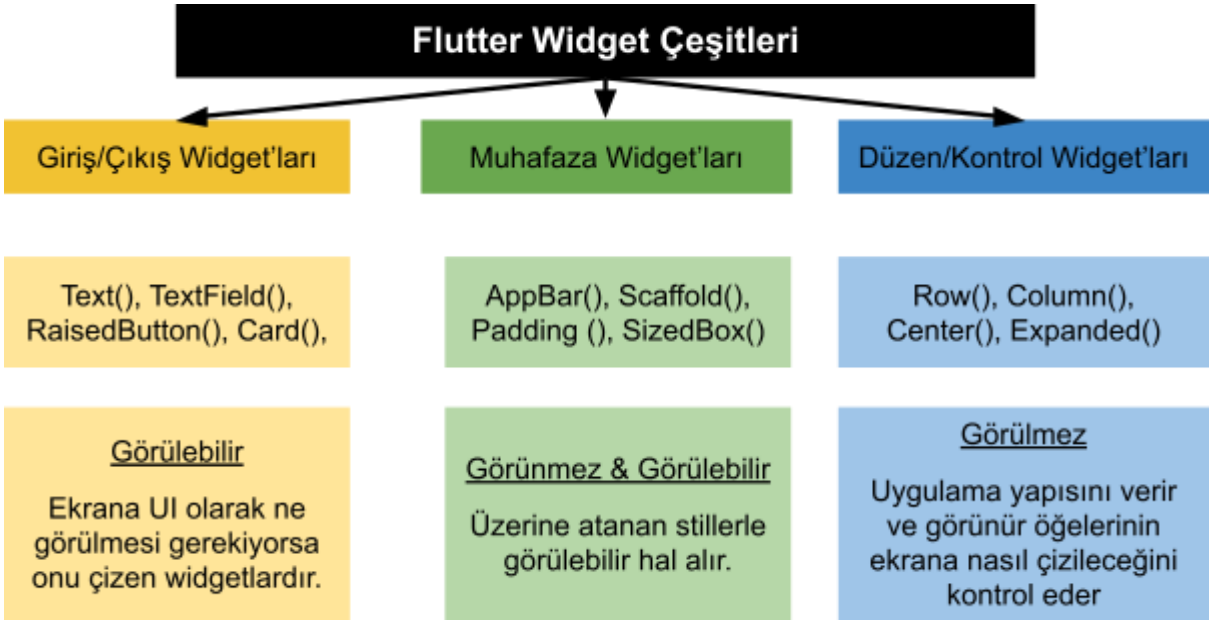
Flutter Widget Türleri ve Uygulamaları



A) Hazırlık:

İlk olarak, Flutter projesini oluşturarak işe başlayalım: Bunun için hem **VS Code'u** hem de **Android Studio Emulator'u** açın. Yeni bir Flutter projesi oluşturun ve projeye bir isim verin (örneğin: **shoppingList_app**). lib/main.dart dosyasını açın.

B) Uygulamanın Ana Yapısını Oluşturma

Kod 1: Flutter Material Paketi Import Etme

```dart
import 'package:flutter/material.dart';
```

- Bu satır, Flutter'ın temel bileşenlerini içeren `material.dart` paketini projeye dahil eder.
- Material Design, Google tarafından geliştirilen modern tasarım standartlarını içerir ve birçok uygulamada yaygın olarak kullanılır.

Kod 2: Ana Fonksiyon (`main`) ile Uygulamayı Başlatma

```
Run | Debug | Profile
void main() {
  runApp(const MyApp());
}
```

- `main()` fonksiyonu, Dart dilinde her uygulamanın çalışmaya başladığı ana giriş noktasıdır.
- `runApp()` fonksiyonu, verilen widget'ı çalıştırarak uygulamayı ekrana getirir. Burada `MyApp` adlı widget uygulamanın kök widget'ı olarak çalıştırılır.
- `const` ifadesi, widget'ın değişmez (immutable) olduğunu belirtir. Böylece performans optimizasyonu sağlanır.

Kod 3: Ana Widget Olarak `MyApp` Tanımlama

```
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      home: ShoppingListScreen(),
      debugShowCheckedModeBanner: false,
    ); // MaterialApp
  }
}
```

- MyApp, uygulamanın ana widget'ıdır ve StatelessWidget sınıfından türetilmiştir.
- StatelessWidget: Değişmez (stateless) widget'lar, durum (state) içermeyen ve sabit kalan widget'lardır. Uygulamanın temel yapısını oluşturmak için kullanılır.
- home: Uygulama açıldığında gösterilecek ana ekranı belirtir. Burada, ana ekran olarak ShoppingListScreen widget'ı tanımlanmıştır.

- MaterialApp, uygulamanın temel yapısını sağlar ve Material Design prensiplerini kullanır. Bu widget, genellikle uygulamanın kök widget'ı olarak kullanılır.
- debugShowCheckedModeBanner: false olarak ayarlanmıştır. Uygulamanın sağ üst köşesinde beliren "Debug" yazısını gizler.
- super.key: Üst sınıfın (StatelessWidget) key parametresini alarak widget'ın benzersizliğini korumasını sağlar. Bu, widget'ın yeniden uygulama içi kullanılmasını sağlar.
- build() fonksiyonu, her widget'ın arayüzünü tanımlayan ana fonksiyondur.

C) Alışveriş Liste Ekranının (Shopping List Screen) Oluşturulması

Kod 1: `ShoppingListScreen` adlı widget'ı ekleme

```dart
class ShoppingListScreen extends StatefulWidget {
  const ShoppingListScreen({super.key});

  @override
  State<ShoppingListScreen> createState() => _ShoppingListScreenState();
}

class _ShoppingListScreenState extends State<ShoppingListScreen> {
  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```

- ShoppingListScreen alınacaklar listesini göstermek için kullanılan **widget sınıfıdır**.
- **StatefulWidget**: Bu sınıf, **durum (state)** içeren ve dinamik olarak güncellenebilen bir widget tanımlar.
- createState() metodu, StatefulWidget için bir durum nesnesi oluşturur.
- Burada _ShoppingListScreenState adında özel bir state sınıfı tanımlanır.
- State nesnesi, widget'ın durumunu saklar ve gerektiğinde kullanıcı arayüzünün (UI) yeniden çizilmesini sağlar.
- Yapılacaklar listesi gibi sürekli değişen içeriklerin yönetilmesi için **State** sınıfı kullanılır. Örneğin: Listeye yeni bir nesne eklendiğinde veya çıkarıldığında arayüz otomatik olarak güncellenir.

- _ShoppingListScreenState sınıfı, State<ShoppingListScreen> sınıfından türetilmiştir. Bu, ShoppingListScreen widget'ının durumunu yönetir.
- _ (alt çizgi) ile başlayan sınıf adları, özel (private) olarak tanımlanır. Yani bu sınıf sadece bu dosyada kullanılabilir.

D) Kullanıcı Arayüzünü Oluşturma

Kod 1: Arayüz Yapısını Oluşturma-1

```
class _ShoppingListScreenState extends State<ShoppingListScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text("Alışveriş Listesi"),
        centerTitle: true,
        backgroundColor: Colors.teal,
      ), // AppBar
    ); // Scaffold
  }
}
```

- build() metodu, uygulamanın kullanıcı arayüzünü oluşturur.
- Scaffold: Sayfanın temel yapısını sağlar. İçinde AppBar, body gibi bölümler bulunur.
- AppBar: Uygulamanın üst kısmında bir başlık çubuğu görüntüler.
    - title: Başlık için bir Text widget kullanılır.
    - centerTitle: true: Başlık metnini ortalar.
    - backgroundColor: AppBar'ın arka plan rengini teal yapar.

Kod 2: Arayüz Yapısını Oluşturma-2

```
      ), // AppBar
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          children: [],
        ), // Column
      ), // Padding
    ); // Scaffold
```

- Padding: Sayfa içeriğine kenarlardan boşluk ekler. Burada, tüm içeriğe 16 piksel boşluk verilmiştir.
- Column: Sayfa içeriğini dikey olarak sıralar.

Kod 3: Kullanıcıdan Metin Girişi Alma (TextField)

```
children: [
  TextField(
    controller: null,
    decoration: const InputDecoration(
        labelText: "Ne alınacak?", border: OutlineInputBorder()),
  ), // TextField
],
```

- **TextField:** Kullanıcıdan metin girişini alır.
  - controller: Kullanıcıdan alınan metni kontrol eder ve kaydeder.
  - InputDecoration: Metin kutusunun tasarımını ve içeriğini belirler.
    - labelText: Metin kutusunun üstünde görünen etiket
    - OutlineInputBorder(): Kutunun etrafında kenarlık gösterir.

Kod 4: Görev Ekleme Butonu (ElevatedButton)

```
  ), // TextField
  SizedBox(height: 15,),
  ElevatedButton(onPressed: null, child: const Text("Ekle")),
],
```

- **ElevatedButton:** Listeye ekleme işlemini başlatmak için kullanılan buton.
  - onPressed: Butona basıldığında fonksiyonunu çalıştırır.
  - child: Butonun üzerindeki metni temsil eder (örneğin, “Ekle”).
- **SizedBox:** İki widget arasında dikey boşluk bırakmak için kullanılır. Burada 15 piksel boşluk eklenmiştir.

E) Alışveriş Listesi İçin State Sınıfının Yapılandırılması

Kod 1: Liste ve TextEditingController Tanımlama

```
class _ShoppingListScreenState extends State<ShoppingListScreen> {
  final List<String> _tasks = [];
  final TextEditingController _controller = TextEditingController();
```

- **_tasks** adında bir liste tanımlanmıştır ve bu liste, kullanıcı tarafından eklenen liste öğelerini saklar.
- **List<String>**, bir **liste veri yapısıdır** ve bu listenin sadece **String** türünden öğeler içereceğini belirtir.
- Liste başlangıçta boş olarak başlatılır ([]), ancak kullanıcı öğe ekledikçe listeye elemanlar eklenecektir.
- **`TextEditingController`**, bir **`TextField`** **widget'ının** içeriğini programatik olarak yönetmek için kullanılır.
- **_controller** görev eklemek için kullanılan metin kutusunun (TextField) içeriğini kontrol etmek için kullanılır.
- Örneğin: Kullanıcı bir öğe girdiğinde, bu kontrolör aracılığıyla metin alınır ve işlem tamamlandıktan sonra kutu temizlenir.

Kod 2: Görev Ekleme Fonksiyonu Tanımlama

```
final TextEditingController _controller =

void _addTask() {
  if (_controller.text.isNotEmpty) {
    setState(() {
      _tasks.add(_controller.text);
    });
    _controller.clear();
  }
}
```

- **_addTask()** fonksiyonu metin kutusuna girilen öğeyi listeye ekler.
    1. **if (_controller.text.isNotEmpty):**
        - Kullanıcının girdiği metin boş değilse (yani bir metin yazılmışsa), aşağıdaki işlemler gerçekleştirilir.
        - Bu kontrol, boş görevlerin eklenmesini önler.
    2. **setState():**
        - setState() fonksiyonu, state'in değiştiğini bildirir ve arayüzü günceller.
    3. **_tasks.add(_controller.text):**
        - Kullanıcının metin kutusuna girdiği içerik, _tasks listesine eklenir.
    4. **_controller.clear():**
        - Görev eklendikten sonra, metin kutusunun içeriği temizlenir, böylece kullanıcı yeni bir görev girmeye hazır olur.

F) Çağrılma İşlemlerinin Tamamlanması

Kod 1: Text Field'a _controller'ın çağırılması

```
TextField(
  controller: null,
```

```
TextField(
  controller: _controller,
```

- **controller TextField widget'ının** içeriğine erişmek veya güncellemek için kullanılan bir özelliktir.
- Burada girilen metin değerini **_controller** değişkenine atamış oluruz.

Kod 2: OnPressed fonksiyonunda addTask'ın çağırılması

```
  height: 15,
), // SizedBox
ElevatedButton(onPressed: null, child: const Text("Ekle")),
```

```
), // SizedBox
ElevatedButton(onPressed: _addTask, child: const Text("Ekle")),
```

G) Listeyi Görüntülemek

Kod 1: ListView Kullanma

```
ElevatedButton(onPressed: _addTask, child: const Text("Ekle")),
const SizedBox( height: 20,),
Expanded(
    child: ListView.builder(
        itemCount: _tasks.length,
        itemBuilder: (context, index) {
          return ListTile(
            title: Text(_tasks[index]),
            trailing: IconButton(
                onPressed: () {
                  setState(() {
                    _tasks.removeAt(index);
                  });
                },
                icon: const Icon(Icons.delete)), // IconButton
          ); // ListTile
        })) // ListView.builder // Expanded
```

- **SizedBox:** İki widget arasında dikey boşluk bırakmak için kullanılır.
- **Expanded:** ListView'in mevcut alanı doldurmasını sağlayan widget'tır böylece listeye istediğimiz kadar öğe girebiliriz.
- **ListView.builder:** Liste öğelerini dinamik olarak oluşturmak için kullanılır.

  itemCount: Listede kaç öğe olacağını belirtir (burada _tasks.length kullanılır).

  itemBuilder: Liste öğelerinin nasıl görüneceğini belirler. Her öğe için bir ListTile oluşturulur.
- **ListTile:** Tek bir liste öğesini temsil eder.

  title: Öğenin adını görüntüler.

  trailing: Sağ tarafta, öğe silme işlemi için bir IconButton gösterir.
  - **IconButton:** Silme butonunun işlevini tanımlar.
  - **onPressed:** Butona basıldığında ilgili görevi listeden çıkarır. setState() çağrılarak arayüz güncellenir

Tüm Kodlar

```
import 'package:flutter/material.dart';


void main() {

runApp(MyApp());

}


class MyApp extends StatelessWidget {

const MyApp({super.key});


@override

Widget build(BuildContext context) {

return const MaterialApp(

home: ShoppingListScreen(),

debugShowCheckedModeBanner: false,

);

}

}


class ShoppingListScreen extends StatefulWidget {

const ShoppingListScreen({super.key});


@override

State<ShoppingListScreen> createState() => _ShoppingListScreenState();

}


class _ShoppingListScreenState extends State<ShoppingListScreen> {

final List<String> _tasks = [];
```

```
final TextEditingController _controller = TextEditingController();



void _addTask() {

if (_controller.text.isNotEmpty) {

setState(() {

_tasks.add(_controller.text);

});

_controller.clear();

}

}



@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: const Text("Alışveriş Listesi"),

centerTitle: true,

backgroundColor: Colors.teal,

),

body: Padding(

padding: const EdgeInsets.all(16.0),

child: Column(

children: [

TextField(

controller: _controller,

decoration: const InputDecoration(

labelText: "Ne alınacak?", border: OutlineInputBorder()),

),
```

```
const SizedBox(

height: 15,

),

ElevatedButton(onPressed: _addTask, child: const Text("Ekle")),

const SizedBox( height: 20,),

Expanded(

child: ListView.builder(

itemCount: _tasks.length,

itemBuilder: (context, index) {

return ListTile(

title: Text(_tasks[index]),

trailing: IconButton(

onPressed: () {

setState(() {

_tasks.removeAt(index);

});

},

icon: const Icon(Icons.delete)),

);

}))

],

),

),

);

}

}
```