# BÖLÜM

**Navigasyon ve**

**Yönlendirme**

Tüm uygulamalar bir ekrandan diğerine geçme konseptine sahiptir.

Kullanıcı sepet butonuna tıklar ve kart ekranına gideriz. Kullanıcı "alışverişe devam et" düğmesini tıklar ve satın almak için daha fazla ürüne göz atarız. Bazı uygulama geliştiricileri buna yönlendirme diyor.

Diğerleri buna navigasyon diyor.

Ne derseniz deyin, bu Flutter'ın gerçekten kolaylaştırdığı bir alandır çünkü gezinmenin yalnızca dört yolu vardır:

* Yığınlar – Her widget tam ekrandır. Kullanıcı, önceden tanımlanmış bir iş akışından geçmek için bir düğmeye dokunur. Tarih korunur ve bir geri düğmesine basarak bir seviye geri gidebilirler.
* Çekmeceler – Ekranın çoğu bir widget gösterir, ancak sol kenarda bir çekmece kullanıcıya göz atıyor. Bastıklarında veya sağa kaydırdıklarında, bir seçenekler menüsü ortaya çıkarak dışarı kayar. Birine basmak, ekranın ana bölümündeki widget'ı değiştirir.
* Sekmeler – Ekranın üst veya alt kısmında bir dizi sekme için bir miktar yer ayrılmıştır . Bir sekmeye bastığınızda, o sekmeye karşılık gelen widget'ı gösteririz.

© Rap Payne 2019 139

R. Payne, *Flutter ile Uygulama Geliştirmeye*

*Başlama*, https://doi.org/10.1007/978-1-4842-51812\_7

• İletişim kutuları – Bunlar teknik olarak gezinmenin bir parçası olmasa da, başka bir widget'ı görmenin bir yoludur, bu yüzden buna izin vereceğiz. İletişim kutuları kalıcıdır (diğer bir deyişle açılır pencereler) ve kullanıcı tarafından kapatılana kadar kalır.

Bu yöntemlerin her biri, uygulamanızın atası olarak bir

MaterialWidget'a sahip olmasına bağlıdır. Yığın gezintisinden başlayarak bunların detayına gidelim.

# Yığın gezintisi

Deneyimli bir geliştiriciyseniz, kuyruklar ve yığınlar hakkında bilgi sahibisinizdir. Değilse, kısaca açıklayayım. Diyelim ki bir mutfakta çalışıyorsunuz. Tabaklar temizlenirken üst üste dizilir, değil mi? Her plaka yığının üstüne konur.

Buna yığının üzerine itme denir. Biraz yemek servis etme zamanı geldiğinde, doğal olarak eklenen son tabağı, yığının üstündeki tabağı alırsınız. Buna *yığının üstünden* patlatma denir.

Flutter'ın navigasyonu yığınlarla çalışır. Kullanıcıyı yeni bir sahneye göndermek istediğinizde, yığının en üstüne bir widget iteceksiniz() ve kullanıcı bu widget'ı görecek. Her itişinizde (), sahne yığınını daha da uzun hale getiriyorsunuz. Daha önce bulundukları yere geri dönmeleri için hazır olduğunuzda, son sahneyi yığının en üstüne çıkarırsınız() ve ne ortaya çıkar? Bir önceki sahne.

Flutter'ın yığınıyla, genellikle sahneleri (diğer adıyla rotaları) önceden tanımlayacak ve her birine bir ad vereceksiniz. Bu, MaterialApp düzeyinde şu şekilde yapılmalıdır:

Widget build(BuildContext context) { return MaterialApp(title: 'Alışveriş Uygulaması', initialRoute: '/', rotalar: {'/': (BuildContext ctx) => LandingScene(),

'/browse': (BuildContext ctx) => Browse(),

'/product': (BuildContext ctx) => ViewProduct(),

'/checkout: (BuildContext ctx) => Checkout(), },

);

}

Yönlendirme ile artık ev özelliğini kullanmadığımızı unutmayın. Yerine intialRoute özelliğini kullanın.

**Bahşiş** initialRoute'unuz "/" ise, bunu tamamen atlayabilir ve Flutter yapabilirsiniz "/" olduğunu varsayar.

## İleri ve geri gitme

Kullanıcıyı bir sahneye manuel olarak yönlendirmek için Navigator'ı kullanacaksınız. pushNamed(bağlam, rota) ve Navigator.pop(bağlam).

Bir kullanıcıyı başka bir rotaya itmek için:

Yükseltilmiş Düğme( child: const Text('Ödeme'), onPressed: () => Navigator.pushNamed(bağlam, '/checkout),

),

Bitirdiklerinde ve geri dönmek istediklerinde:

Yükseltilmiş Düğme( child: const Text('Geri dön'), onPressed: () => Navigator.pop(bağlam),

),

Ama bekleyin, dahası var! Bir İskeleniz varsa, uygulama çubuğuna otomatik olarak bir geri ok eklendiğine dikkat edin (Şekil 7-1). Dokunulduğunda geri dönmek için çalışır. Kullanıcınız Android'deyse, her yerde bulunan Android geri düğmesi de çalışır (Şekil 7-2).



### **Şekil 7-1.** Uygulama çubuğundaki geri oku



### **Şekil 7-2.** Android geri düğmesi yığınlarla çalışır

**İpucu:** MaterialApp'inizde önceden tanımlanmış bir yönlendirme tablosu kullanmayan başka bir yönlendirme çeşidi daha vardır . Bunun yerine, rotayı anında oluşturursunuz :

Navigator.push<void>(bağlam, MaterialPageRoute

<void>(builder: (BuildContext bağlamı) =>

SecondRoute());

Gördüğünüz gibi, biraz daha karmaşık. Ancak özel geçişler istiyorsanız veya yalnızca önceden tanımlanmış rotalar istemiyorsanız popülerdir.

## Bir sahne kapatıldıktan sonra sonuç alma

Yığın gezintisi ile her pop() çağırana geri döner. Bu nedenle, her sahneden bir değer döndürmek mümkündür. Bu son derece yaygın değildir, ancak kullanıcıyı bir

iş akışı. Diyelim ki uygulamanızın bir kullanıcı nesnesini koruyan bir bölümü var. Kullanıcı nesnesi MyUserWidget'ta tanımlanmıştır ve oturum açma kimlik bilgilerini değiştirmek için bir düğme, telefon numarasını değiştirmek için başka bir düğme ve Twitter tanıtıcısını değiştirmek için başka bir düğme sağlarız. Kullanıcı her bir düğmeye dokunduğunda, onları verileri değiştirecekleri bir rotaya itebiliriz(). Öyleyse, değiştirilen bu verileri MyUserWidget'a geri döndürmemiz gerekir. Bir değişkenin döndürülen değeri almasını sağlamak için push() işlevini biraz farklı yapardık:

Burada 'asenkron'a ihtiyaç var çünkü aşağıda 'bekliyoruz'. üzerine basıldı: () zaman uyumsuz {

\_user.twitterKullanıcı Adı = await Navigator.pushNamed(bağlam, '/twitter'); },

**Not** bu *beklemek* anahtar kelimesi, pushNamed() öğesinin bir Gelecek döndürdüğünü belirtir. Ayrıca, bu rotadan döndürülen herhangi bir değerin \_user.twitter tanıtıcısı.

Peki bu değer nasıl geri dönüyor? Tabii ki pop() içinde!

Navigator.pop<String>(bağlam, twitterHandle);

Navigator.pop() öğesi aşırı yüklenmiş. İkinci bir parametre eklerseniz, ilk etapta push() öğesini çağıran widget'a döndürülür. Yukarıdaki örnekte, twitterHandle döndürülür.

Sığ uygulamalar push() ve pop() ile harika çalışır. Ancak uygulamanız, çok sayıda seçenek içeren derin bir gezinme ağacına sahip olabilir. Bunun gibi uygulamalar genellikle en iyi şekilde push() ve pop() için çok sayıda düğmeye sahip olarak sunulmaz. Bunun yerine, bir gezinme menüsüne sahip olmalıdırlar.

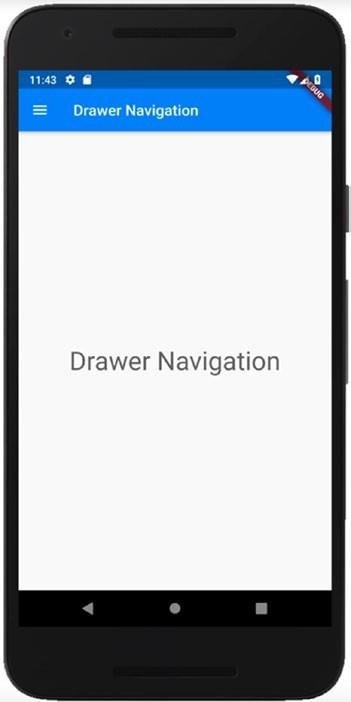
Flutter bize iki tür sağlar. Daha basit uygulamaların sekmeleri olabilir. Daha karmaşık uygulamaların çekmeceleri olacaktır. Şimdi çekmecelere bakalım.

# Çekmece gezintisi

Çekmeceler, çok sayıda gezinme seçeneğimiz olduğunda harikadır - bir sekmeye sığmayacak kadar çok seçenek. Birçok duyarlı web sitesinde, sayfanın üst kısmında, sitedeki diğer sayfalara bağlantılar içeren bir menü görürsünüz. Daha sonra site küçük bir cihazda veya hatta dar bir tarayıcıda görüntülendiğinde, bu menünün yerini, tıklandığında aynı seçeneklerle dolu bir menüyü bırakacak bir hamburger menüsü alır. Temel olarak, bu, sınırlı ekran alanına yanıt veren ve kullanıcı talep edene kadar gizlenen menü seçenekleri sunan sitedir.

Çoğu telefon zaten sınırlı ekran alanına sahip olduğundan, menü seçeneklerinizi, kullanıcı onları görmeye hazır olana kadar bu değerli ekran alanını silip süpürmeyen bir çekmeceye koymayı tercih edebilirsiniz (Şekil 7-3). Ne zaman

o hazır, artık tanıdık hamburger menüsüne (üç satırlı simge) basacaklar ve seçenekler soldan dışarı kayacak (Şekil 7-4). Kullanıcı birini seçtiğinde, onları yeni bir rotaya Navigator.push() edeceğiz.



***Şekil 7-3.*** *Çekmecenin kapalı olduğu bir sahne*



***Şekil 7-4.*** *Çekmecenin açık olduğu bir sahne*

## Çekmece widget'ı

Dışarı kaydırma, içeri kaydırma ve menü seçenekleri içerme özelliğine sahip yerleşik bir Flutter widget'ı olan bir Çekmece widget'ına ihtiyacınız olacak. Bir çekmece kullandığınızda, onu her zaman bir İskele'nin çekmece özelliğine dahil edersiniz, örneğin:

Widget build(BuildContext context) { return Scaffold( appBar: AppBar(

title: const Text('Çekmece Navigasyonu'),

), body: const Text('DrawerNavigation'), drawer: Drawer(child: ListView(

çocuklar: <Widget>[

Metin('Seçenek 1'),

Metin('Seçenek 2'),

Metin('Seçenek 3'),

],

),),

);

}

İskelenizde bir çekmeceniz olduğunda, hamburger simgesinin olduğuna dikkat edin

Uygulama çubuğunun geri düğmesinin yerini alır. Kendi düğmelerinizi manuel olarak oluşturmadığınız sürece her iki düğmeyi de aynı anda göremezsiniz. Bu nedenle, çekmece navigasyonu ve yığın navigasyonu birlikte çalışabilirken, dikkatli olmazsanız biraz garip olabilir. Birlikte gerçekten iyi çalıştıklarına bir örnek, en üst düzeyde bir Çekmeceye sahip olmak ve ardından bunun altındaki tüm düzeylerde yığın gezinmesini kullanmaktır.

**Bahşiş** Uygulamanızın tamamında tutarlı bir çekmecenin kullanılabilir olmasını ister misiniz? Eğer öyleyse, genellikle her sahneye bir İskele koyarız ve çekmeceyi de içine dahil ederiz. Bu nedenle, Çekmecenizi içine koymak en iyisidir. Kendi widget'ı ve dahil edin:

return Scaffold( appBar: AppBar(

title: const Text('Çekmece Navigasyonu'),

), gövde: const Text('DrawerNavigation'), çekmece: MyDrawer(),

);

## Çekmecenin doldurulması

Çekmeceyi eklemek kolaydır. İşin püf noktası, girişleri çekmeceye almak ve ardından başka bir widget'a gitmelerini sağlamaktır. Drawer'ın tek bir widget'ı kabul eden bir alt özelliği olduğunu unutmayın. Çekmecenize birden fazla alt öğe eklemek için , Column (kaydırmaz) veya ListView (kaydırma) gibi bunları destekleyen bir widget kullanacaksınız.

Hangisini seçerseniz seçin, dokunulabilir bir şey koymak isteyeceksiniz çünkü gezinmek için Navigator.push() veya Navigator'ı çağıracaksınız.

pushNamed() komutunu kullanarak yığın gezintisinde yaptığınız gibi hareket edin.

**Bahşiş** Çekmecenin üst kısmında geniş bir alanı kaplayacak şekilde oluşturulmuş DrawerHeader adlı harika bir widget var. Kullanıcıya hangi uygulamada olduklarını hatırlatmak için logonuzu veya diğer marka bilgilerinizi koymak harikadır. Sadece kozmetik ama gerçekten harika görünüyor.

return Drawer(

child: ListView( children: <Widget>[

DrawerHeader( child: Stack(

children: <Widget>[

Image.asset(

'lib/assets/images/BrandLogo.jpg',

),

Konteyner( alignment: Alignment.bottomRight, child: Text( 'Benim Markam', style: Theme.of(context).textTheme.display1,

)),

],

),

),

ListTile( satır başı: const

Simge(Icons.looks\_one), başlık: const Text('Widget 1'), onTap: () {

Navigator.pushNamed(bağlam, '/widget1'); },

),

ListTile( satır başı: const

Simge(Icons.looks\_two), başlık: const Text('Widget 2'), onTap: () {

Navigator.pushNamed(bağlam, '/widget2'); },

),

ListTile( satır aralığı: const

Simge(Icons.looks\_3), başlık: const

Text('Widget 3'), onTap: () {

Navigator.pushNamed(bağlam, '/widget3'); },

),

],

),

);

Çekmece navigasyonu harika, ancak UX uzmanlarının bununla ilgili birkaç sorunu var. Uygulamaların kullanılabilirliğini önemli ölçüde azalttığını, uygulamanızı daha az keşfedilebilir ve daha zor hale getirdiğini iddia ediyorlar1. Sorunun şu olduğunu söylüyorlar

[1http://bit.ly/HamburgerNa](http://bit.ly/HamburgerNav) [YÖNLEND](http://bit.ly/HamburgerNav)İ[RME](http://bit.ly/HamburgerNav)

Seçenekler, kullanıcı tarafından istenene kadar gizlenir. İtirazları, her zaman görünür olan bir uygunlukla çözülebilir. Hangisinden bahsetmişken ...

# Sekme Gezintisi

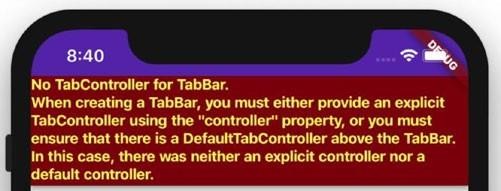
Tahmin edebileceğiniz gibi, bir sekme sistemi N sekmeyi N widget'la eşleştirir. Kullanıcı sekme 1'e bastığında, widget 1'i görür ve bu şekilde devam eder (Şekil 7-5). Eşleştirme bir TabBar widget'ı, bir TabBarView widget'ı ve bir TabBarController ile yapılır.



***Şekil 7-5.*** *Üstte ve altta bir sekme çubuğu*YÖNLENDİRME

## TabController (Denetleyici)

TabController en az belirgin olan kısımdır. Sadece bir tanesine sahip olmanız gerektiğini bilin veya Şekil 7-6'daki hatayı alırsınız.



### **Şekil 7-6.** Bir TabController'ı unuttuğunuzda

Bir tane oluşturmanın en kolay yolu, her şeyi bir length özelliğiyle bir DefaultTabController() içine sarmaktır. Sorun çözüldü. Bu kısım oldukça basit – o kadar basit ki, Flutter'ın neden sizin için dolaylı olarak bir tane oluşturmadığını merak edebilirsiniz. Eğer böyle düşünüyor olsaydınız, yanılmış olmazdınız:

Widget build(BuildContext context) { return DefaultTabController(

uzunluk: 3, çocuk: İskele ( ...

);

}

## TabBarView

Ardından, bir TabBarView widget'ı eklemek isteyeceksiniz. Bu, kullanıcı bir sekmeye bastığında sonunda gösterilecek olan widget'ları tutar ve nerede olduklarını tanımlar.

YÖNLEND RME

gösterilecektir. Genellikle bu, ekranın geri kalanının tamamıdır, ancak widget'ları TabBarView'ın üstüne veya altına veya gerçekten etrafındaki herhangi bir yere koyma fırsatınız vardır:

child: Scaffold(

body: TabBarView(

çocuklar: <Widget>[

WidgetA(),

WidgetB(),

WidgetC(),

],

),

## TabBar ve Sekmeler

Son olarak, sekmeleri kendilerinin tanımlıyoruz. Sekmeler metin veya simge ya da her ikisini birden içerebilir. Burada, her biri hem simge hem de metin içeren üç sekmeli bir Sekme Çubuğu verilmiştir:

çocuk: İskele( appBar: AppBar(

title: const Text('Sekmede

Gezinme'), alt: TabBar( sekmeler: const <Widget>[

Tab(simge: Simge(Icons.looks\_one), çocuk:Metin('A'yı Göster')), Tab(simge: Simge(Icons.looks\_two), çocuk:Metin('B'yi Göster')), Tab(simge:

Simge(Icons.looks\_3), alt: Metin('C'yi Göster')), ]), ...

**Dikkat** Her sekme arasında bire bir yazışma vardır ve her TabBarView alt öğesi; Konumsal olarak eşleştirilirler. TabBarView içindeki widget'larla aynı sayıda sekmeye sahip olmanız gerekir.

## Alt kısımdaki TabBar

Daha önce TabBar'ı, elbette ekranın üst kısmında görünen appBar'a koymayı seçtiğimizi unutmayın. Ancak bazen tasarımınız sekmelerin ekranın alt kısmında görünmesini gerektirir. Bu kolaydır çünkü İskele bottomNavigationBar adlı bir özelliğe sahiptir ve bir TabBar'ı tutacak şekilde oluşturulmuştur:

çocuk: İskele ( ... bottomNavigationBar: Malzeme(

color: Theme.of(context).colorScheme.primary, alt: TabBar(tabs: const <Widget>[

Tab(simge: Simge(Icons.looks\_one), alt: Metin('A'yı Göster')), Tab(simge: Simge(Icons.looks\_two), alt: Metin('B'yi Göster')), Tab(simge: Simge(Icons.looks\_3), alt: Metin('C'yi Göster')),

]),

),

),

**Not** TabBar, bir ekranda açık renkli metnin normal görünümüne sahiptir. koyu arka plan. Böylece, TabBar'ı bir ışığın üzerine yerleştirdiğinizde arka planda, metni görmek zor olabilir (ışık üzerine ışık). Düzeltmek için bu, TabBar'ınızı daha önce yaptığımız gibi daha koyu bir arka plan rengine sahip bir Malzeme widget'ına sarın .

# İletişim Kutusu widget'ı

Son gezinme kategorimiz tartışmasız bir gezinme kategorisi değildir - iletişim kutuları. Bir anlamda, başka bir widget gösteriyorsunuz, bu yüzden ... Gezinti?

Ama başka bir anlamda, temelde bir popYÖNLEND RME -up gösteriyorsunuz, bu yüzden ...

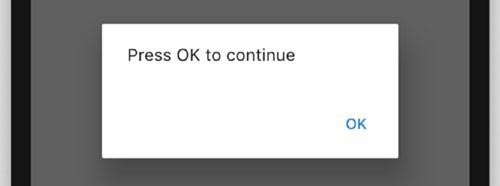
navigasyon değil. ̄\\_(ツ)\_/ ̄

Her iki durumda da, diyaloglar yaygın bir şeydir ve onları ele almalıyız. Kitabın başka hiçbir yerine tam olarak uymadıkları için, bir an için bir navigasyon konusuymuş gibi yapalım. Hey, benimle burada çalış.

## showDialog( ) ve AlertDialog

showDialog() yerleşik bir Flutter yöntemidir. Bir Widget döndüren bir bağlam ve oluşturucu yöntemi sağlamanız gerekir, genellikle ya SimpleDialog

Uyarı İletişim Kutusu. AlertDialog'un bir *actions* parametresi vardır: kullanıcının iletişim kutusunu kapatmasına izin veren (genellikle) FlatButtons Listesi (Şekil 7-7).



### **Şekil 7-7.** Basit bir AlertDialog

Yükseltilmiş Düğme( child: const Text('Ben bir düğmeyim. Press me'), onPressed: () => showDialog<void>( Bağlam: Bağlam, builder: (BuildContext bağlamı) {

return AlertDialog(

content: const Text('Devam etmek için Tamam'a basın'), eylemler: <Widget>[

Düz Düğme( child: const Text('Tamam'),

onPressed: () => Navigator.pop(bağlam)),

],

); YÖNLENDİRME

},

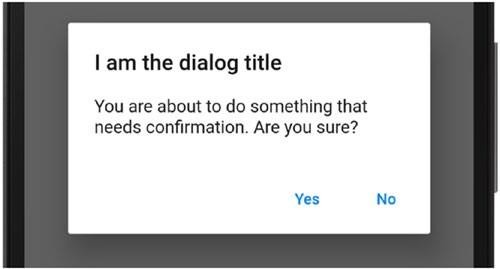
),

),

Bu olması gerekenden daha karmaşık görünüyor. Ve bu en basit şeklidir! Kullanıcıya seçenekler sunmak isterseniz daha karmaşık hale gelir.

## İletişim Kutusu ile Yanıtlar

showDialog() bir Future<T döndürür> bu da çağırana bir değer döndürmesini sağlayabileceğiniz anlamına gelir. Kullanıcının evet veya hayır olarak yanıt vermesini istediğinizi varsayalım (Şekil 7-8).



### **Şekil 7-8.** Bir değer döndüren AlertDialog

İletişim kutusunu oluşturabilir ve yanıtı şu şekilde işleyebilirsiniz:

Yükseltilmiş Düğme( child: const Text('Yanıt al'), onPressed: () async {

Oluşturucu, kullanıcının seçimini burada döndürür. Bu bir Future<String> olduğu için, onu 'bekliyoruz' onu bir Dizeye dönüştürün

Dize yanıtı = await showDialog<String>(

YÖNLEND RME Bağlam: Bağlam, builder: (BuildContext bağlamı) {

return AlertDialog(

content: const Text('Emin misiniz?'), eylemler: <Widget>[

Düz Düğme( child: const Text('Evet'), Kapatıldığında "Evet" döndürün. onPressed: () => Navigator.pop(bağlam, 'Evet')),

FlatButton( child: const Text('Hayır'),

Görevden alındığında "Hayır" döndürün.

onPressed: () => Navigator.pop(bağlam, 'Hayır')), ],

);

},

);

Yukarıda 'beklediğimiz' yanıtla bir şeyler yapın. Yazdır (Yanıt);

},

),

**Bahşiş** Adından da anlaşılacağı gibi, SimpleDialog widget'ı, AlertDialog'un daha basit bir sürümüdür. Eylemleri yoktur ve titleTextStyle, contentTextStyle ve benzeri gibi daha az yapıcı parametresi vardır. Esas olarak, kullanıcının komut istemine yanıt vermesine ihtiyacınız yoksa kullanın ama sadece bilgilendirmek için.

# Navigasyon yöntemleri birleştirilebilir

Çekmeceli bir widget'a ve oradan sekmeli bir widget'a toplu olarak gidebilseniz de dikkatli olmalısınız. Yöntemler uyumsuz değil, ama oğlum, karıştırıldıklarında karmaşık hale gelebilirler! Mesela

push() aracılığıyla navigasyonu çekmecesi olan bir widget'a yığarsanız, uygulama çubuğundaki geri düğmesi artık kullanılamaz. Android'in altında yumuşak bir geri düğmesi var, ancak iOS'ta yok. Böylece kullanıcı artık geri dönmenin bir yolu olmadan sıkışıp kaldı.

Başka bir örnek, bir TabBarView'ın widget'ları vardır, ancak bunlar tabiri caizse barındırılır, bu nedenle İskele'ye sahip olmamaları gerekir. Diğer iki yöntemden birini kullanarak aynı widget'a gitmeye çalıştıysanız, bunu yapmanın hiçbir yolu yoktur

Tekrar gel... Gösterilecek çekmece ve dokunulacak geri düğmesi yok. Yine, kullanıcı sıkışıp kaldı.

Sadece iki farklı türe bağlı kalmanızı ve seviyeleri tutarlı tutmanızı öneririz. Örneğin, kullanıcı için sekmeli bir gezinti deneyimine sahip olmak oldukça yaygındır ve her sekmede yığın gezintisi ile çalışacaksınız. Ancak bundan çok daha karmaşık hale gelin ve ellerinizi doldurabilirsiniz.