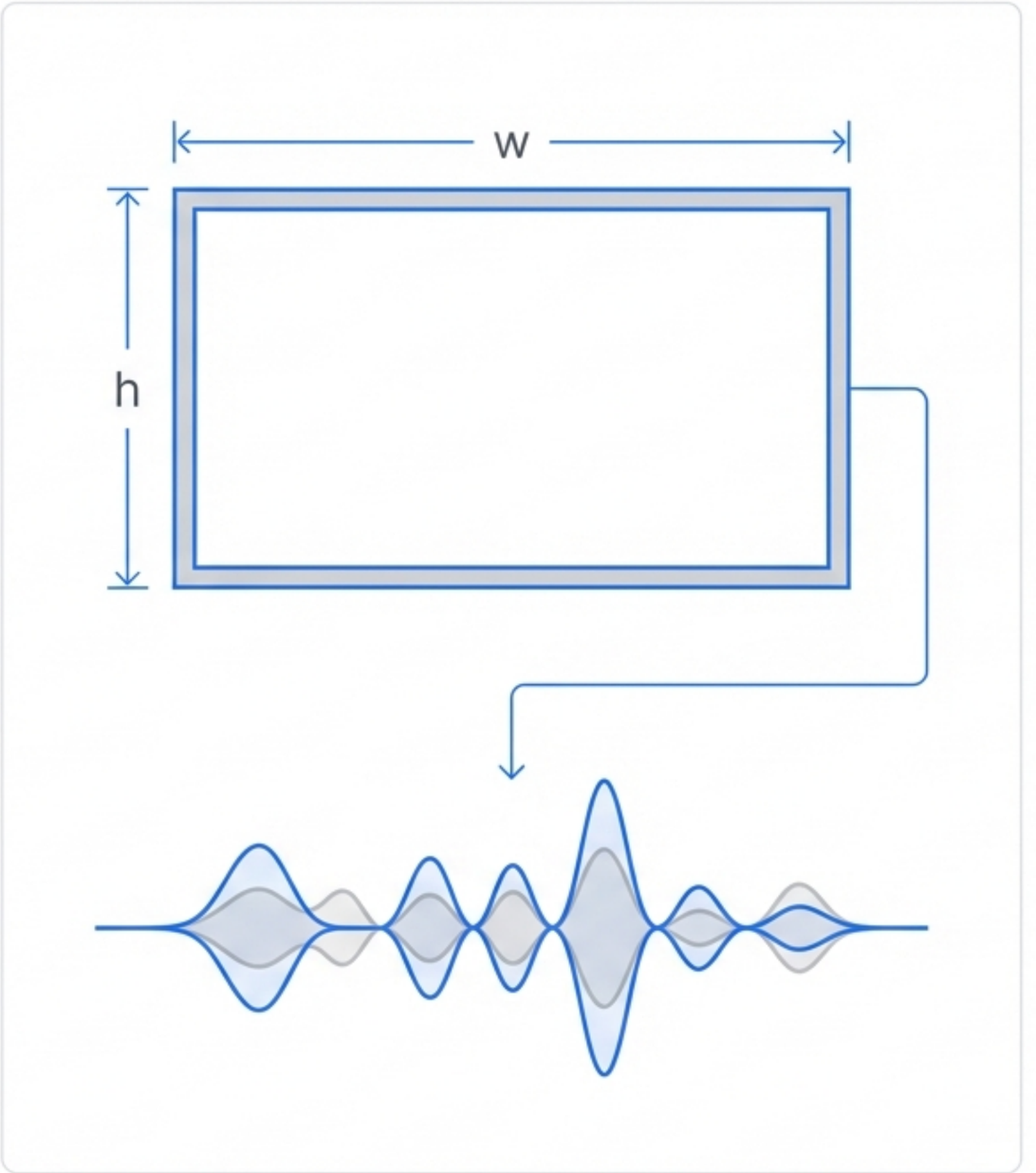
Flutter'da Multimedya Entegrasyonu

`video_player` ve `assets_audio_player` Paketleri ile Video ve Ses Yönetimi



Bu rehber, kod blokları ve mimari şemalar içeren teknik bir el kitabıdır.

İki Farklı Yaklaşım: Video ve Ses

Flutter'da medya yönetimi, içerik türüne göre farklı mimari desenler gerektirir.

Video (Görüntü)

- **Paket:** `video_player`
- **Mimari:** `FutureBuilder` ve Controller tabanlı.
- **Zorluk:** Kaynak yönetimi (Bellek sızıntılarını önlemek için `dispose` kritiktir).

Ses (Müzik)

- **Paket:** `assets_audio_player`
- **Mimari:** `StreamBuilder` ve olay tabanlı (Event-driven).
- **Avantaj:** Anlık durum güncellemeleri için reaktif yapı.

Video Modülü: Kurulum ve Bağımlılıklar

Video Kaynakları

Paket üç farklı kaynaktan video oynatmayı destekler:

- **1. Asset:** Uygulama içine gömülü dosyalar.
- **2. Network:** İnternet üzerindeki URL adresleri.
- **3. File:** Cihazın dosya sistemindeki videolar.

pubspec.yaml

```yaml
dependencies:
  flutter:
    sdk: flutter
  video_player: ^latest_version
```

Durum Yönetimi: VolumeManager

Video sesi değiştirildiğinde tüm arayüzü yeniden çizmek yerine, sadece Slider'ı güncellemek için `Provider` kullanılır.

```dart
class VolumeManager with ChanageNotifier {
  var _volume = 50.0;
  double get volume => _volume;

  void setVolume({required double volumeValue,
    required VideoPlayerController controller}) {
    _volume = volumeValue;
    controller.setVolume(_volume); // Denetleyiciyi güncelle
    notifyListeners();
  }
}
```

Arayüz Güncellemesi

Denetleyici Yaşam Döngüsü (Controller Lifecycle)

Video oynatıcı kaynak tüketimi yüksek bir bileşendir. `**VideoPlayerController**` başlatılmalı (`**initState**`) ve iş bittiğinde mutlaka yok edilmelidir (`**dispose**`).

```
@override
void initState() {
  super.initState();
  controller = VideoPlayerController.asset("assets/butterfly.mp4");
  controller.setLooping(true);
  initVideo = controller.initialize(); // Başlatma işlemi
}
```

```
@override
void dispose() {
  controller.dispose(); // KRİTİK: Bellek sızıntısını önler
  super.dispose();
}
```

Başlatma Mantığı: FutureBuilder

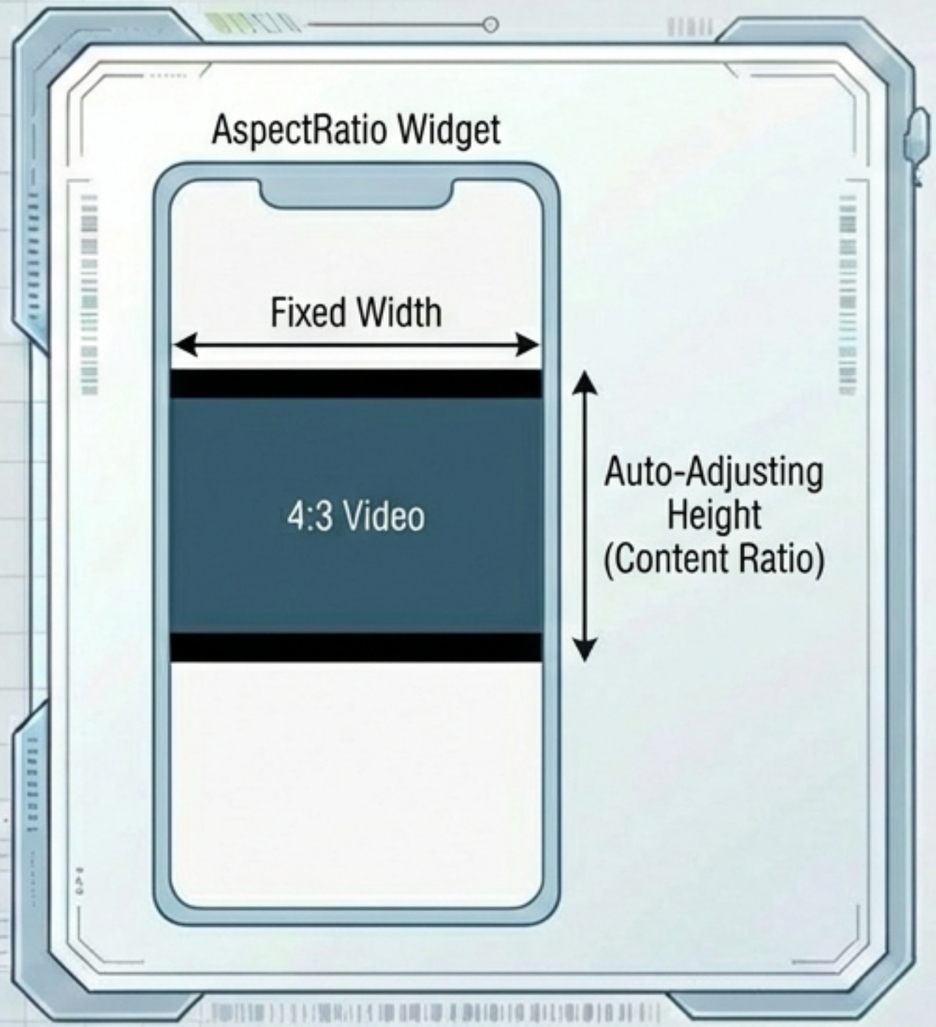
Denetleyici veri döndürmez, sadece durum bildirir. Bu yüzden `snapshot.hasData` yerine `snapshot.connectionState` kontrol edilir.

```
FutureBuilder(
  future: initVideo,
  builder: (context, snapshot) {
    ...
    if (snapshot.connectionState == ConnectionState.done) {
      return PlayWidget(controller); // Video hazır
    }
    return const Center(child: CircularProgressIndicator()); // Yükleniyor
  },
);
```

Video yüklendi ve oynamaya hazır

Arayüz Düzeni: AspectRatio

Videoların genişlik/yükseklik oranı bozulmadan ekrana yerleştirilmelidir.




```
AspectRatio(
  aspectRatio:
    controller.value.aspectRatio,
  child: VideoPlayer(controller),
),
```

`VideoPlayerController` videonun oranını otomatik olarak hesaplar.

Etkileşim: Oynatma ve Ses Kontrolü

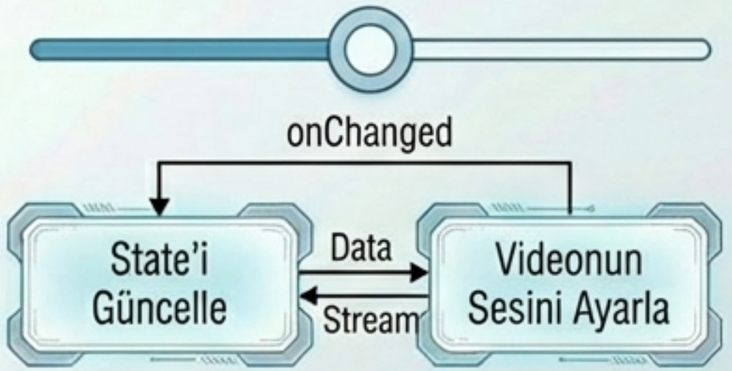
Oynat/Duraklat (Play/Pause)

Mantık: `controller.value.isPlaying` durumunu kontrol ederek tersini uygular.

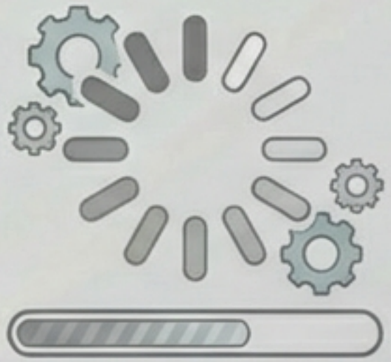
```
if (!controller.value.isPlaying)
  controller.play();
else
  controller.pause();
}
```

Ses Slider'ı

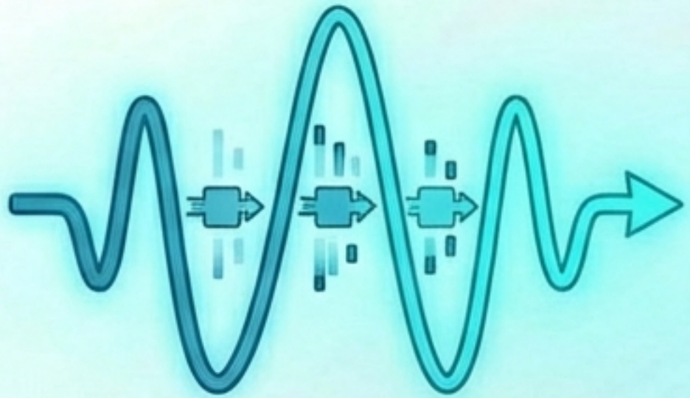
`Consumer` widget'ı kullanılarak `VolumeManager` dinlenir. `onChanged` olayı, hem state'i günceller hem de videonun sesini ayarlar.



Ses Modülüne Geçiş: Akışlar (Streams)



Video (Future)

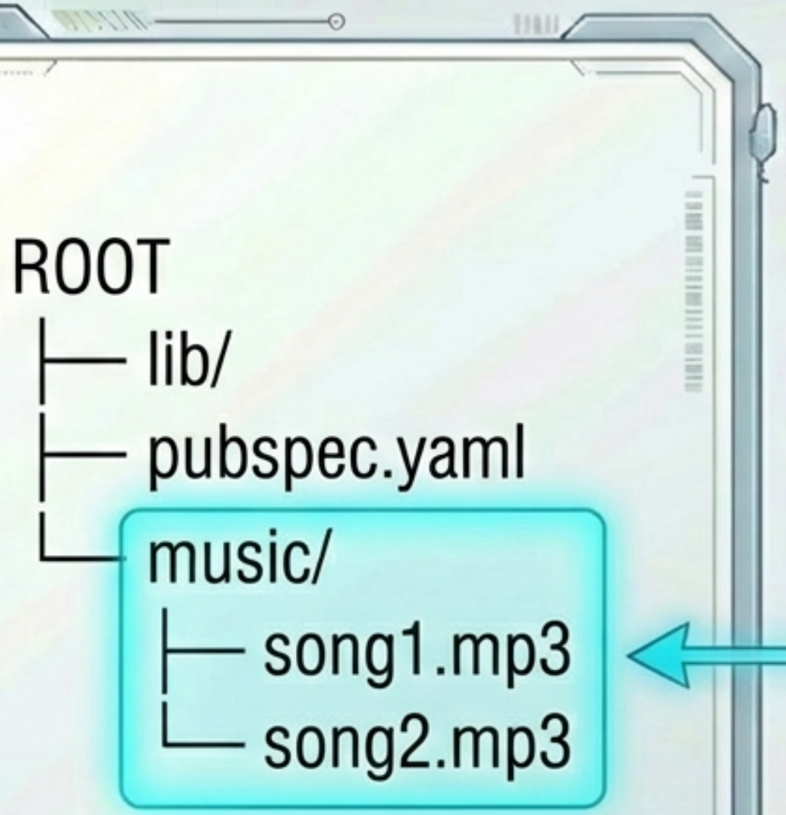


Ses (Stream)

Video yönetiminden farklı olarak, **`assets_audio_player`** paketi yoğun olarak Stream (Akış) yapısını kullanır.

`assets_audio_player`

Ses Kurulumu ve Dosya Yapısı




```
dependencies:
  assets_audio_player: ^2.0.9+2

flutter:
  assets:
    - music/
```

Klasör yolu `**pubspec.yaml**` dosyasında tam olarak belirtilmelidir.

Oynatıcı Mantığı ve API

Statik tanımlama ile performans optimizasyonu ve basit API kullanımı.

```dart
// Statik tanımlama
static final _assetsAudioPlayer = AssetsAudioPlayer();

// Dosya açma
_assetsAudioPlayer.open(Audio("assets/music/song1.mp3"));

// Kontrol metodları
_assetsAudioPlayer.play();
_assetsAudioPlayer.pause();
_assetsAudioPlayer.stop();
```

Dosyayı yükler ve hazırlar

`assets_audio_player`

Reaktif Arayüz: Süre Takibi

_assetsAudioPlayer.currentPosition` akışını dinleyerek geçen süreyi ekrana yazdırır.

```
StreamBuilder(
  stream: _assetsAudioPlayer.currentPosition,
  builder: (context, asyncSnapshot) {
    final time = asyncSnapshot.data;
    if (time != null) {
      return Text("${time.inMinutes.remainder(60)}m "
                  "${time.inSeconds.remainder(60)}s");
    }
    return const Text("0m 0s");
  },
```

`asyncSnapshot.data` nullable kontrolü önemlidir.

`assets_audio_player`

Reaktif Arayüz: Oynatma Durumu

Anlık durum (`**isPlaying**`) akışına göre buton ikonunu değiştirme.

Temiz Kod İpuçları (Best Practices)

Kod okunabilirliğini artırmak için metod referansları (tear-off) tercih edilmelidir.

Lambda (Daha uzun)

```
onPressed: () => _assetsAudioPlayer.play()
```

Tear-off (Önerilen)

```
onPressed: _assetsAudioPlayer.play
```

Tear-off söz dizimi daha temizdir ve gereksiz closure oluşturmaz.

`assets_audio_player`

Özet ve Çıkarımlar

Video	Ses
Başlatma: `FutureBuilder`	Başlatma: `AssetsAudioPlayer.open()`
Kaynak Yönetimi: `dispose()` şart	Durum Takibi: `StreamBuilder`
Düzen: `AspectRatio`	Arayüz: Reaktif Güncelleme

Multimedya uygulamalarında kullanıcı deneyimi, arayüzün anlık tepki verme hızına (Reactivity) ve kaynakların doğru yönetilmesine bağlıdır.

`assets_audio_player`