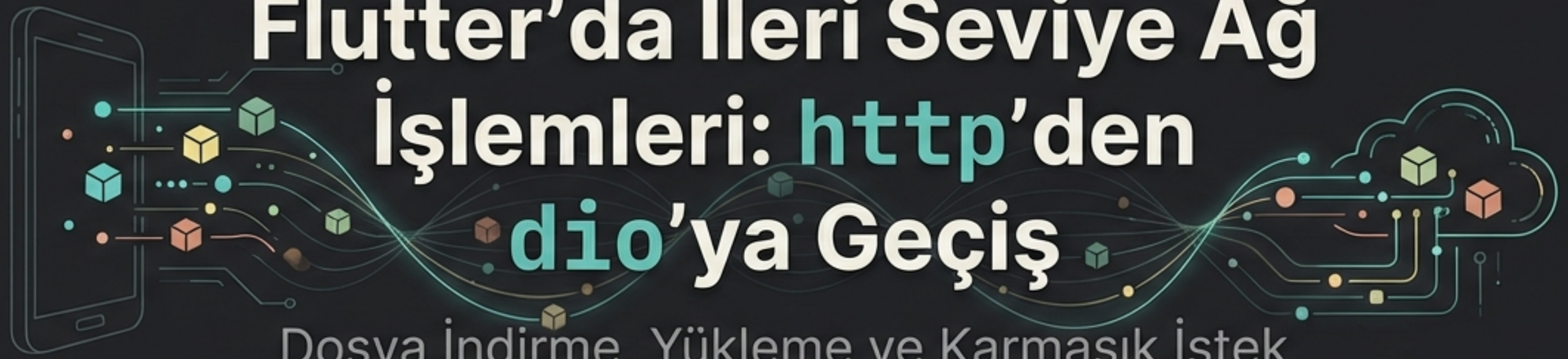
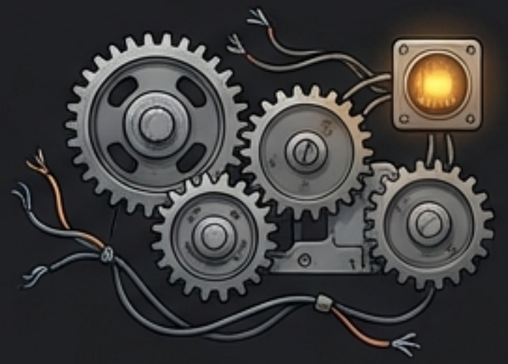


Flutter’da İleri Seviye Ağ İşlemleri: `http`’den `dio`’ya Geçiş

Dosya İndirme, Yükleme ve Karmaşık İstek Yönetimi İçin Teknik Bir Kılavuz

İkilem: HTTP mi, Dio mu?

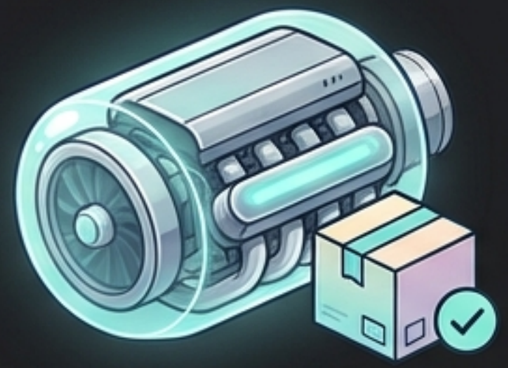
`http` Paketi



Temel istekler için verimlidir ancak I/O işlemleri "düşük seviye" (low-level) kalır.

- Bayt (Bytes) ve kodlamalarla (encodings) manuel uğraş gerekir.
- `downloadFile()` gibi hazır fonksiyonlar yoktur.
- Dosya nesnelerini yönetmek geliştiriciye kalır.

`dio` Paketi



Dosya indirme/yükleme ve karmaşık ağ görevleri için güçlü, hazır fonksiyonlar sunar.

- Sezgisel API yapısı.
- I/O işlemleri için basitleştirilmiş mantık.

Uygulamanız veri indirme veya yükleme (I/O) işlemi yapacaksa, sadeliği nedeniyle `dio` önerilir.

Dio ile Hızlı Başlangıç: Temiz Kod

```dart
final dio = Dio();

// GET İsteği - JSON serialize işlemine gerek yok
final response = await dio.get("https://website.com");
print(response.data);

// POST İsteği - Veri nesnesi doğrudan geçilebilir
await dio.post("https://website.com",
  data: {
    "key1": "1",
    "key2": 2,
  });
});
```

Query Parameters

`http` paketinde URI objesi oluşturmak gerekirken, `dio` ile `queryParameters` bir `Map` olarak kolayca geçilebilir:

```dart
await dio.get(url, queryParameters:
  {"param1": 1, "param2": "2"});
```

Gücü Ölçeklemek: Eşzamanlı İstekler (Concurrency)

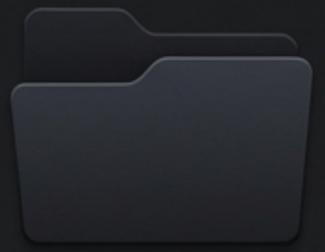

```dart
// Temel ayarları tek seferde tanımlayın
final options = BaseOptions(
  baseUrl: "https://website.com/api/",
  connectTimeout: 3000, // 3 saniye
);
final dio = Dio(options);
```

`BaseOptions` sayesinde URL tekrarı yok, sadece relative path.

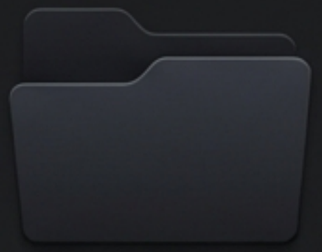
```dart
// 3 isteği aynı anda çalıştır ve hepsini bekle
final response = await Future.wait([
  dio.get("/version"),
  dio.get("/products/list"),
  dio.post("/login", options: Options(
    (headers: {"Authentication": "auth-key"})),
]);
```

İndirme Öncesi Hazırlık: Dosya Yolu Yönetimi

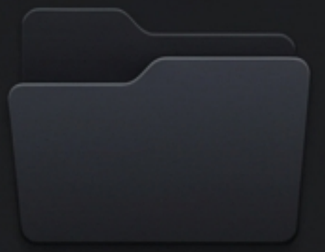
Dosyaları cihaza kaydetmek için işletim sistemi dizinlerine güvenli erişim gereklidir. Flutter ekibi tarafından geliştirilen `path_provider` paketi bu standart dizinleri `Directory` objesi olarak döndürür.



Temp



External



Downloads

```dart
final tempDir = await getTemporaryDirectory();
final extDir = await getExternalStorageDirectory();
final downloadDir = await getDownloadsDirectory();
// Erişmek için: tempDir.path
```

Bu sunumdaki örneklerde cihazın geçici (temporary) dizini kullanılacaktır.

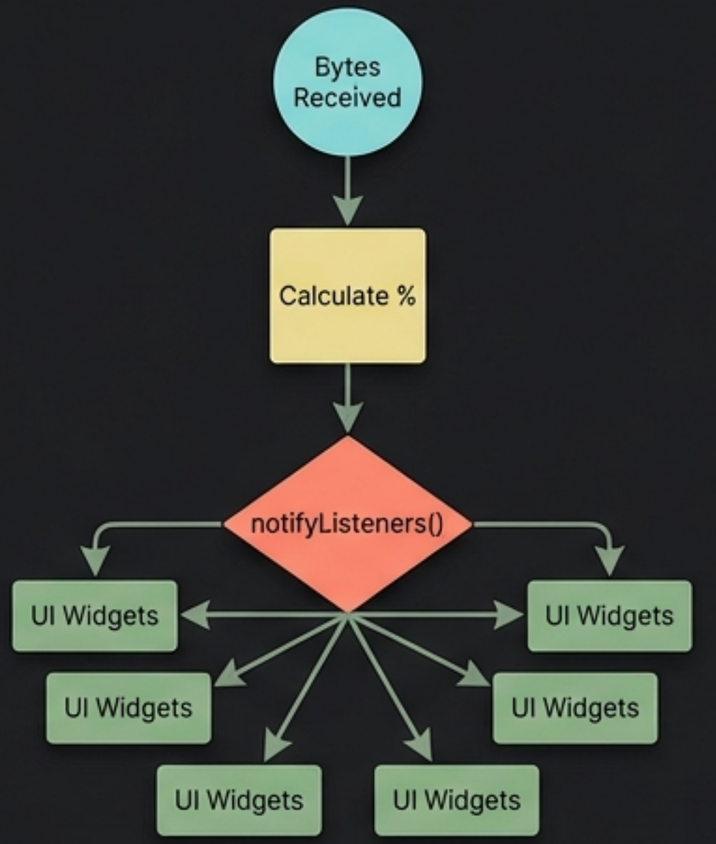
İlerleme Takibi Mimarisi (Logic)

Kullanıcıya indirme yüzdesini göstermek için mantığı UI'dan ayırıyoruz. `DownloadProgress` sınıfı ağ mantığını kapsüller.

```
class DownloadProgress with ChanageNoFeider {
  var _progress = 0.0;
  double get progress => _progress;

  void start({required String url, required String filename}) async {
    _resetProgress();
    // İndirme mantığı buraya gelecek...
  }

  void _updateProgress(double value) {
    _progress = value;
    notifyListeners(); // UI'ı tetikler
  }
}
```



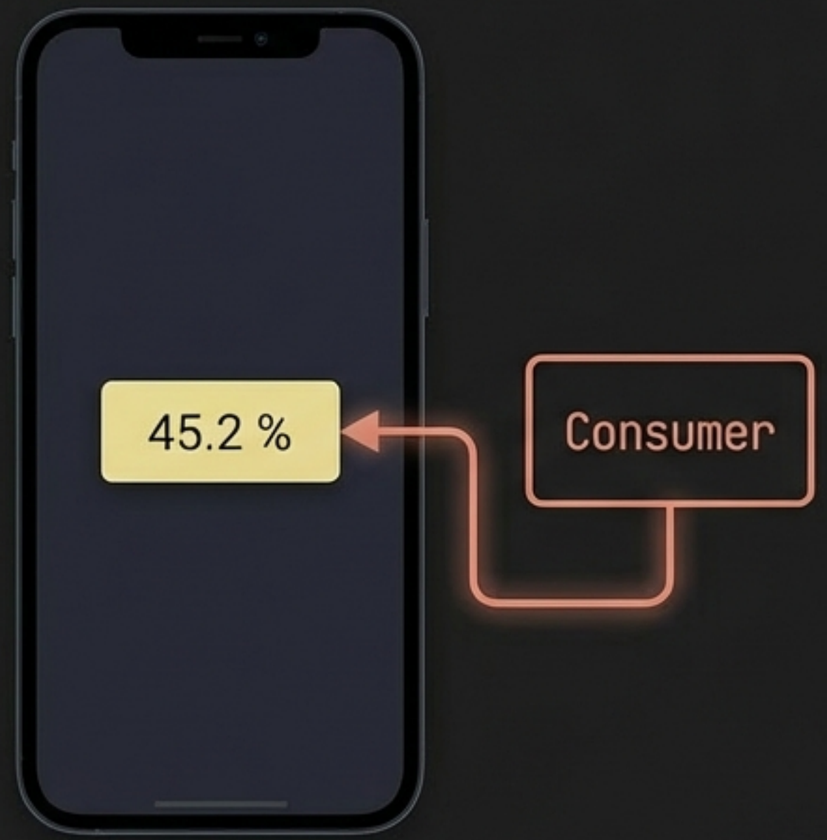
Dio ile İndirme İşlemi (The Code)

```
await Dio().download(url, "${directory.path}/$filename",
  options: Options(
    headers: {HttpHeaders.acceptEncodingHeader: "*"} // Gzip sıkıştırma sorunu için
  ),
  onReceiveProgress: (received, total) {
    if (total != -1) {
      // İndirilen bayt / Toplam dosya boyutu
      var pos = received / total * 100;
      _updateProgress(pos);
    }
  }
);
```

⚠ **Dikkat:** Gzip sıkıştırması nedeniyle toplam dosya boyutu bazen kullanılamayabilir. Bu durumda `total` değeri -1 döner; hesaplama yapmadan önce mutlaka kontrol edilmelidir.

Kullanıcı Arayüzü Entegrasyonu

Görsel Tasarım & Bağlantı



Kod Uygulaması

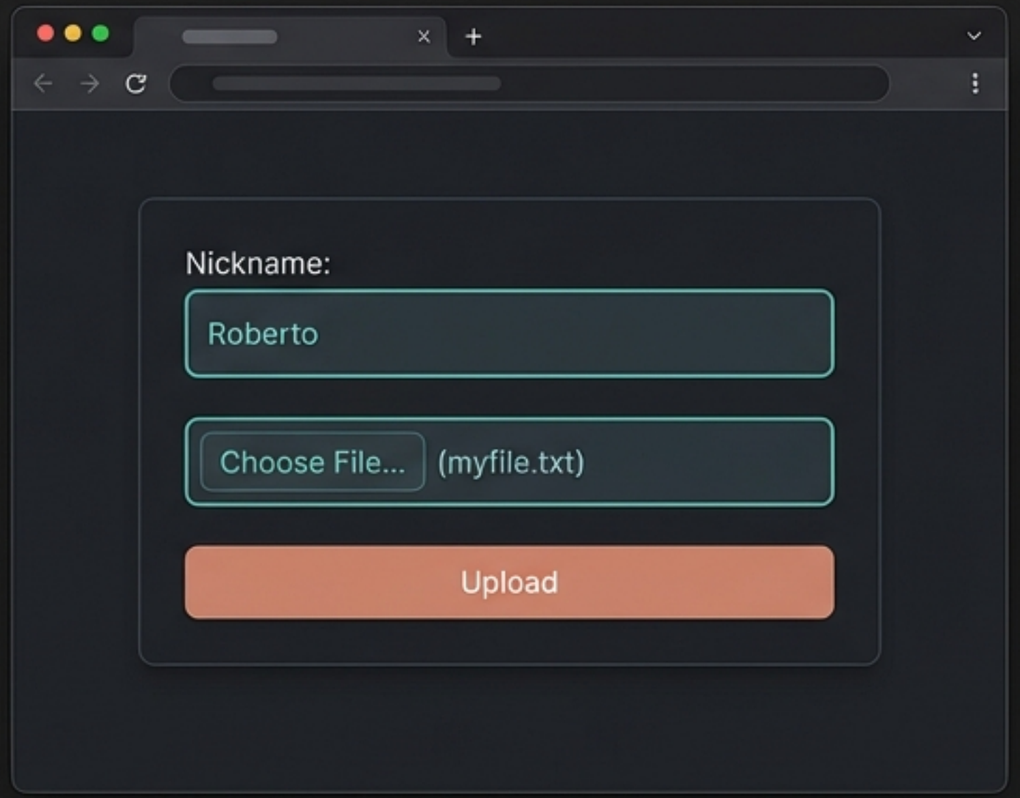
Sadece bu widget yeniden çizilir (Rebuilds).

```
Consumer(
  builder: (context, status, _) {
    var progress = status.progress.toStringAsFixed(1);
    return RaisedButton(
      child: Text("$progress %"),
      onPressed: () => status.start(
        url: "https://website.com/files/test.pdf",
        filename: "myfile.pdf"
      ),
    );
  },
)
```

Senaryo: Form Doldurma ve Veri Yükleme

Bir HTML formu doldurur gibi sunucuya metin ve dosya göndermek için POST isteği ve FormData kullanılır.

Concept



Flutter

```dart
final payload = FormData.fromMap({
  "nickname": "Roberto",
  "file": await MultipartFile.fromFile(filePath),
});

// Varsayılan encoding: "multipart/form-data"
await dio.post("/admin/adduser", data: payload);
```

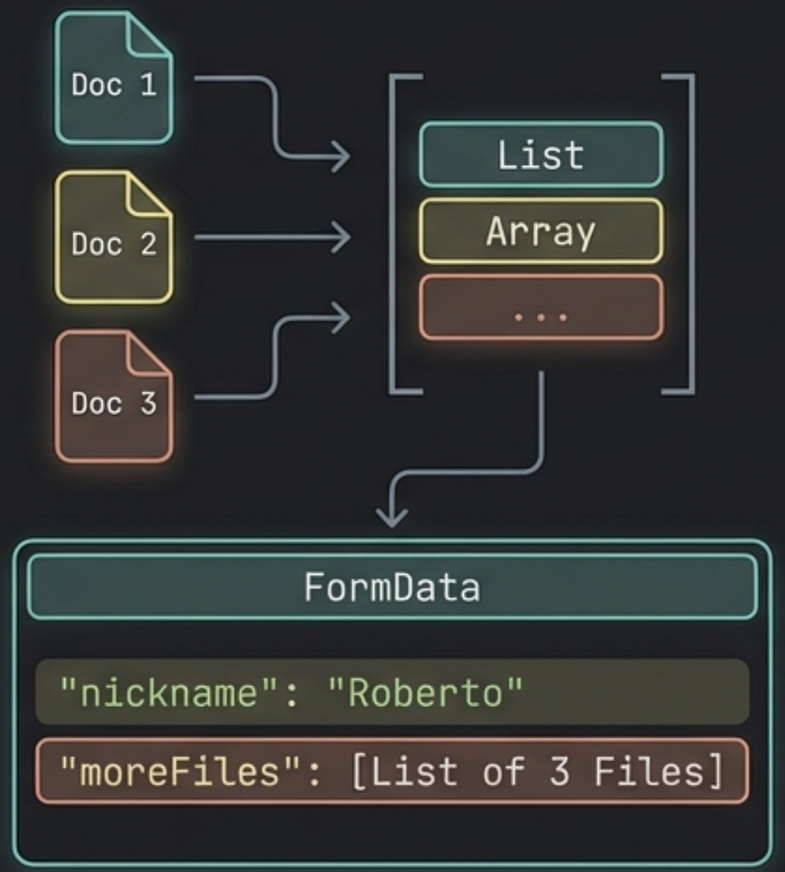
Çoklu Dosya Gönderimi (Multipart)

Kod

```dart
final payload = FormData.fromMap({
  "nickname": "Roberto",
  "moreFiles": [
    await MultipartFile.fromFile(filePath1),
    await MultipartFile.fromFile(filePath2),
    await MultipartFile.fromFile(filePath3),
  ]
});
```

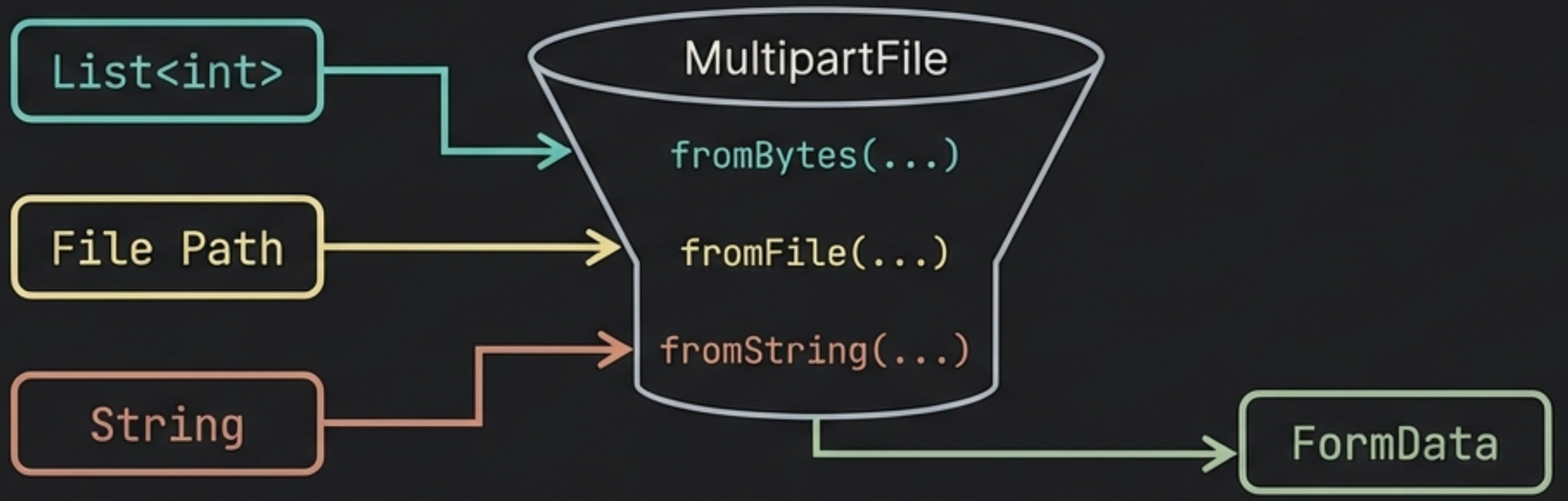
Tek bir istek içinde birden fazla dosya göndermek için dosyalar bir liste (array) içinde tanımlanabilir.

Visual



Yükleme Takibi ve Esneklik

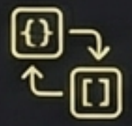
```dart
await Dio().post(url, data,
  onSendProgress: (sent, total) {
    var pos = sent / total * 100;
    _updateProgress(pos);
  }
);
```



Neden Dio'da Kalmalıyız?

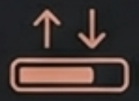
Tutarlı API

Veri formatlarını dönüştürmek ve obje oluşturmak için hazır araçlar.



Otomatik Dönüşümler

JSON serialize işlemleri ve dosya encoding işleri otomatiktir.



İlerleme Takibi

Hem indirme (receive) hem yükleme (send) için yerleşik callback'ler.

Esnek I/O

Dosyalar, baytlar veya stringler kolayca paketlenip (FormData) gönderilebilir.

Basit HTTP istekleri için seçim sizin; ancak dosya ve veri operasyonları içeren uygulamalarda `dio` tekerleği yeniden icat etmenizi engeller.