          case "Ev":
51              ...
52              break;
53          case "İş":
54              ...
55              break;
56          default:
57              ...
58      }
59      widget._gorevEkleyici(baslik: _baslik, tarih: _tarih, simge: simgem);
60      Navigator.of(context).pop();
61  }

```

Kod 142. `_gorevEkle` metodu kodları

`_hataBildir` metodu bir **AlertDialog** bileşeni üretilmektedir. **AlertDialog** bileşenleri arayüzün tüm etkileşimini kapatarak mesaj verilmesi ve gerekli hallerde kullanıcıdan Onay ya da İptal bilgisi alınmasını sağlayabilmektedir. Ekran Görüntüsü 40'ta, Kod 143'te oluşturulmuş **AlertDialog** bileşeninin görüntüsü sunulmuştur.

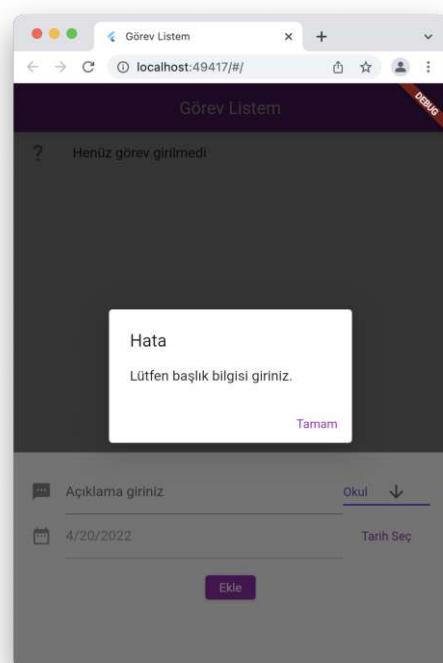
Kod 143'te 76-85 numaralı satırlar arasında bir **AlertDialog** bileşeni üretilmektedir. **\_hataBildir** metodu **String** tipinde bir **mesaj** parametresi almaktadır. Bu parametre, 78 numaralı satırda diyalog kutusunun içerik alanında gösterilmek üzere **content** parametresine gönderilmiştir. 77 numaralı satırda **Hata** ifadesinin gönderildiği **title** parametresi ise diyalog kutusunun başlığını oluşturmaktadır. Üretilen **AlertDialog** bileşeni yalnızca bilgilendirme amacıyla güttüğünden bu bileşende tek bir **Tamam** düğmesi oluşturulmuştur. Bu düğme 79 numaralı satırda **actions** parametresine atanmıştır. **actions** parametresi bir **Widget** listesi kabul etmektedir. Bu listeye eknecek bileşenler diyalog kutusunun alt bölümünde gösterilecektir. Düğmelerin **onPressed** parametrelerine atanan anonim fonksiyonlarda istenen işlemler gerçekleştirilebilir. 81 numaralı satırda **AlertDialog** kutusunu **Navigator** nesnesinden (ekrandan kaldırın) çıkarılan **pop** metodu çalıştırılmaktadır.

Üretilen diyalog kutusu 86-91 numaralı satırlar arasında **showDialog** metodu ile ekrana gönderilmektedir.

```

75 _hataBildir(String mesaj) {
76   final hataMesaji = AlertDialog(
77     title: const Text('Hata'),
78     content: Text(mesaj),
79     actions: <Widget>[
80       TextButton(
81         onPressed: () => Navigator.pop(context, 'Tamam'),
82         child: const Text('Tamam'),
83       ),
84     ],
85   );
86   showDialog(
87     context: context,
88     builder: (BuildContext context) {
89       return hataMesaji;
90     },
91   );
92 }
```

Kod 143. **\_hataBildir** metodu kodları



Ekrani Görüntüsü 40. *AlertDialog* bileşeni

## 2. TASARLA ve ÜRET

Basit bir iletişim listesi uygulaması üretiniz. Bu uygulamada kişilerin ad soyadlarını, telefon numaralarını, işyeri/okul bilgilerini, e-postalarını, not ortalamalarını ve doğum tarihlerini alınız.

## 3. DEĞERLENDİR

Öğrencilerin eğitim sonundaki başındaki programlamaya yönelik tutumlarını belirlemek için <https://forms.gle/pwN8E2Lato53iue67> adresinde yer alan formu uygulayınız. Bu form öğrencilerin e-posta bilgilerini toplamaktadır. Eğitim başında uygulanan form ile aynıdır. Verilerin eşleştirilmesi amacıyla, öğrencilerinize her iki uygulamada da aynı e-posta adresini kullanmaları gerektiğini bildiriniz.

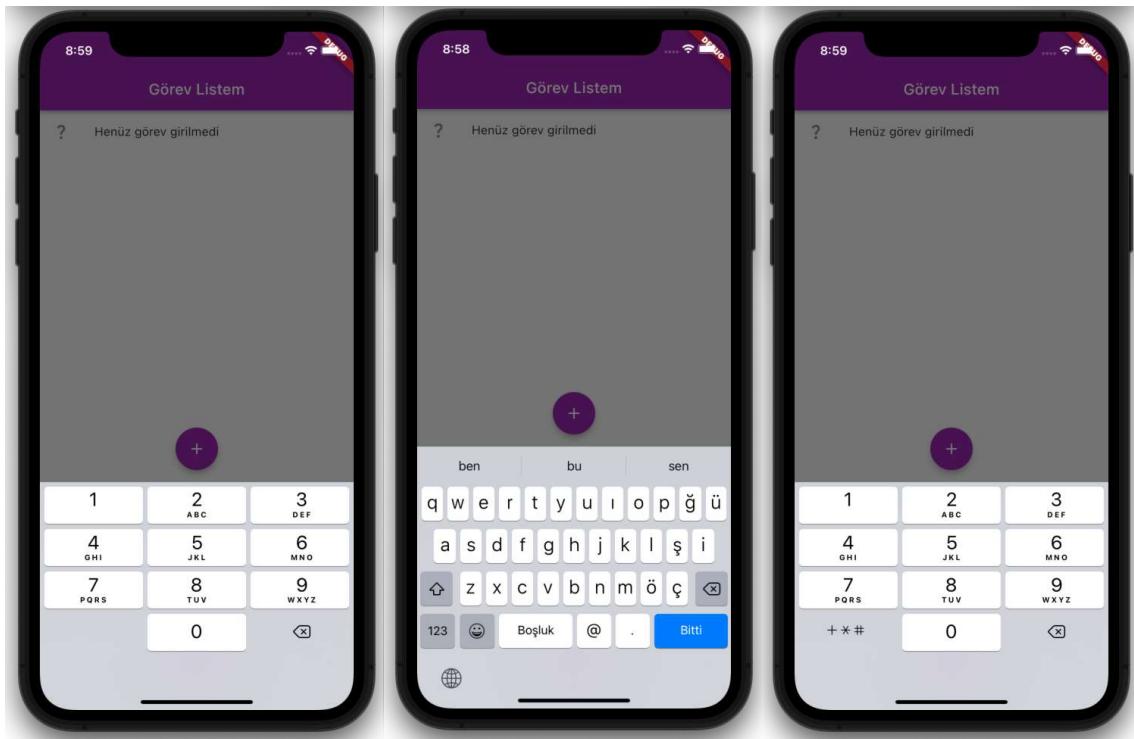
Öğrencinin özdeğerlendirme yapabilmesi için **Öz Değerlendirme Formu** (<https://forms.gle/kts381jqHWcCah2j6>) uygulanır.

### Proje Hazırlama

Öğrencilerin grup çalışmasına verdikleri katkılara yönelik özdeğerlendirme yapabilmeleri için **Grup Çalışması Öz Değerlendirme Formu’nu** (<https://forms.gle/FWk3AhU2sFX5heVK8>) uygulayınız.

## 4. İLAVE ETKİNLİK

Tasarla ve Üret etkinliğinde **TextInput** bileşeninin **keyboardtype** parametresinin işlevini inceleyiniz. Bu parametre mobil cihazlarda gösterilen sanal klavyenin türünü değiştirebilmektedir. Bu sayede alınacak bilgi türüne göre sayısal klavyeler ya da e-posta girişini kolaylaştıran klavyeler gösterilebilmektedir.



Ekran Görüntüsü 41. keyboardtype parametresi ile değiştirilen sanal klavyeler

## 5. KAYNAKÇA

Google. (t.y.). <https://docs.flutter.dev> adresinden 18.11.2021 tarihinde erişilmiştir.

# MOBİL UYGULAMA

## LİSE

