Görsel Programlama

Dosyalama

Emir ÖZTÜRK

Dosya Türleri

- Binary
  - Byte değerleri
  - 65,66,67 -> ABC
- Text
  - Ascii, utf8, unicode
  - 65,66,67 -> 656667

Dosyalar binary veya text olarak saklanabilmektedirler. Text dosyaları belirli encoding kurallarını kullanarak okunabilir metnin saklanmasında kullanılır. Binary dosyalarda ise tamamen byte değerleri saklanır. Örneğin text bir dosyaya 65,66 ve 67 yazıldığında ekranda 656667 gözükecekken, binary dosyanın text olarak açılması durumunda bu byte değerlerine karşılık gelen ascii kodları gözükecek ve ekrana ABC yazdırılacaktır.

Dosya Türleri

- Sıralı
  - Yavaş
  - Farklı kayıt boyutu
  - Az yer ihtiyacı
- Rastgele
  - Hızlı
  - Çok yer ihtiyacı
  - Sabit kayıt boyutu

Dosyalar sıralı veya rastgele olarak da açılabilir. Sıralı dosyalarda kayıt boyutu belli değildir. Bu sebeple okuma sırasında bir kaydı bulabilmek adına tüm kayıtlara baştan bakmak gerekir. Rastgele dosyalarda ise belirli bir blok boyutu vardır ve bu boyut bilindiğinde bloklar arası hızlı geçiş sağlanabilir. Verinin blok boyutundan küçük olması durumunda rastgele dosyalar sıralı dosyalara göre daha çok yer kaplayacaklardır

Dosyaların bulunduğu yer

- Cihaz içi dosyalar
  - path_provider
  - İki yere erişim
    - Temp ve Document dizinleri
  - Cihaza bağlı dizin isimleri
- Uzaktan gelen akışlar

Masaüstü platformlarda dosya sistemlerine erişim mümkün olsa da mobil cihazlarda bu işlem sınırlı bir biçimde gerçekleştirilmektedir. Path_provider paketi ile temp ve document dizinlerine erişim sağlanabilmektedir.

Dosya referansı oluşturma

- Referans
  - Future<File> dosyaAl() async{
    final path = _lokalYol;
    return await File("dosyaYolu");
    }
- Future

Bir dosyaya erişebilmek için flutter'da dosyaya referans oluşturmak gerekmektedir. Dosya alma işlemi sırasında uygulamanın cevap verebilmeye devam etmesi adına asenkron metot kullanılabilir. Asenkron metotlar döndüreceği değeri Future nesnesi içerisinde döndürmektedir. Bu sayede fonksiyon işlemini bitirene kadar ui tarafının işler halde kalması sağlanmaktadır.

Future

- Asenkron
- Ana thread
- Uygulama arkaplan işleri
- Uygulamanın cevap verebilirliği (Responsiveness)

Asenkron uygulamalar ana thread'den bağımsız çalışırlar. Masaüstünde farklı uygulamalar arası geçiş mobil cihazlara göre daha kolay olduğu için future kullanılmadan da arayüz tasarımı gerçekleştirilebilmektedir. Hem uygulamanın yaptığı işi takip edebilmek adına hem de uygulamanın çalışır olduğunu göstermek adına bloklamadan çalışması önemlidir.

Future

- Asenkron fonksiyon çağırımı
  - Dönen tamamlanmamış future
- Asenkron fonksiyon çağırımı tamamlandığında
  - future nesnesinin değerini alma
- Future değeri için Future<Tür>

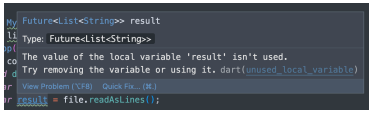
Asenkron fonksiyonlar çağırıldığında yeni bir thread ile çalıştırılırlar ve ana döngü çalışmaya devam eder. Bu fonksiyonlardan future nesnesi döner ve döndürdükleri değer bu future nesnesi içerisinde yer alır.

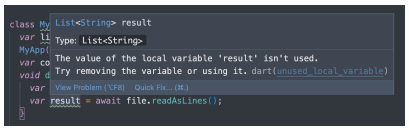
Asenkron fonksiyon
Bitişin beklenmesi

- Asenkron fonksiyonların bitmesini beklemek için
  - await
- await kullanacak metodun async olması gerekir
- Future<Tür> nesnesindeki tür otomatik olarak elde edilir

Future nesnesi içerisindeki değeri almak ya da asenkron fonksiyonun sonucunu beklemek için await kelimesi kullanılabilir. Bu sayede fonksiyon asenkron olsa da bitimine kadar ana döngü fonksiyonu bekleyecektir.

Örnek
Veri türünün değişimi





Asenkron fonksiyon
Hata eldesi

- Asenkron bir fonksiyonun başarısız olma durumu
- Fonksiyonda try catch
- Fonksiyonu çağıran fonksiyonda try catch
- Catch edilen hatanın yukarıya iletilmesi

Asenkron fonksiyon çalışmayı durdurabilir veya sonuç döndürmeme ihtimali bulunmaktadır. Bu durumda asenkron fonksiyonun içerisinde try catch ile hata yakalanabilir. Bunun dışında fonksiyonun değer döndürmesi aşamasında da try catch ile çağıran fonksiyonda hata kontrolü yapılabilir.

Dosyaya Yazma

- Referansı await ile alma
- writeAs__() metotları
  - await
  - then
- await değişkeni almak için kullanılabilir
- then elde edilen değişkeni parametre alan bir metot yazmak için kullanılabilir

Dosya referansı alındıktan sonra dosyaya yazmak için writeAsString metodu kullanılabilir. Bu metot asenkron olarak çalışacağı için await ile beklenebilir veya then ile fonksiyon sonucunda başka bir fonksiyon çalıştırılabilir.

Dosyadan Okuma

- Referansı await ile alma
- readAs___() ile okuma
  - await ya da then
- Başka türlerde kullanılacaksa
  - parse

Dosyadan okuma yapmak için readAsString kullanılabilir. Bu durumda okunan değerin tekrar diğer değişkenle dönüştürülme ihtiyacında parse işlemleri kullanılacaktır.

Akışlar

- Stream
- Dosya boyutu çok büyük olabilir
- Dosya string şeklinde okunmayabilir
  - Binary

Dosyalar çok büyük olduğunda tüm dosyayı belleğe okumak mümkün olmayacaktır. Ayrıca dosyalar binary olarak okunmak istenebilir. Böyle bir durumda stream'ler kullanılabilir.

Stream okuma ve yazma

- File()
  - openRead()
  - openWrite()
  - Stream döndürür
- stream.read()
- stream.write()
- stream.close()

Stream olarak dosyaları okumak için File üzerinde openRead ve openWrite kullanılabilmektedir. Bu metotlar stream döndürmektedir ve stream üzerinde read, write işlemleri gerçekleştirilmektedir.

Anahtar değer saklama / okuma

- dependencies altına shared_preferences eklenmeli
- Prefs = SharedPreferences.getInstance()
- prefs.setInt("anahtar",değer)
- prefs.getInt("anahtar")
- prefs.remove("anahtar")

Ayrıca flutter içerisinde lokal bir değişkende anahtar değer ikilileri okunabilmektedir. SharedPreferences elde edildikten sonra anahtara karşılık gelen bir değer get-set işlemi ile okunup yazılabilir.