Görsel Programlama

Olay tabanlı programlama, Widget, Temel Widget'lar

Emir ÖZTÜRK

Olay Tabanlı Programlama

- Prosedürel programlama
  - Adım adım çalıştırma
- Olay tabanlı programlama
  - Tetikleme
  - Olayı gerçekleştiren bileşenler

Prosedürel programlama yapılan dillerde yazılan kodlar başlangıç satırından itibaren sıra ile çağırılır. Her satır işlemeyi bitirdikten sonra bir alt satırdaki komuta geçilir. Fonksiyon ya da prosedür çağırma durumunda iste fonksiyona yönlenilir ve fonksiyon içerisinde de satır sıralı çalışma mantığı devam eder. Olay tabanlı programlamada ise ana thread çizdirme ve ekrandan girdi işlemlerinin beklenmesi gibi işleri üstlenirken yazılan kod satırları belirli olayların tetiklenmesi durumunda çalıştırılır. Örneğin ekrandaki bir tuşa basıldığında yapılacak işlemler belirli bir kod bloğunun içerisinde tanımlandıktan sonra bu tuşa basılmadan bu kod tetiklenmez. Tuşa basıldığı an ana thread bu tuşa basıldığı bilgisini elde eder ve tuşa basıldığında çalıştırılacak kod satırlarınının yazıldığı bloğu çalıştırmaya başlar

Flutter GUI
UI as Code

- Dart dili
- Ön yüzün programlama dili ile yazılması
- Ön yüzün arkaplan kodu ile aynı dilde yazılması
  - Wpf - C# - xaml
  - React native - javascript - jsx
- En temel parçalar
  - Widget

GUI kullanılan tüm dillerde UI kısmının hazırlanması için (JavaFX için xml, Android geliştirme için xml, C# ile Windows üzerinde yapılan wpf ile geliştirme için xaml gibi) belirlenmiş bir dil bulunur. Flutter UI tarafında da Dart dili ile yazılmaktadır. Flutter'ın diğer dillere göre avantajı hem ui hem de arkaplan kodu (logic) için aynı dili kullanmasıdır.
UI geliştirme tarafında hiyerarşik bir yapının kurulması için flutter'da tüm bileşenler Widget olarak adlandırılmıştır.

Widget

- Tüm bileşenler widget
- 08.2022 itibariyle 172 adet
- 4 ana kategori
  - Değer widget'ları
  - Düzen widget'ları
  - Navigasyon widget'ları
  - Diğer widget'lar
- https://docs.flutter.dev/reference/widgets

Tüm bileşenler widget olarak tanımlanır (extend edilerek).
Widget'lar 4 ana kategoride incelenebilir.

Değer Widget'ları
Value Widgets

- Değer saklamak
  - Ekrandan
  - Belirli bir veri kaynağından
- Değer göstermek
- Örneğin
  - Text
  - TextField
  - Image
  - Checkbox
  - Icon

Değer widget'ları ekrandan veri alınan, veri gösterilen veya kullanıcı ile iletişimi sağlayan widget'lardır.

Düzen Widget'ları
Layout Widgets

- Ekrandaki widget'ların konum ve boyutlarını belirlemek
- Responsive tasarımda nasıl davranış göstereceğini belirlemek
- Örneğin
  - Container
  - Column
  - Row
  - Scaffold
  - SizedBox
  - Align
  - Card

Düzen widget'ları ekranda değer widget'ları gibi parçaların nasıl nerede veya nasıl bir boyutta gözükeceğine dair tanımların yapıldığı widget'lardır. Özellikle birden fazla boyutta ekrana geliştirilecek bir uygulamanın responsive olması amaçlanmaktadır.

Navigasyon Widget'ları
Navigation Widgets

- Birden fazla sayfa
- Yönlenme
- Örneğin
  - MaterialApp
  - TabBar
  - Navigator
  - BottomNavigationBar

Navigasyon widget'ları sayfalar arası geçiş ve yönlendirme için kullanılır.

Diğer Widget'lar
Other Widgets

- Geriye kalan tüm işler için
- Örneğin
  - GestureDetector
  - Cupertino
  - Transitions
  - Theme

Diğer 3 widget tanımına uymayan (örneğin animasyonların gerçekleştirildiği) widget'lar da bu kategoride tanımlanır.

Temel Widget'lar
Text

- Text("String ifade")
- Style ile görünüm değişimi
  - Font
  - Font-weight
  - Color
  - Size

Temelde kullanılabilecek widget'ların bir örneği Text widget'ıdır. İlk parametre olarak göstereceği string bir ifade verilebilir.
Eğer text'in görüntüsünün değiştirilmesi isteniyorsa style: parametresi ile bu değişiklik gerçekleştirilebilir. Style Text için TextStyle() sınıfından bir nesne alır. TextStyle() sınıfı ise parametre olarak font türü, font ağırlığı, renk, boyut gibi değişkenleri içermektedir.

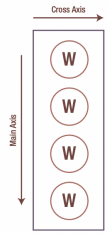
Temel Widget'lar
Row

- children
- mainAxisAlignment
- crossAxisAlignment

Container yalnızca bir widget alabileceği için birden fazla widget eklenmek isteniyorsa bunlar satır veya sütun şeklinde tanımlanabilir. Bir satırda yatay olarak birden fazla widget'ın eklenmesi için Row widget'ı kullanılabilir. Row ile children parametresine bir widget dizisi verilmektedir. Ayrıca Row içerisindeki bileşenlerin sıralanışı main ve cross axis alignment ile belirlenir.

Temel Widget'lar
Column

- children
- mainAxisAlignment
- crossAxisAlignment



Column row ile aynı özelliklere sahip ve dikey olarak bileşenleri saklayan bir widget'tır. Row için main axis yatayken column için dikeydir. CrossAxis de aynı şekilde row'un tersidir.

Temel Widget'lar
Button

- ~~RaisedButton~~ ElevatedButton
- ~~FlatButton~~ TextButton
- IconButton
- child
- onPressed

Kullanıcı ile iletişime geçmenin bir yolu da basılacak tuşlardır. Flutter için 3 farklı tuş biçimi bulunmaktadır. Bunlardan ikisi yeni sürümde yeniden adlandırılmıştır. Elevatedbutton standart olarak 3d görünümlü bir tuş üretir. TextButton bir link gibi davranan tuş tanımlamak için kullanılır. IconButton ise seçilen bir simgenin gösterildiği bir tuş şeklidir.
Her tuşun içerisine eklenebilecek bir widget alan Child ve tıklandığında ne yapılacağına dair bir fonksiyon alan onPressed parametresi bulunur.

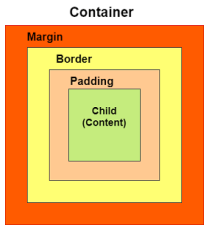
Temel Widget'lar
Icon

- Simge
- Icon(enum,renk,boyut)
- Icon(Icons.icon_adi,color: Colors.renk_adi,size: double)

IconButton'a vermek üzere hazır simgelerden bir tanesi seçilebilir. Bunun için Icon() sınıfı kullanılabilir. Icon sınıfı ilk parametre olarak Icons enum'ı altındaki bir simgeyi, ikinci parametre olarak bu simgenin rengini ve son parametre olarak bu simgenin boyutunu alır.

Temel Widget'lar
Container

- child
- margin
- padding
- EdgeInsets
- decoration
  - BoxDecoration



Widget'ların saklandığı en temel widget container'dır. Container içerisine bir adet widget'ı Child özelliği ile alır. Bu Child widget'ı gösterirken ne kadar iç ve dış boşluk vereceğini margin ve padding özellikleri belirler. Padding ve margin EdgeInsets sınıfı kullanılarak her yön için farklı değerlerde verilebilir. Decoration ile BoxDecoration kullanılarak container dışında bir şekil belirlenebilir ve bu şeklin stili ayarlanabilir.

Stateless ve Stateful Widget'lar

- State kavramı
- Uygulamanın sayfa yenilenmesi
- Yenileme ihtiyacı duymayan parçalar

Flutter performansın arttırılması amacıyla belirli bileşenleri sabit, yalnızca değiştirilmesi gereken bileşenleri ise dinamik yapabilmek adına state kavramını sunar. State içeren Stateful ve sabit olan Stateless widget'ların ayrılması ile uygulamanın yalnızca ihtiyaç duyulan kısımlarının dinamik olarak tanımlanması sağlanır. Bu sayede yeniden çizdirme veya veri güncelleme işlerinde stateful kısımlar yenilenirken stateless kısımlar değişmez.

Componentization

- Uygulamanın sınırlarının belirlenmesi
- Uygulamanın parçalara ayrılması
- Ayrılmış parçaların karmaşıklık kontrolü
- Karmaşık olan parçaların daha alt parçalara bölünmesi

Flutter'da her birim bir widget olduğundan belirli tekrarda kullanılacak kısımlar da bir widget olarak gruplanabilmektedirler. Eğer modüler yapıda tekrarlı kullanılacak bir widget grubu varsa bu grubun yeni bir widget altında toplanması mantıklı olacaktır. Her widget'ı gruplamak ya da tüm widget'ları ayrı ayrı tanımlamak uygulama karmaşıklığını arttıracağından, öncelikle uygulama sınırları ve bileşenleri belirlenmeli, ardından widget'ların gruplanması işlemleri buna göre gerçekleştirilmelidir. Çok karmaşık bölümlerin alt parçalara bölünme işlemi optimum düzeyde durdurulmalıdır.
