

Görsel Programlama

Stateful widget, Custom widget oluşturma, ListView

Emir ÖZTÜRK

State

- Uygulama ekranının çizdirilmesi
- Her çizdirme işleminde instance oluşturma maliyeti
- Sınırlı kaynak
- Sınırlı pil
- Yalnızca ihtiyaç duyulan kısımların güncellenmesi

Flutter uygulamaları çalıştırıldığında uygulama ekranının çizdirilmesi işlemi gerçekleştirilir.

Uygulamanın tüm bileşenlerinin yeniden çizdirilmesi özellikle işlemci gücünün ve pilin sınırlı olduğu platformlarda maliyet olarak kabul edilmektedir. Bu sebeple uygulamanın belirli bir kısmı güncellendiğinde yalnızca o kısmın yeniden çizdirilmesi sağlanabilmektedir.

Stateless - Stateful

- Stateless
 - build()
 - Değişmez
 - Güncelleme desteği yok

```
class Sabit extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Text("Stateless olan");  
  }  
}
```

Uygulama çalıştırıldığında build metodunun çağırılması ile bileşenler çizdirilir. Çizdirme işlemi tamamlandığında herhangi bir değişiklikte bu çizdirme işlemi tekrarlanmaz ve bu sebeple güncellenmesi gereken bölümler varsa bunlar güncellenemez.

Stateless - Stateful

- Stateless
 - Veri alındıktan sonra ekranda değişmeyecekse
 - Ekranda sabit olan alanlar

```
class Sabit extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Text("Stateless olan");  
  }  
}
```

Stateless widget'lar ekranda değişmeyecek ve kullanıcı ile iletişime geçmeyecek kısımlar için kullanılmaktadır. Tasarlanan ekranda sabit olan alanların çizdirilmesi için kullanılması uygundur.

Stateless - Stateful

- Stateful
 - Build
 - Rebuild
 - UI değişikliği gerekli olduğunda

```
class Sabit extends StatefulWidget {  
  @override  
  State<Sabit> createState() => _SabitState();  
}  
  
class _SabitState extends State<Sabit> {  
  @override  
  Widget build(BuildContext context) {  
    return Text("Stateless olan");  
  }  
}
```

Stateful widget'ların state oluşturma metodu ve bu metodun çağırılması ile oluşan state sınıfı bulunmaktadır.

State sınıfının da build metodu bulunur ve statefulwidget'ın her durum oluşturma isteğinde Build metodu yeniden çağırılır.

Ekranda belirli alanların güncellenmesi istendiğinde stateful widget'lar kullanılmalıdır.

Stateless - Stateful

- Stateful
 - Girdi sonucunda UI güncellenecekse
 - Ekran çevrildiyse
 - Ekrandaki bir alanın gelen akışa göre değiştirilmesi gerekiyorsa

```
class Sabit extends StatefulWidget {  
  @override  
  State<Sabit> createState() => _SabitState();  
}  
  
class _SabitState extends State<Sabit> {  
  @override  
  Widget build(BuildContext context) {  
    return Text("Stateless olan");  
  }  
}
```

Ekrandan bir değer alındığında, ekranın şekli (oranı) değiştiğinde veya işlenen bir değer yine ekranda güncellenmesi gerektiğinde stateful widget'ların kullanılması gereklidir.

State

- İlk çizdirme
 - Build
- State güncelleme
 - setState

```
class Sabit extends StatefulWidget {  
  @override  
  State<Sabit> createState() => _SabitState();  
}  
  
class _SabitState extends State<Sabit> {  
  int deger = 0;  
  void degisim() {  
    setState() {  
      deger++;  
    }  
  };  
}  
  
@override  
Widget build(BuildContext context) {  
  return Text(deger.toString());  
}
```

State sınıfı stateless widget'larda olduğu gibi bir Build metodu içerir. Çizdirme durumunda yine stateless widget'lardaki gibi Build metodu çağırılır ve ekrana çizdirme gerçekleşir. Stateless widget'lardan farklı olarak değer değiştirileceği durumda setState metodu tetiklenir ve bu metodun içerisine değişecek kısımlar tanımlanır. setState tetiklendiğinde içerisindeki işlemleri bitirdikten sonra Build metodunu çağırarak yeniden çizdirme işlemini gerçekleştirecektir.

Custom Widget Oluşturma

- Widget ağacının kalabalıklaşması
- Kod okunurluğu
- Mantıksal elementlerin alt elementlere bölünmesi
- Birden fazla kez kullanılacak widget grupları için kod tekrarının engellenmesi

Belirli sayıda bileşenin ekrana eklenmesi durumunda widget ağacı kalabalıklaşacak ve frontend için yazılan kodun takibi zorlaşacaktır. Birden fazla alt dala sahip her bileşen karmaşıklığı arttıracak ve kod okunurluğunu azaltacaktır. Bunun çözümü olarak mantıksal olarak ekranda bir arada gruplanmış widget'ları alt widget'lar şeklinde gruplamaktır. Ayrıca bir widget birden fazla kullanılacaksa (Örneğin bir liste elemanı gibi) yine widget gruplama işleminin kullanılması uygun olacaktır.

Custom Widget Oluşturma

- Yeni bir dart dosyası
- component_adi_widget ya da component_adi_item ismi
 - Zorunluluk değil
- Widget'in feedback vermesi gerekiyorsa
 - Gesture detection

Bir grup widget'ın birleştirilmesi için bu widget'lar yeni bir Dart dosyasına atılabilir. Dosyanın adı genellikle snake case adı verilen _ karakterleri ile birleştirilmiş kelimeler ile verilir. Oluşturulan widget'lar ekrandan alınan bir tıklama veya dokunmatik ekranlarda desteklenen dokunma hareketlerini tespit edecekse gesture detection kullanılabilir.

GestureDetector

- GestureDetector()
- Child:
- Gesture'lar
 - onTap
 - onDoubleTap
 - onLongPress
 - onHorizontalDragUpdate, onVerticalDragUpdate, onPanUpdate, onScaleUpdate

GestureDetector widget'ı bir widget'ın en dışında eklenebilir ve Child özelliği ile bir widget alır. Bu sayede gesturedetector içerisine eklenen widget'lar belirli hareketleri yakalayabilme kabiliyetine sahip olur. Örneğin bir kere, iki kere veya uzun basma / tıklama - yukarı aşağı kaydırma veya yeniden boyutlandırma hareketleri yakalanabilmektedir. Bu hareketler yakalandığında yapılması istenen işlemler metot olarak verilebilir.

Listview

- Elemanların listelenmesi
- Masaüstü geliştirme ortamlarında listbox
- Listelenen elemanların widget olarak tanımlanma gerekliliği

Veri ile ilgili uygulamalarda listeleme önem taşımaktadır. Belirli kişilerin listesi, yapılacaklar listesi, toplanan sensör verilerinin zamana göre bir listesi gibi listeler hazırlanıp ekranda gösterilmektedir. Bu gibi liste yapılarının gösterilmesi için masaüstü uygulama geliştirme platformlarının bir çoğu listbox ya da benzer bir isimde liste kutusuna sahiptir. Flutter'da ise belirli formatta tanımlanmış bir widget'ın bir taşıyıcı içerisinde sıralı bir şekilde eklenmesi ile listeleme gerçekleştirilir.

Listview Oluşturma

- Children
 - Elemanlar eklenir, scroll etme şansı bulunmaz
- Builder
 - itemCount
 - itemBuilder:(context, index){}

Listview içerisinde liste elemanları children veya Builder ile verilebilmektedir.

Listview Oluřturma

- Children ile elemanlar eklenir
- Scroll etme problemi
- Eleman sayısının çok olmasında layout problemleri
- Tüm elemanların yüklenmesinde performans problemi

Listview bileřeni children özellięi alır. Eęer sabit ve kısa listeler bulunuyorsa kullanılması uygundur. Fakat liste ekranın boyutunu aşacaksa ve bu listenin üzerinde kaydırma işlemi yapılması gerekiyorsa children bu işlemleri gerçekleřtirmeyecektir. Ayrıca eleman sayısı çok olduęunda tümünü çizdirmeyi amaçlayacağı için performans problemleri yaşamak da olasıdır.

Listview Oluřturma

- Builder ile eleman eklenir
 - itemBuilder:(context, index){}
- itemCount ile gösterilecek eleman sayısı belirlenir
- Scroll etme mevcut
- Yalnızca gösterilen elemanlar yüklenir

Bunun yerine listview bileřeninin builder özellięinin kullanılması gerekmektedir. Builder kaç adet liste elemanı olduęunu alır ve bu liste elemanlarının hangisinin etkileřime girdięini tespit etmek için ise index kullanır. Builder metodu ile yalnızca ekranda gösterilen elemanlar çizdirilir. Dięer elemanlar ise istek durumunda (on demand) yenilenir ve çizdirilir.

Listview Elemanları belirleme

- Listview widget'ının bulunduęu sınıfta elemanların bulunduęu liste
- Builder içerisindeki index kullanılarak liste[index]
- List<String> liste;
- ListView.builder((ctx,i){return Text(liste[i]);});

Bir listview içerisinde özel oluşturulmuş widget'lar ile elemanların listelenmesi gerekmektedir. Fakat bu widget'ların verilerinin tek tek elle girilmesi olası deęildir. Bu sebeple elde olan ham verinin widget'a atanması (yapıcı ya da parametre yolu ile) ve hangi liste elemanının atanacağıının belirlenmesi için listview.builder'ın index özellięinin kullanılması gerekmektedir.