**DERS: FLUTTER WIDGET YAPISI VE SCAFFOLD WIDGET**

**Giriş ve Genel Bakış**

**Flutter'ın Temelleri**

**Flutter'ın Mimarisi:** Flutter, Google tarafından geliştirilen, açık kaynaklı bir UI (Kullanıcı Arayüzü) yazılım geliştirme kitidir. Dart programlama dilini kullanır. En önemli özelliği, tek bir kod tabanıyla hem iOS hem de Android platformları için derleme yapabilmesidir (cross-platform).

**Çalışma Prensibi:** Flutter, kendi rendering motoru (Skia) sayesinde platformun yerel widget'larını kullanmak yerine, kendi widget'larını çizerek çalışır. Bu, yüksek performans ve platformlar arası tutarlılık sağlar.

**Avantajları:**

**Hızlı Geliştirme:** Hot Reload özelliği sayesinde değişiklikler anında görülebilir.

**Yüksek Performans:** Kendi rendering motorunu kullanır.

**Zengin Widget Kütüphanesi:** Hazır ve özelleştirilebilir widget'lar sunar.

**Cross-Platform:** Tek kod tabanı ile iOS ve Android'e çıktı alınabilir.

**Açık Kaynak:** Geniş bir topluluk tarafından desteklenir.

**Widget Kavramı:** Flutter'da her şey bir widget'tır. Ekranda gördüğünüz her şey (metinler, butonlar, resimler, düzenler) widget'lardan oluşur. Widget'lar, uygulamanın kullanıcı arayüzünü oluşturmak için kullanılan temel yapı taşlarıdır.

**Widget Ağacı**

**Yapı ve Hiyerarşi:** Flutter uygulamaları, widget'ların iç içe geçmesiyle oluşan bir ağaç yapısına sahiptir. Bu ağaç, uygulamanın görsel hiyerarşisini tanımlar. En üstte genellikle bir MaterialApp (veya CupertinoApp) widget'ı bulunur. Bunun altında Scaffold, Container, Column, Row, Text gibi widget'lar yer alır.

**Etkileşim:** Widget'lar, bu ağaç yapısı sayesinde birbirleriyle etkileşimde bulunur. Örneğin, bir butonun tıklanma olayı (event) üst widget'lara iletilir ve bu olaylar işlenir.

// Widget Ağacı Örneği (Basitleştirilmiş)

MaterialApp

└── Scaffold

├── AppBar

│ └── Text ("Uygulama Başlığı")

└── Body

└── Center

└── Text("Merhaba Dünya!")

**Flutter Widget Yapısı**

**Temel Kavramlar**

**StatelessWidget:** Durumu (state) olmayan, yani içeriği değişmeyen widget'lardır. Örneğin, bir metin (Text), bir ikon (Icon) veya sabit bir resim (Image) StatelessWidget olarak tanımlanabilir. Veri değiştiğinde, widget yeniden oluşturulur.

class MyStatelessWidget extends StatelessWidget {

final String text;

MyStatelessWidget({required this.text});

@override

Widget build(BuildContext context) {

return Text(text);

}

}

**StatefulWidget:** Durumu (state) olan, yani içeriği zamanla değişebilen widget'lardır. Kullanıcı etkileşimi (buton tıklaması, form girişi) veya veri güncellemeleri sonucunda widget'ın durumu değişebilir. StatefulWidget iki sınıftan oluşur:

**StatefulWidget Sınıfı:** Widget'ın kendisini tanımlar.

**State Sınıfı:** Widget'ın durumunu ve build metodunu içerir. Durum değiştiğinde, setState metodu çağrılarak widget'ın yeniden çizilmesi sağlanır.

class MyStatefulWidget extends StatefulWidget {

@override

\_MyStatefulWidgetState createState() => \_MyStatefulWidgetState();

}

class \_MyStatefulWidgetState extends State<MyStatefulWidget> {

int \_counter = 0;

void \_incrementCounter() {

setState(() {

\_counter++;

});

}

@override

Widget build(BuildContext context) {

return Column(

children: [

Text('Sayaç: $\_counter'),

ElevatedButton(

onPressed: \_incrementCounter,

child: Text('Artır'),

),

],

);

}

}

**Kullanım Alanları:**

**StatelessWidget:** Sabit içerikler, statik sayfalar, ikonlar, metinler.

**StatefulWidget:** Kullanıcı etkileşimi olan alanlar (formlar, butonlar), animasyonlar, veri güncellemeleri olan bölümler.

**Widget Yaşam Döngüsü (StatefulWidget için)**

**initState():** State objesi oluşturulduğunda bir kere çağrılır. Başlangıç değerleri, abonelikler (subscriptions) burada tanımlanır.

**build():** Widget'ın arayüzünü oluşturur. State değiştiğinde veya üst widget yeniden oluşturulduğunda tekrar çağrılır. *En önemli metottur.*

**didChangeDependencies():** initState()'den sonra ve InheritedWidget'lara bağımlılık değiştiğinde çağrılır.

**didUpdateWidget():** Widget'ın konfigürasyonu değiştiğinde (örneğin, üst widget tarafından yeni özellikler (props) gönderildiğinde) çağrılır.

**setState():** State objesinin içindeki veriyi günceller ve build() metodunu tekrar çağırarak widget'ın yeniden çizilmesini tetikler.

**deactivate():** Widget ağacından kaldırılmadan önce çağrılır.

**dispose():** Widget ağacından tamamen kaldırıldığında çağrılır. Kaynakları (abonelikler, dinleyiciler) temizlemek için kullanılır.

class MyLifecycleWidget extends StatefulWidget {

@override

\_MyLifecycleWidgetState createState() => \_MyLifecycleWidgetState();

}

class \_MyLifecycleWidgetState extends State<MyLifecycleWidget> {

@override

void initState() {

super.initState();

print('initState çağrıldı');

// Abonelikleri başlat, başlangıç değerlerini ayarla

}

@override

Widget build(BuildContext context) {

print('build çağrıldı');

return Text('Yaşam Döngüsü Örneği');

}

@override

void didChangeDependencies() {

super.didChangeDependencies();

print('didChangeDependencies çağrıldı');

}

@override

void didUpdateWidget(MyLifecycleWidget oldWidget) {

super.didUpdateWidget(oldWidget);

print('didUpdateWidget çağrıldı');

}

@override

void deactivate() {

super.deactivate();

print("deactivate çağrıldı");

}

@override

void dispose() {

print('dispose çağrıldı');

// Abonelikleri iptal et, kaynakları temizle

super.dispose();

}

}

**Örnek Uygulamalar**

**Basit Widget Hiyerarşisi:**

MaterialApp(

home: Scaffold(

appBar: AppBar(title: Text('Widget Hiyerarşisi')),

body: Center(

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: [

Text('Merhaba', style: TextStyle(fontSize: 24)),

Icon(Icons.star, color: Colors.amber),

ElevatedButton(

onPressed: () {},

child: Text('Tıkla'),

),

],

),

),

),

)

**Widget ağacı diyagramı:** Yukarıdaki kodun görsel bir temsilini çizin (ağaç yapısı şeklinde).

**Özelleştirilmiş Widget:**

class MyCustomButton extends StatelessWidget {

final String label;

final VoidCallback onPressed; // Fonksiyon tipinde bir prop

MyCustomButton({required this.label, required this.onPressed});

@override

Widget build(BuildContext context) {

return ElevatedButton(

onPressed: onPressed,

child: Text(label),

style: ElevatedButton.styleFrom(

backgroundColor: Colors.blue,

foregroundColor: Colors.white,

),

);

}

}

// Kullanımı:

MyCustomButton(label: 'Kaydet', onPressed: () { print('Kaydedildi!'); }),

**Scaffold Widget Detayları**

**Scaffold Nedir?**

**Tanım:** Scaffold, Flutter'da Material Design temel görsel düzenini sağlayan bir widget'tır. Uygulama çubuğu (AppBar), gövde (body), alt navigasyon çubuğu (BottomNavigationBar), yan menü (Drawer), yüzen eylem butonu (FloatingActionButton) gibi temel bileşenleri kolayca eklemenizi sağlar.

**Amaç:** Tutarlı ve Material Design prensiplerine uygun bir uygulama iskeleti oluşturmak.

**Önem:** Flutter uygulamalarının çoğunda kullanılır. Uygulama yapısını hızlı ve kolay bir şekilde kurmanızı sağlar.

**Bileşenler ve Özellikler**

**appBar:** Uygulamanın üst kısmında yer alan başlık çubuğudur. Genellikle bir başlık (title), eylem butonları (actions) ve bazen de sekmeler (tabs) içerir.

Scaffold(

appBar: AppBar(

title: Text('Uygulama Başlığı'),

actions: [

IconButton(icon: Icon(Icons.search), onPressed: () {}),

IconButton(icon: Icon(Icons.settings), onPressed: () {}),

],

),

// ...

)

**body:** Uygulamanın ana içeriğinin yer aldığı bölümdür. Buraya istediğiniz widget'ları (Text, Image, ListView, GridView, vb.) yerleştirebilirsiniz.

Scaffold(

// ...

body: Center(

child: Text('Ana İçerik'),

),

)

**floatingActionButton:** Genellikle ekranın sağ alt köşesinde yer alan, yuvarlak bir butondur. Ana eylemleri (örneğin, yeni bir öğe ekleme) tetiklemek için kullanılır.

Scaffold(

// ...

floatingActionButton: FloatingActionButton(

onPressed: () {

// Butona tıklandığında yapılacak işlemler

},

child: Icon(Icons.add),

),

floatingActionButtonLocation: FloatingActionButtonLocation.centerDocked, // Konumlandırma

)

**drawer:** Ekranın solundan (veya sağından) kayarak açılan bir yan menüdür. Uygulama içi gezinme, ayarlar veya diğer seçenekler için kullanılır.

Scaffold(

// ...

drawer: Drawer(

child: ListView(

children: [

DrawerHeader(

child: Text('Menü Başlığı'),

decoration: BoxDecoration(color: Colors.blue),

),

ListTile(title: Text('Öğe 1'), onTap: () {}),

ListTile(title: Text('Öğe 2'), onTap: () {}),

],

),

),

)

**bottomNavigationBar:** Ekranın altında yer alan bir navigasyon çubuğudur. Uygulamanın farklı bölümleri arasında geçiş yapmak için kullanılır.

Scaffold(

// ...

bottomNavigationBar: BottomNavigationBar(

items: [

BottomNavigationBarItem(icon: Icon(Icons.home), label: 'Ana Sayfa'),

BottomNavigationBarItem(icon: Icon(Icons.search), label: 'Arama'),

BottomNavigationBarItem(icon: Icon(Icons.person), label: 'Profil'),

],

currentIndex: \_selectedIndex, // Seçili olan öğenin indeksi

onTap: (index) {

setState(() {

\_selectedIndex = index;

});

},

),

)

**Diğer Özellikler:** backgroundColor, resizeToAvoidBottomInset, primary, extendBody, extendBodyBehindAppBar, bottomSheet gibi birçok özelliği vardır.

**Kod Örnekleri ve Uygulama Senaryoları**

**Basit Not Alma Uygulaması:**

class NoteApp extends StatefulWidget {

@override

\_NoteAppState createState() => \_NoteAppState();

}

class \_NoteAppState extends State<NoteApp> {

List<String> \_notes = [];

void \_addNote() {

setState(() {

\_notes.add('Yeni Not ${\_notes.length + 1}');

});

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: Text('Notlarım')),

body: ListView.builder(

itemCount: \_notes.length,

itemBuilder: (context, index) {

return ListTile(title: Text(\_notes[index]));

},

),

floatingActionButton: FloatingActionButton(

onPressed: \_addNote,

child: Icon(Icons.add),

),

);

}

}

Bu örnek, Scaffold'un appBar, body (ListView ile) ve floatingActionButton özelliklerini kullanarak basit bir not alma uygulamasının temel yapısını gösterir.

**Responsive Tasarım İpuçları**

**MediaQuery:** Ekran boyutlarını ve yönünü (yatay/dikey) almak için MediaQuery kullanın.

**LayoutBuilder:** Widget'ın boyutlarına göre farklı düzenler oluşturmak için LayoutBuilder kullanın.

**Flexible ve Expanded:** Column ve Row içinde widget'ların esnek bir şekilde yerleşmesini sağlamak için Flexible ve Expanded kullanın.

**AspectRatio:** Widget'ların belirli bir en-boy oranını korumasını sağlamak için AspectRatio kullanın.

// Örnek: MediaQuery kullanımı

double screenWidth = MediaQuery.of(context).size.width;

double screenHeight = MediaQuery.of(context).size.height;

if (screenWidth > 600) {

// Tablet düzeni

} else {

// Telefon düzeni

}

**İleri Düzey Konular ve Uygulamalı Çalışmalar**

**En İyi Uygulamalar**

**Kod Okunabilirliği:**

Widget'ları küçük ve işlevsel parçalara ayırın.

Anlamlı isimler kullanın.

Yorum satırları ekleyin.

Kodunuzu düzenli tutun (girintileme, boşluklar).

**Performans:**

Gereksiz setState çağrılarından kaçının. Sadece değişen widget'ları güncelleyin.

const anahtar kelimesini kullanarak sabit widget'ları önbelleğe alın.

ListView.builder gibi optimize edilmiş widget'ları kullanın.

Büyük resimleri optimize edin.

**Yeniden Kullanılabilirlik:**

Ortak widget'ları ayrı dosyalara ayırın ve tekrar kullanın.

Parametreli widget'lar oluşturun.

**Widget Optimizasyonu**

RepaintBoundary: Widget ağacının belirli bölümlerinin gereksiz yere yeniden çizilmesini engeller.

AutomaticKeepAliveClientMixin: ListView gibi scrollable widget'lar içinde, görünmeyen widget'ların durumunu korur.

**Hata Ayıklama Stratejileri:**

Flutter DevTools'u kullanın (widget ağacını inceleme, performans analizi, hata ayıklama).

print() ifadeleriyle loglama yapın.

Debug modunda çalıştırın.

Hata mesajlarını dikkatlice okuyun.

Stack Overflow ve Flutter topluluğundan yardım alın.

**Öğrenci Projeleri İçin Öneriler**

**Basit Bir To-Do Listesi Uygulaması:** Görev ekleme, silme ve tamamlama işlevleri olan bir uygulama.

**Haber Okuma Uygulaması (Basit):** Bir haber kaynağından (API) başlıkları ve özetleri çeken ve listeleyen bir uygulama.

**Basit Bir Hesap Makinesi:** Temel matematiksel işlemleri yapabilen bir hesap makinesi.

**Profil Sayfası:** Kullanıcı bilgilerini (isim, resim, vb.) gösteren ve düzenlemeye izin veren bir sayfa.

**Tartışma Soruları ve Özet Notlar**

Stateless ve Stateful widget arasındaki temel farklar nelerdir? Hangi durumlarda hangisini tercih edersiniz?

Scaffold widget'ının temel bileşenleri nelerdir ve ne işe yararlar?

Widget yaşam döngüsünün hangi adımları vardır ve bu adımlar ne zaman çağrılır?

setState() metodunun görevi nedir? Neden önemlidir?

Performanslı bir Flutter uygulaması geliştirmek için nelere dikkat etmeliyiz?

**Sonuç ve Özet**

Flutter'da her şeyin bir widget olduğunu ve widget'ların ağaç yapısı şeklinde bir araya geldiğini öğrendik.

StatelessWidget ve StatefulWidget arasındaki farkları, kullanım alanlarını ve yaşam döngüsünü inceledik.

Scaffold widget'ının Flutter uygulamalarındaki önemini, temel bileşenlerini ve nasıl kullanılacağını öğrendik.

Responsive tasarım, performans optimizasyonu ve hata ayıklama gibi konulara değindik.

**Ek Notlar & İleri Okuma/Kaynaklar**

**Resmi Flutter Dokümantasyonu:** [https://flutter.dev/](https://www.google.com/url?sa=E&q=https%3A%2F%2Fflutter.dev%2F) (En önemli kaynak!)

**Flutter Widget Kataloğu:** [https://docs.flutter.dev/ui/widgets](https://www.google.com/url?sa=E&q=https%3A%2F%2Fdocs.flutter.dev%2Fui%2Fwidgets)

**"Flutter in Action" Kitabı**

**"Pragmatic Flutter" Kitabı**

**Flutter ile ilgili blog yazıları ve video dersler** (Udemy, YouTube, Medium, vb.)

**Flutter Topluluğu:** [https://flutter.dev/community](https://www.google.com/url?sa=E&q=https%3A%2F%2Fflutter.dev%2Fcommunity) (Stack Overflow, Reddit, Discord)

**Resmi Dart Dökümantasyonu**: [https://dart.dev/guides](https://www.google.com/url?sa=E&q=https%3A%2F%2Fdart.dev%2Fguides)