Mobile Translation.md

Chapter 4 : Değer Widget'ları

Son bölümde her şeyin bir widget olduğunu öğrendik. Bizler tarafından oluşturulan ve Flutter'ın bize sağladığı her şey birer widget'tır. Elbette, bunun istisnaları vardır, ancak özellikle Flutter'a başlarken bu şekilde düşünmekte bir sakınca yoktur. Bu bölümde, Flutter'ın bize sağladığı en temel widget grubunu inceleyeceğiz - değer tutanlar. Metin widget'ı, Simge widget'ı ve Görsel widget'ı hakkında konuşacağız, bunların hepsi tam olarak adlarının ima ettiği şeyi görüntüler. Daha sonra kullanıcıdan girdi almak için tasarlanmış olan girdi widget'larına giriş yapmış olacağız.

Metin Widget'ı

Eğer ekranda bir string görüntülemek istiyorsanız, Text widget'ı tam ihtiyacınız olan şey.

```
Text('Hello World'),
```

İpucu: *Metniniz bir değişmez ise, önüne const kelimesini koyun ve widget çalışma zamanı yerine derleme zamanında oluşturulacaktır. Apk/ipa dosyanız biraz daha büyük olacaktır ancak cihazda daha hızlı çalışacaktır. Buna değer.*

Metnin boyutu, yazı tipi, ağırlığı, rengi ve daha fazlası üzerinde değişiklikler yapabilirsiniz. Ancak bu konuyu Bölüm 8, “Pencere Öğelerinizi Stilize Etme” kısmında ele alacağız.

Simge Widget'ı

Flutter, kameralardan insanlara, kartlara, araçlara, oklara, pillere ve Android/iOS cihazlarına kadar zengin bir yerleşik simge setiyle birlikte gelir (Şekil 4-1). Tam listeyi burada bulabilirsiniz: https://api.flutter.dev/flutter/material/Icons-class.html.

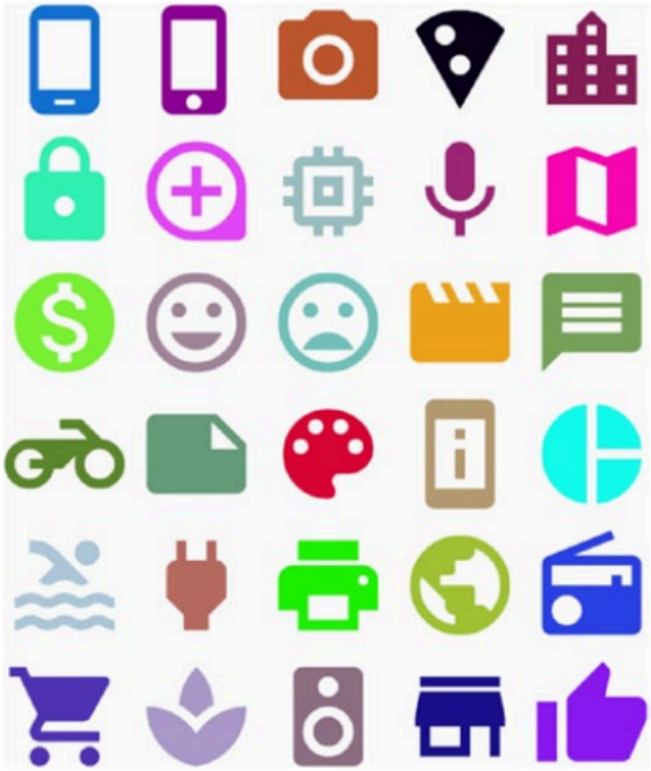

Figure 4-1. *An assortment of Flutter's built-in widgets in random colors*

Bir simge yerleştirmek için Simge widget'ını kullanırsınız. Herhangi bir sürpriz bulunmamakta. Kullanacağımız simgenin hangisi olduğunu belirtmek için Icons sınıfını kullanırsınız. Bu sınıfın Icons.phone_android ve Icons.phone_iphone ve Icons.cake gibi yüzlerce statik değeri vardır. Her biri, daha önce resmedilenler gibi farklı bir simgeye işaret eder. Uygulamanıza büyük kırmızı bir doğum günü pastasını (Şekil 4-2) şu şekilde yerleştirebilirsiniz:

```
Icon(
  Icons.cake,
  color: Colors.red,
  size: 200.0,
)
```

Figure 4-2. *The red cake icon*

Görsel Widget'ı

Flutter'da görüntüleri görüntülemek Metin veya Simgelerden biraz daha karmaşıktır. Birkaç yöntemi içerir:

1. Görüntü Kaynağını Alma : Bu, uygulamanın içine gömülü veya internetten canlı olarak alınmış bir görüntü olabilir. Görüntü, logo veya süslemeler gibi uygulamanızın ömrü boyunca hiç değişmeyecekse gömülü bir görüntü olmalıdır.

2. Boyutlandırma : Doğru boyut ve şekle göre yukarı veya aşağı ölçeklendirme.

Gömülü Görseller

Gömülü resimler çok daha hızlıdır ancak uygulamanızın yükleme boyutunu artıracaktır. Resmi yerleştirmek için, resim dosyasını proje klasörünüze, muhtemelen işleri düzenli tutmak için images adlı bir alt klasöre koyun. assets/ images gibi bir şey işinizi görecektir.

Ardından pubspec.yaml dosyasını düzenleyin. Bunu ekleyin:

```
flutter:
        assets:
                - assets/images/photo1.png
                - assets/images/photo2.jpg
```

Dosyayı kaydedin ve projenizin dosyayı işlemesini sağlamak için komut satırından “flutter pub get” komutunu çalıştırın.

İpucu: *pubspec.yaml dosyası projeniz hakkında her türlü harika bilgiyi tutar. Ad, açıklama, depo konumu ve sürüm numarası gibi proje meta verilerini tutar. Kütüphane bağımlılıklarını ve yazı tiplerini listeler. Projenize yeni katılan diğer geliştiriciler için gidilecek yerdir. Aranızdak, JavaScript geliştiricileri için örneklendirecek olursak; pubspec.yaml, Dart projenizin package.json dosyasıdır.*

Ardından asset() yapıcısını aşağıdaki gibi çağırarak görüntüyü özel widget'ınıza yerleştirirsiniz:

```
Image.asset('assets/images/photo1.jpg',),
```

Ağ Üzerinden Görseller

Ağ üzerinden görseller, daha çok web geliştiricilerinin alışık olduğu türdendir. Basitçe bir görüntüyü HTTP aracılığıyla İnternet üzerinden getirir. Ağ yapıcısını kullanacak ve bir URL'yi dize olarak aktaracaksınız.

```
Image.network(imageUrl),
```

Beklediğiniz gibi, bunlar gömülü görsellerden daha yavaştır çünkü istek İnternet üzerinden bir sunucuya gönderilirken ve görsel, cihazınız tarafından indirilirken bir gecikme olur. Avantajı ise bu görsellerin canlı olmasıdır; herhangi bir görsel, sadece görsel URL'si değiştirilerek dinamik olarak yüklenebilir.

Görselleri Boyutlandırma

Göseller neredeyse her zaman bir konteyner içine konur. Bu bir gereklilik olduğundan değil, sadece başka bir widget'ın içinde olmayacağı bir gerçek dünya kullanım durumu hayal edemiyorum. Konteyner, bir görselin çizileceği boyutta söz sahibidir. Eğer Görselin doğal boyutu konteynerin boyutuna mükemmel bir şekilde uysaydı, bu inanılmaz bir tesadüf olurdu. Bunun yerine, Flutter'ın düzen motoru görüntüyü kabına sığacak şekilde küçültür, ancak büyütmez. Bu uyum BoxFit. scaleDown olarak adlandırılır ve varsayılan davranış için mantıklıdır. Peki başka hangi seçenekler mevcut ve hangisini kullanacağımıza nasıl karar vereceğiz? Flutter, aşağıdaki BoxFit seçeneklerini sağlar:

- **fill** - Hem genişlik hem de yükseklik tam olarak sığacak şekilde uzatır. Görüntüyü bozar
- **cover** - Alan dolana kadar küçültür veya büyütür. Üst/alt veya yanlar kırpılir
- **fitHeight** - Yüksekliğin tam olarak sığmasını sağlar. Genişliği kırpar veya gerektiğinde fazladan boşluk ekler
- **fitWidth** - Genişliği uygun hale getirir. Yüksekliği kırpar veya gerektiğinde ekstra alan ekler

- **contain** - Hem yükseklik hem de genişlik sığana kadar küçültür. Üstte/altta veya yanlarda fazladan boşluk olacaktır

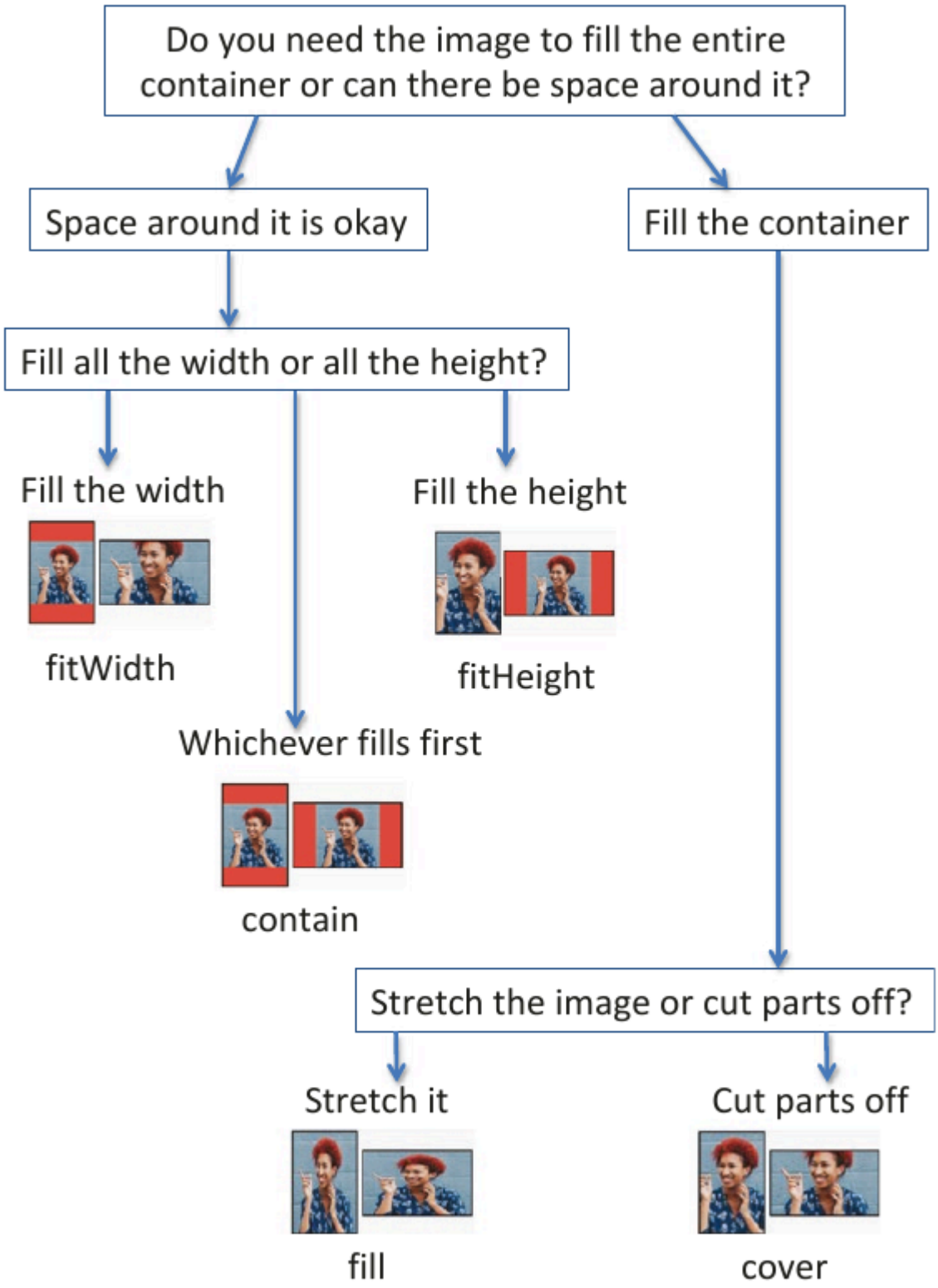

Figure 4-3. *How to decide an image's fit*

Sığdırmayı belirtmek için sığdırma özelliğini ayarlayacaksınız.

```
Image.asset(
        'assets/images/woman.jpg',
        fit: BoxFit.contain, )
```

Girdi Widget'ları

Birçoğumuz, en başından beri <input> ve <select> içeren HTML <form>'larının bulunduğu bir web geçmişinden geliyoruz. Tüm bunlar, kullanıcının web uygulamalarına veri girmesini sağlamak için var; mobil uygulamalarda da bu olmadan yaşayamayacağımız bir etkinlik. Flutter, Web'de olduğu gibi veri girmek için widget'lar sağlar, ancak bunlar aynı şekilde çalışmaz. Oluşturmak ve kullanmak için çok daha fazla çalışma gerekiyor. Bunun için üzgünüm. Ancak aynı zamanda daha güvenli ve bize çok daha fazla kontrol sağlıyorlar.

Karmaşıklığın bir kısmı, bu widget'ların kendi durumlarını korumamasıdır; bunu manuel olarak yapmanız gerekir.

Karmaşıklığın bir başka kısmı da girdi widget'larının birbirlerinden habersiz olmalarıdır. Başka bir deyişle, siz onları bir Form widget'ı ile gruplayana kadar birlikte iyi çalışmazlar. Sonunda Form widget'ına odaklanmamız gerekiyor. Ancak bunu yapmadan önce, metin alanlarının, onay kutularının, radyo düğmelerinin, kaydırıcıların ve açılır menülerin nasıl oluşturulacağını inceleyelim.

Dikkat: Bir StatefulWidget içinde kullanılmadıkları sürece girdi widget'ları ile çalışmak gerçekten zordur çünkü doğaları gereği durum değiştirirler. Geçen bölümde StatefulWidgets'tan kısaca bahsettiğimizi ve 9. Bölüm olan "Durum Yönetimi "nde bu konudan derinlemesine bahsedeceğimizi hatırlayın. Ancak o zamana kadar lütfen bizim sözümüze güvenin ve şimdilik onları durum bilgisine sahip bir widget'a yerleştirin.

TextField'lar

Sahip olduğunuz tek şey tek bir metin kutusuysa, muhtemelen bir TextField widget'ı istersiniz. İşte üzerinde bir Text etiketi bulunan TextField widget'ının basit bir örneği:

```
const Text('Search terms'),
TextField(
  onChanged: (String val) => _searchTerm = val,
)
```

Bu onChanged özelliği, her tuş vuruşundan sonra tetiklenen bir olay işleyicisidir. Tek bir değer alır - bir String. Bu, kullanıcının yazdığı değerdir. Önceki örnekte, _searchTerm adlı bir lokal değişkeni kullanıcı ne yazarsa ona ayarlıyoruz.

TextField ile bir başlangıç değeri sağlamak için, gereksiz derecede karmaşık TextInputController'a ihtiyacınız vardır:

```
TextEditingController _controller =
        TextEditingController(text: "Initial value here");
```

Ardından TextField'ınıza denetleyici hakkında bilgi verin.

```
const Text('Search terms'),
TextField(
        controller: _controller,
        onChanged: (String val) => _searchTerm = val,
),
```

Kullanıcının kutuya yazdığı değeri almak için _controller.text özelliğini de kullanabilirsiniz.

Text('Arama terimleri') ifadesini fark ettiniz mi? Bu, TextField'ın üzerine bir etiket koyma konusundaki zayıf girişimimiz. Çok çok daha iyi bir yolu var. Şuna bir göz atın ...

TextField'ınızı Süslendirme

TextField'ınızı daha kullanışlı hale getirmek için tonlarca seçenek var - sonsuz seçenek değil, ama çok sayıda. Ve bunların hepsi InputDecoration widget'ı aracılığıyla kullanılabilir (Şekil 4-4):

```
return TextField(
        controller: _emailController,
        decoration: InputDecoration(
                labelText: 'Email',
                hintText: 'you@email.com',
                icon: Icon(Icons.contact_mail),
        ),
),
```

Figure 4-4. *A TextField with an InputDecoration*

Tablo 4-2'de daha fazla InputDecoration seçeneği sunulmaktadır.

- **labelText** - TextField'ın üzerinde görünür. Kullanıcıya bu TextField'ın ne için olduğunu söyler.
- **hintText** - TextField içindeki hafif soluk metin. Kullanıcı yazmaya başladığında kaybolur.
- **errorText** - TextField'ın altında görünen hata mesajı. Genellikle kırmızıdır. Doğrulama tarafından otomatik olarak ayarlanır (daha sonra ele alınacaktır), ancak gerekirse manuel olarak ayarlayabilirsiniz.
- **prefixText** - Kullanıcının yazdığı şeylerin solundaki TextField içindeki metin.

- **suffixText** - prefixText ile aynı ama en sağda.
- **icon** - TextField'ın tamamının soluna bir simge çizer.
- **prefixIcon** - Sol taraftaki TextField'ın içine bir tane çizer.
- **suffixIcon** - prefixIcon ile aynıdır ancak en sağdadır.

İpucu: *Bunu bir parola kutusu yapmak için (Şekil 4-5), obscureText özelliğini true olarak ayarlayın. Kullanıcı yazdıkça, her karakter bir saniyeliğine görünür ve bir nokta ile değiştirilir.*

```
return TextField(
        obscureText: true,
        decoration: InputDecoration(
                labelText: 'Password',
        ),
);
```

Figure 4-5. *A password box with obscureText*

Özel bir soft klavye mi istiyorsunuz? Hiç sorun değil. Sadece keyboardType özelliğini kullanın. Sonuçlar Şekil 4-6 ila 4-9'da gösterilmektedir.

```
return TextField(
        keyboardType: TextInputType.datetime,
);
```

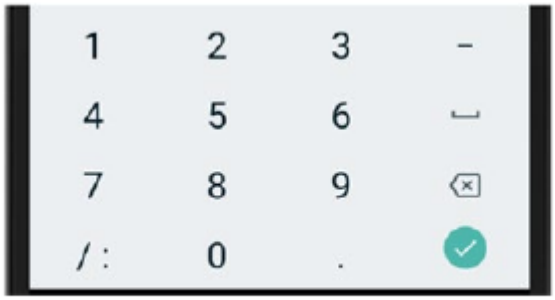

Figure 4-6. *TextInputType.datetime*

```
return TextField(
        keyboardType: TextInputType.email,
);
```

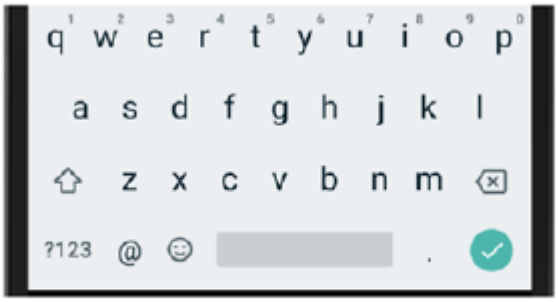

Figure 4-7. *TextInputType.email. Note the @ sign*

```
return TextField(
        keyboardType: TextInputType.number,
);
```

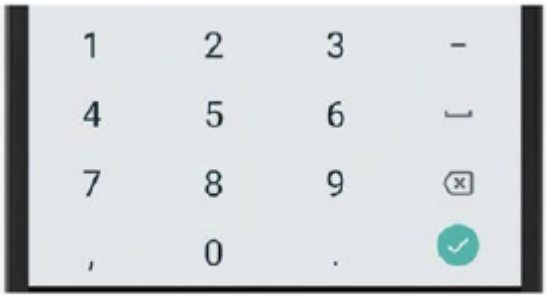

Figure 4-8. *TextInputType.number*

```
return TextField(
        keyboardType: TextInputType.phone,
);
```

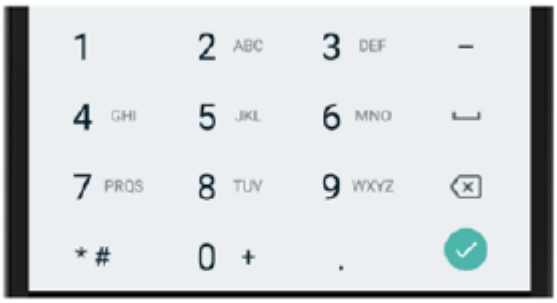

Figure 4-9. *TextInputType.phone*

İpucu: *Girilmesine izin verilen metin türünü sınırlamak istiyorsanız, bunu TextInput'un inputFormatters özelliği ile yapabilirsiniz. Aslında bir dizidir, böylece aşağıdakilerden birini veya daha fazlasını kullanabilirsiniz...*

- ***BlacklıstıngTextInputFormatter*** *- Belirli karakterlerin girilmesini yasaklar. Kullanıcı yazdığında görünmezler.*

- ***WhitelistingTextInputFormatter*** *- Yalnızca bu karakterlerin girilmesine izin verir. Bu listenin dışındaki hiçbir şey görünmez.*
- ***LengthLimitingTextInputFormatter*** *- X karakterden fazla yazılamıyor.*

Bu ilk ikisi, istediğiniz (beyaz liste) veya istemediğiniz (kara liste) kalıpları belirtmek için düzenli ifadeleri kullanmanıza olanak tanır. İşte bir örnek:

```
return TextField(
        inputFormatters: [
                WhitelistingTextInputFormatter(RegExp('[0-9-]')),
                LengthLimitingTextInputFormatter(16)
        ];
        decoration: InputDecoration(
                labelText: 'Credit Card',
        ),
);
```

WhitelistingTextInputFormatter'da yalnızca 0-9 arası sayılara, bir boşluğa veya bir tire işaretine izin veriyoruz. Ardından LengthLimitingTextInputFormatter en fazla 16 karaktere izin veriyor.

Onay Kutuları

Flutter onay kutuları (Şekil 4-10) bir boolean değer özelliğine ve her değişiklikten sonra ateşlenen bir onChanged yöntemine sahiptir. Diğer tüm girdi widget'ları gibi, onChanged yöntemi de kullanıcının ayarladığı değeri alır. Dolayısıyla, Onay Kutuları söz konusu olduğunda, bu değer bir bool'dur.

```
Checkbox(
        value: true,
        onChanged: (bool val) => print(val)),
```

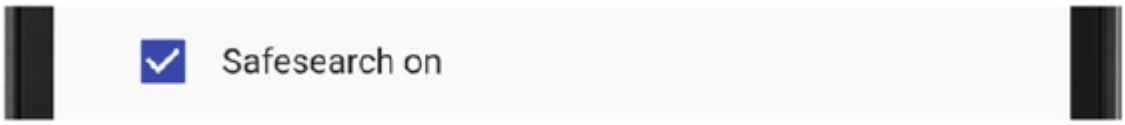

Figure 4-10. *A Flutter Checkbox widget*

İpucu: *Bir Flutter Anahtarı (Şekil 4-11), bir Onay Kutusu ile aynı amaca hizmet eder - açık veya kapalıdır. Yani Switch widget'ı aynı seçeneklere sahiptir ve aynı şekilde çalışır. Sadece farklı görünüyor.*

Figure 4-11. *A Flutter Switch widget*

Radyo Butonları

Elbette bir radyo butonundaki sihir, birini seçtiğinizde aynı gruptaki diğerlerinin seçiminin kaldırılmasıdır. Açıkçası onları bir şekilde gruplandırmamız gerekiyor. Flutter'da, groupValue özelliğini aynı yerel değişkene ayarladığınızda Radyo widget'ları gruplanır. Bu değişken, o anda açık olan bir Radyonun değerini tutar.

Her Radyo ayrıca, seçili olsun ya da olmasın söz konusu widget ile ilişkili değer olan kendi değer özelliğine sahiptir. onChanged yönteminde, groupValue değişkenini radyonun değerine ayarlayacaksınız:

```
SearchType _searchType;
// Other code goes here

Radio<SearchType>(
  groupValue: _searchType,
  value: SearchType.anywhere,
  onChanged: (SearchType val) => _searchType = val,
),
const Text('Search anywhere'),

Radio<SearchType>(
  groupValue: _searchType,
  value: SearchType.text,
  onChanged: (SearchType val) => _searchType = val,
),
const Text('Search page text'),

Radio<SearchType>(
  groupValue: _searchType,
  value: SearchType.title,
  onChanged: (SearchType val) => _searchType = val,
),
const Text('Search page title'),
```

Bu basitleştirilmiş kod Şekil 4-12'deki gibi bir şey oluşturacaktır.

Figure 4-12. *Flutter Radio widgets*

Kaydırıcılar

Kullanıcınızın bir üst ve alt sınır arasında sayısal bir değer seçmesini istediğinizde kaydırıcı kullanışlı bir olanaktır (Şekil 4-13).

Figure 4-13. *A slider with the value of 25*

Flutter'da bir tane elde etmek için, bir onChanged olayı ve bir double değer özelliği gerektiren Slider widget'ını kullanacaksınız. Ayrıca varsayılan olarak 0,0 olan bir min ve varsayılan olarak 1,0 olan bir maks değerine sahiptir. Sıfırdan bire kadar olan bir aralık nadiren kullanışlıdır, bu nedenle genellikle bunu değiştirirsiniz. Ayrıca, kullanıcıya hangi değeri seçtiğini söyleyen bir gösterge olan label özelliği de vardır.

```
Slider(
        label: _value.toString(),
        min: 0, max: 100,
        divisions: 100,
        value: _value,
        onChanged: (double val) => _value = val,
),
```

Açılır Menüler

Açılır listeler, bir numaralandırmada olduğu gibi az sayıda şeyden birini seçmek için harikadır. Diyelim ki şöyle bir enumumuz var:

```
enum SearchType { web, image, news, shopping }
```

"SearchType "ı 'web', 'resim', 'haber' ya da 'alışveriş' olarak tanımladığımız açıktır. Kullanıcımızın bunlardan birini seçmesini isteseydik, başlangıçta Şekil 4-14'teki gibi görünebilecek bir DropdownButton widget'ı sunabilirdik.

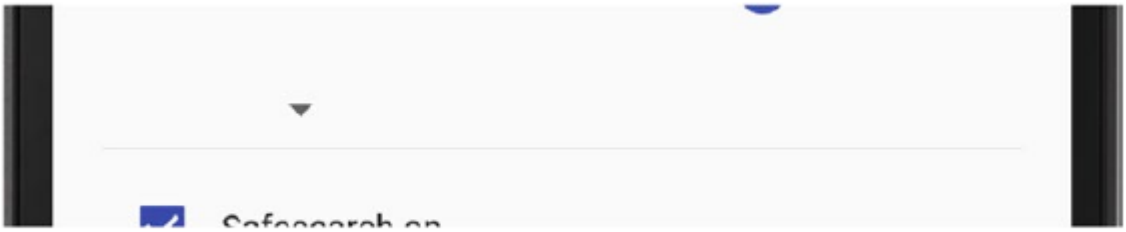

Figure 4-14. *DropdownButton with nothing chosen*

Ardından, açılır menüye dokunduklarında Şekil 4-15'teki gibi görünür.

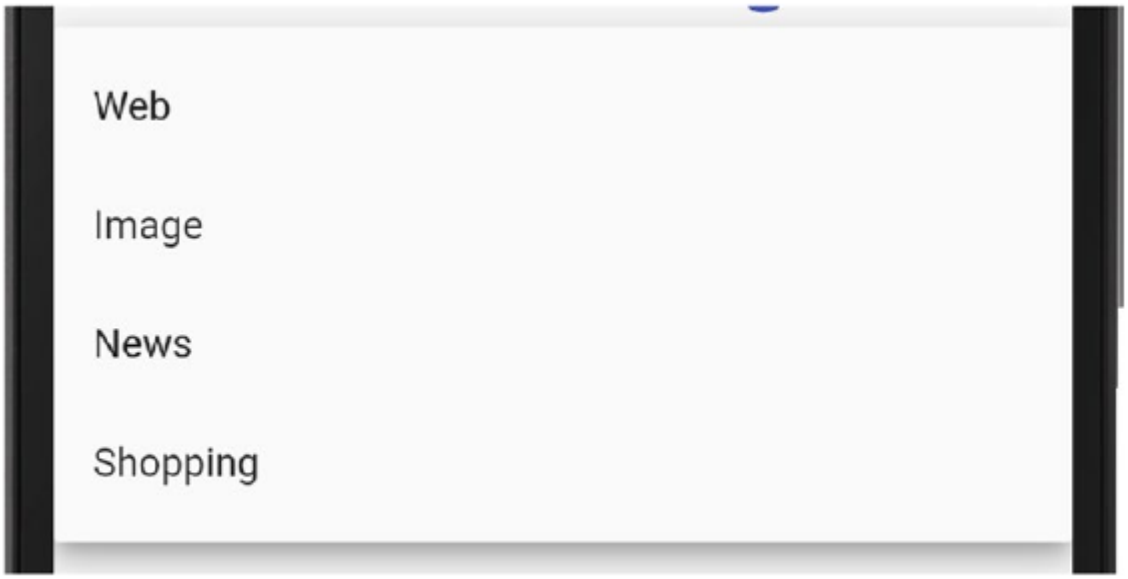

Figure 4-15. *DropdownButton expanded to show the choices*

Ve seçeneklerden birine dokunduklarında, o seçenek seçilir (Şekil 4-16).

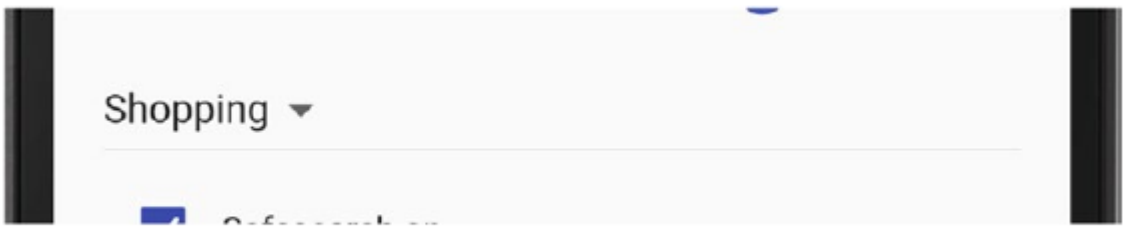

Figure 4-16. *DropdownButton with an option selected*

Bu DropdownButton'ı oluşturmak için Flutter kodumuz aşağıdaki gibi görünebilir:

```
SearchType _searchType = SearchType.web;
// Other code goes here

DropdownButton<SearchType>(
  value: _searchType,
  items: const <DropdownMenuItem<SearchType>>[
    DropdownMenuItem<SearchType>(
      child: Text('Web'),
      value: SearchType.web,
    ),
    DropdownMenuItem<SearchType>(
      child: Text('Image'),
      value: SearchType.image,
    ),
    DropdownMenuItem<SearchType>(
      child: Text('News'),
      value: SearchType.news,
    ),
    DropdownMenuItem<SearchType>(
      child: Text('Shopping'),
      value: SearchType.shopping,
    ),
  ],
  onChanged: (SearchType val) => _searchType = val,
)
```

Form Widget'larını Bir Araya Getirme

İyi görünen ve harika çalışan tüm bu farklı alan türlerine sahip olmamız harika. Ancak, grup olarak bir şekilde kontrol edilebilmeleri için genellikle bunların birlikte gruplanmasını istersiniz. Bunu bir Form widget'ı ile yapacaksınız.

Form Widget'ı

HTML'de olduğu gibi, Form widget'ı olmadan da gayet iyi yaşayabilirsiniz. Görsel bileşeni olmayan kullanışlı bir widget'tır. Yani aslında hiçbir zaman cihaz üzerinde işlendiğini görmezsiniz. Tek amacı, tüm girdilerini sarmak ve böylece onları - ve verilerini - bir birim halinde gruplandırmaktır. Bunu bir anahtar kullanarak yapar. Son bölümde anahtarları tanıttığımızı ve birkaç durum dışında anahtarların göz ardı edilebileceğini söylediğimizi hatırlayın. Bu, anahtarlara ihtiyaç duyulan bir yerdir. Eğer bir Form kullanmaya karar verirseniz, FormState tipinde bir GlobalKey'e ihtiyacınız olacaktır:

```
GlobalKey<FormState> _key = GlobalKey<FormState>();
```

Bu anahtarı formunuza bir özellik olarak ayarlayacaksınız:

```
@override
Widget build(BuildContext context) {
        return Form(
                key: _key,
                autovalidate: true,
                child: // All the form fields will go here
        );
}
```

İlk bakışta, Form hiçbir şeyi değiştirmiyor gibi görünüyor. Ancak daha yakından bakıldığında, artık aşağıdakilere erişimimiz olduğu ortaya çıkıyor

- **autovalidate** - Bir bool. True, herhangi bir alan değişir değişmez doğrulamaları çalıştırmak anlamına gelir. False, manuel olarak çalıştıracağınız anlamına gelir. (Doğrulamalar hakkında birkaç sayfa sonra konuşacağız).
- Önceki örnekte _key olarak adlandırdığımız anahtarın kendisi. Bu _key'in bir currentState özelliği vardır ve bu da aşağıdaki yöntemlere sahiptir:
    i. **save()** - Her birinin onSaved öğesini çağırarak form içindeki tüm alanları kaydeder.
    ii. **validate()** - Her alanın doğrulama işlevini çalıştırır.
    iii. **reset()** - Form içindeki her alanı initialValue'ya geri sıfırlar

Tüm bunlarla birlikte, Form'un iç içe geçmiş alanları nasıl gruplandırdığını tahmin edebilirsiniz. FormState üzerinde bu üç yöntemden birini çağırdığınızda, iç alanları yineler ve her birinde o yöntemi çağırır. Form düzeyindeki tek bir çağrı hepsini ateşler.

Ama bir saniye bekleyin! Eğer _key.currentState.save() bir alanın onSaved() metodunu çağırıyorsa, bir onSaved metodu sağlamamız gerekir. Validator'ı çağıran validate() ile aynı. Ancak TextField, Dropdown, Radio, Checkbox ve Slider widget'larının kendileri bu yöntemlere sahip değildir. Şimdi ne yapacağız? Her alanı, bu yöntemlere sahip olan bir FormField widget'ına sarıyoruz. (Ve tavşan deliği daha da derinleşir.)

FormField Widget'ları

Bu widget'ın hayattaki tüm amacı, bir iç widget'a kaydetme, sıfırlama ve doğrulayıcı olay işleyicileri sağlamaktır. FormField bileşeni, bir oluşturucu özelliği kullanarak herhangi bir bileşeni sarabilir:

```
FormField<String>(
        builder: (FormFieldState<String> state) {
                return TextField(); // Any field widget like DropDownButton,
                // Radio, Checkbox, or Slider.
        },
        onSaved: (String initialValue) {
                // Push values to a repository or something here.
```

```
        },
        validator: (String val) {
                // Put validation logic here (further explained below).
        },
),
```

Bu yüzden önce her girdi widget'ının etrafına bir FormField widget'ı sarıyoruz ve bunu builder adlı bir yöntemle yapıyoruz. Ardından onSaved ve validator yöntemlerini ekleyebiliriz.

İpucu: *Bir TextField'a farklı davranın. Bir Form içinde kullanıyorsanız, onu sarmak yerine bir TextFormField widget'ı ile değiştirin. Bu yeni widget'ı TextField ile karıştırmak kolaydır ancak farklıdır. Temel olarak ...*

```
TextFormField = TextField + FormField
```

Flutter ekibi, bir TextField widget'ına bir FormField widget'ı ile birlikte rutin olarak ihtiyaç duyacağımızı biliyordu, bu nedenle bir TextField'ın tüm özelliklerine sahip olan ancak bir onSaved, validator ve reset ekleyen TextFormField widget'ını oluşturdular:

```
TextFormField(
        onSaved: (String val) {
                print('Search Term TextField: form saved $val');
        },
        validator: (String val) {
                // Put your validation logic here
        },
),
```

Şimdi daha güzel değil mi? Sonunda işleri kolaylaştırma konusunda bir mola yakaladık. Onay kutuları bu özelliğe sahip değildir. Ne Radyolar ne de Açılır Menüler. TextFields dışında hiçbiri.

En iyi uygulama: Form içermeyen metin girişleri her zaman bir TextField olmalıdır. Form içindeki metin girişleri her zaman bir TextFormField olmalıdır.

onSaved

Lütfen Formunuzun bir save() metodu olan bir currentState'e sahip bir anahtarı olduğunu unutmayın. Hepsini anladınız mı? Hayır mı? Çok net değil mi? Şöyle deneyelim; bir "Kaydet" düğmesine basıldığında, kodunuzu ... çağıracak şekilde yazacaksınız.

```
_key.currentState.save();
```

... ve bu da bir tane olan her Form Alanı için onSaved yöntemini çağırır.

validator

Benzer şekilde, muhtemelen arayabileceğinizi tahmin etmişsinizdir ...

```
_key.currentState.validate();
```

... ve Flutter her FormField'ın validator metodunu çağıracaktır. Ama dahası da var! Formun autovalidate özelliğini true olarak ayarlarsanız, kullanıcı değişiklik yaptığında Flutter hemen doğrulama yapar.

Her validator fonksiyonu bir değer (doğrulanacak değer) alır ve bir dize döndürür. Girdi değeri geçerliyse null, geçersizse gerçek bir dize döndürecek şekilde yazacaksınız. Dönen bu dize, Flutter'ın kullanıcınıza göstereceği hata mesajıdır.

Yazarken validate

Anında doğrulama yapmanın yolunun Form.autovalidate öğesini true olarak ayarlamak ve TextFormField için bir doğrulayıcı yazmak olduğunu unutmayın:

```
return Form(
        autovalidate: true,
        child: Container(
        TextFormField(
                validator: (String val) {
                        // Let's say that an empty value is invalid.
                        if (val.isEmpty)
                                return 'We need something to search for';
                        return null;
                        },
                ),
        ),
);
```

Açıkçası, bir DropdownButton, Radio, Checkbox, Switch veya Slider'ı yazarken doğrulamanın bir anlamı yoktur çünkü içlerine yazmazsınız. Ancak daha az açık bir şekilde, bir FormField içindeki bir TextField ile çalışmaz. Yalnızca bir TextFormField ile çalışır. Garip, değil mi?

İpucu: *Yine, en iyi uygulama bir TextFormField kullanmaktır. Ancak FormField içinde bir TextField kullanmakta ısrar ediyorsanız, errorText'i şu şekilde kaba kuvvetle ayarlayabilirsiniz:*

```
FormField<String>(
  builder: (FormFieldState<String> state) {
```

```
      return TextField(
        controller: _emailController,
        decoration: InputDecoration(
          // This says if the value looks like an email
          // set errorText to null. If not, display an error message.
          errorText: RegExp(r'^[a-zA-Z0-9.]+@[a-zA-Z0-9]+\.[a-zA-Z]+')
                  .hasMatch(_emailController.text)
              ? null
              : "That's not an email address",
        ),
      );
    },
  )
```

Form gönderme denemesinden sonra validate

Kullanıcı veri girmeyi bitirene kadar kodunuzun doğrulanmasını istemediğiniz zamanlar olabilir. Önce autovalidate'i false olarak ayarlamalısınız. Ardından düğmenin basılı olayında validate() işlevini çağırın:

```
  RaisedButton(
                  child: const Text('Submit'),
                  onPressed: () {
                  // If every field passes validation, run their save methods.
                  if (_key.currentState.validate()) {
                          _key.currentState.save();
                          print('Successfully saved the state.')
                  }
          },
  )
```

Bir Büyük Form Örneği

Biliyorum, biliyorum. Bu oldukça karmaşık bir konu. Bunları bağlam içinde görmek yardımcı olabilir - hepsinin birbirine nasıl uyduğunu. Aşağıda tamamen yorumlanmış bir örnek bulacaksınız... büyük bir örnek. Ama ne kadar büyük olursa olsun, aslında çok daha büyüktü. Lütfen tam örnek için çevrimiçi kaynak kodu depomuza bakın. Umarım Form alanlarının nasıl ilişkili olduğunu anlamanıza yardımcı olurlar.

Diyelim ki kullanıcının Google benzeri bir web araması göndermesi için bir sahne oluşturmak istedik. Arama dizesi için bir TextFormField, arama türünü içeren bir DropdownButton, safeSearch'ü etkinleştirmek/devre dışı bırakmak için bir onay kutusu ve göndermek için bir düğme vereceğiz:

```
  enum SearchType { web, image, news, shopping }

  class ProperForm extends StatefulWidget {
```

```
  @override
  _ProperFormState createState() => _ProperFormState();
}

class _ProperFormState extends State<ProperForm> {
  // A Map (aka. hash) to hold the data from the Form.
  final Map<String, dynamic> _searchForm = <String, dynamic>{
    'searchTerm': '',
    'searchType': SearchType.web,
    'safeSearchOn': true,
  };

  // The Flutter key to point to the Form
  final GlobalKey<FormState> _key = GlobalKey();

  @override
  Widget build(BuildContext context) {
    return Form(
      key: _key,
      // Make autovalidate true to validate on every keystroke. In
      // this case we only want to validate on submit.
      // autovalidate: true,
      child: Container(
        child: ListView(
          children: <Widget>[
            TextFormField(
              initialValue: _searchForm['searchTerm'],
              decoration: InputDecoration(
                labelText: 'Search terms',
              ),
              // On every keystroke, you can do something.
              onChanged: (String val) {
                setState(() => _searchForm['searchTerm'] = val);
              },
              // When the user submits, you could do something for this field
              onSaved: (String val) {
                _searchForm['searchTerm'] = val;
              },
              // Called when we "validate()". The val is the String in the text b
              validator: (String val) {
                if (val.isEmpty) {
                  return 'We need something to search for';
                }
                return null;
              },
            ),
            FormField<SearchType>(
              builder: (FormFieldState<SearchType> state) {
                return DropdownButton<SearchType>(
                  value: _searchForm['searchType'],
                  items: const <DropdownMenuItem<SearchType>>[
                    DropdownMenuItem<SearchType>(
                      child: Text('Web'),
```

```
                value: SearchType.web,
              ),
              DropdownMenuItem<SearchType>(
                child: Text('Image'),
                value: SearchType.image,
              ),
              DropdownMenuItem<SearchType>(
                child: Text('News'),
                value: SearchType.news,
              ),
              DropdownMenuItem<SearchType>(
                child: Text('Shopping'),
                value: SearchType.shopping,
              ),
            ],
            onChanged: (SearchType val) {
              setState(() => _searchForm['searchType'] = val);
            },
          );
        },
        onSaved: (SearchType val) {
          _searchForm['searchType'] = val;
        },
      ),
      FormField<bool>(
        builder: (FormFieldState<bool> state) {
          return Row(
            children: <Widget>[
              Checkbox(
                value: _searchForm['safeSearchOn'],
                // Every time it changes, you can do something.
                onChanged: (bool val) {
                  setState(() => _searchForm['safeSearchOn'] = val);
                },
              ),
              const Text('Safesearch on'),
            ],
          );
        },
        onSaved: (bool val) {
          _searchForm['safeSearchOn'] = val;
        },
      ),
      // This is the 'Submit' button
      ElevatedButton(
        child: const Text('Submit'),
        onPressed: () {
          // If every field passes validation, let them through.
          // Remember, this calls the validator on all fields in the form.
          if (_key.currentState.validate()) {
            // Similarly this calls onSaved() for all fields
            _key.currentState.save();
            // You'd save the data to a database or whatever here
```

```
                    print('Successfully saved the state.');
                  }
                },
              )
            ],
          ),
        ),
      );
    }
  }
```

Kapanış

Flutter formlarını anlamak biraz zaman alır. Lütfen cesaretiniz kırılmasın. Önceki örneğe birkaç kez daha bakın ve biraz kod yazın. Her şey çok hızlı bir şekilde anlam kazanmaya başlıyor. Formlar konusu sizin için biraz göz korkutucu olsa da, Görüntüler, Simgeler ve Metin çok basitti, değil mi?

Bir sonraki bölümde, uygulamamızın canlandığını görmeye başlayacağız çünkü tüm farklı düğme türlerini oluşturmayı ve bunları - ya da herhangi bir widget'ı - dokunmalara ve diğer hareketlere yanıt vermeyi öğreneceğiz!