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 değerin yine ekranda güncellenmesi gerektiğinde stateful widget'ların kullanılması gereklidir.

State

- İlk çizdirme
  - Build
- State güncelleme
  - setState

```dart
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'ın 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.

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

Listview bileşeni children özelliği alır. Eğer sabit ve kısa listeler bulunuyorsa kullanılması uygundur. Fakat liste ekranın boyutunu aşacaksa ve bu listenin üzerinde kaydırma işlemi yapılması gerekiyorsa children bu işlemleri gerçekleştiremeyecektir. Ayrıca eleman sayısı çok olduğunda tümünü çizdirmeyi amaçlayacağı için performans problemleri yaşamak da olasıdır.

Listview
Oluşturma

- Builder
- Çizdirme ve bellek planı
- Verilmesi gereken özellikler
  - itemCount
    - Eleman sayısı verilmeli
  - itemBuilder:(context, index){}
    - Build içerisindeki context verilebilir.
- Yalnızca gösterilen elemanlar yüklenir

Listview.builder metodu ile liste elemanları çizdirilebilmektedir. Bu elemanlar çizdirilirken iki farklı özelliğin tanımlanması gerekmektedir. Bunların ilki itemCount olan eleman sayısıdır. Genellikle listede gösterilecek liste veya dizi uzunluğu burada verilir. itemBuilder ise her eleman çizdirilirken yapılması gerekenlerin tanımlandığı alandır. Burada index değeri kullanılan dizi/liste'nin index'inci elemanını ifade eder.

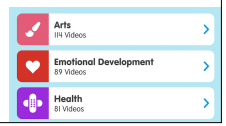
Listview
Elemanları belirleme

- Listview widget'ının bulunduğu sınıfta elemanların bulunduğu 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 değildir. Bu sebeple elde olan ham verinin widget'a atanması (yapıcı ya da parametre yolu ile) ve hangi liste elemanının atanacağının belirlenmesi için listview.builder'ın index özelliğinin kullanılması gerekmektedir.

Listview

- Liste elemanı olarak ListTile kullanılabilir
- ListTile belirli liste özellikleri sunar
  - leading
  - trailing
  - title
  - subtitle



Bir liste elemanı istenilen bir component olarak çizdirilebileceği gibi hazır ListTile sınıfı da kullanılabilir. Bu sınıf bir liste satırını çizdirir ve belirli öntanımlı özellikleri sunar.
