Library App (Firebase Cloud Firestore & MVMM)

Özet ve Konu Başlıkları

Library app, kullanıcıların yaptıkları işlemleri anında Firestore tarafında da gerçek zamanlı güncelleyen ve kaydedebilen, Firestore ile ilgili temel CRUD işlemlerini gerçekleştirebilen metot ve servislere sahip, Kullanıcıların kitap listeleri arasında arama yapabildiği, çeşitli widget ve paketlerin kullanıldığı ve MVMM mimarisi ile oluşturulan bir kütüphane uygulamasıdır.

Kullanıcılar yeni bir kitap ekleyebilir ve eklenilen kitapları ana sayfadan arama yaparak filtreleyebilir. Kullanıcılar, eklenilen kitapları düzenleyerek güncelleyebilir veya silebilir, eklenilen kitaplar için her birine ait ayrı bir ödünç kayıt listesi oluşturabilir.

```
CollectionReference & DocumentReference
get() metodu ile manuel 'DocumentSnapshot' çekme
get() metodu ile manuel 'CollectionSnapshot' çekme
StreamBuilder Widget Kullanımı
StreamBuilder ile doküman dinleme
StreamBuilder ile koleksiyon dinleme
Alternatif metot: StreamProvider Kullanımı
Doküman Ekleme - add( ) metodu
Doküman Ekleme / Güncelleme - set ( ) & update ( )
Doküman Silme - delete( )
MVVM için Refactoring
Book Model Sınıfı ekleme
Database Servisi Oluşturma
VievModel güncelleme
StreamBuilder güncelleme
Veri Giriş Formu Ekleme
DateTime Picker Kullanımı
Servis Oluşturma
Add Book Metotları
Add Book View Güncelleme
Update Book Sayfaları
flutter_slidable paketi kullanımı
Filtreleme TextField Alanı
ListView Filtreleme Mantığı ve Uygulama
Ödünç Kayıtlar Sayfası Ekleme
Ödünç Kayıt Modeli
Book Model güncelleme
addBook metodu güncelleme
updateBook metodu güncelleme
Firesbase Storage Kurulum
Storage dosya erişimi
image_picker paketi
Storage dosya yükleme
Imaj boyut ve kalite ayarı
Yeni kayıttan vazgeçme senaryosu
İşlem iptali fonksiyonu
```

Cloud Firestore Veri Yapısı

NOSQL dir. Veriler json formatındaki gibi daha esnek yapıda saklanabilir.

Flutter da *Sqflite* ve *Hive* gibi paketler ile local veri tabanı oluşturulabiliyor.

Önceden yalnızca real time veri tabanı vardı. Bu tip veri tabanında veriler kocaman bir json, map yapısı halinde tutulur.

Cloud store da veriler nasıl tutulur?

Data –> document (key value formatı ile) --> collection (-> subcollection)

Firestore üzerinde veritabanı oluşturma

Firestore Kurulum

Verilere erişim, CollectionReference ve DocumentReference

Cloud Firestore üzerindeki verilerime ulaşmam gerekir. Erişebilmek için adrese, referansa ihtiyaç duyulur. Koleksiyon ve dokumanlar farklı sınıflarda tutulur ve kodda bu sınıflar uzerınden erişim saglanır.

```
@override
Widget build(BuildContext context) {
  final CollectionReference kitaplarRef = _database.collection('kitaplar');
  //final DocumentReference hobbitRef = _database.collection('kitaplar').doc('Hobb
  final DocumentReference hobbitRef = kitaplarRef.doc('Hobbit');


  return Scaffold(
    appBar: AppBar(title: Text('Cloud CRUD işlemleri')),
    body: Center(
      child: Column(children: [
        Text(
          'Veriler',
          style: TextStyle(fontSize: 20),
        ),  // Text
        Divider(),
        Text(
          'kitaplarRef: ${kitaplarRef.id}',
          style: TextStyle(fontSize: 20),
```

get() metodu ile manuel 'DocumentSnapshot' çekme

DocumentSnapShot: Bir dökümanın verisinin içerisine konulup taşındığı sınıftır. Firebase'in verileri taşımak için kullandığı bir zarf olarak düşünebilliriz. Bu zarfı açıp verileri okumak için bu sınıfın map tipindeki .data() metodunu kullanırız.

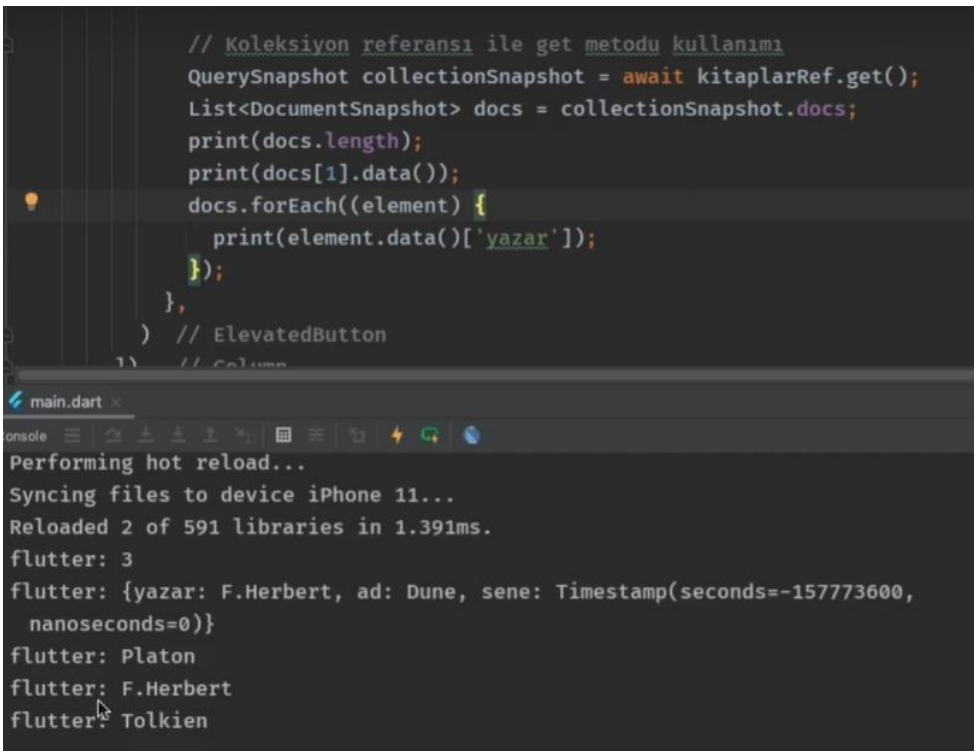
Stringlerle çalışmak tehlikeli olduğu için tutulan verileri bir sınıf içerisinde tutup objeler halinde kullanmamız gerekir.

```
ElevatedButton(
  child: Text('GET DATA'),
  onPressed: () async {
    DocumentSnapshot documentSnapshot = await hobbitRef.get();
    Map<String, dynamic> data = documentSnapshot.data();
    print(data);
```

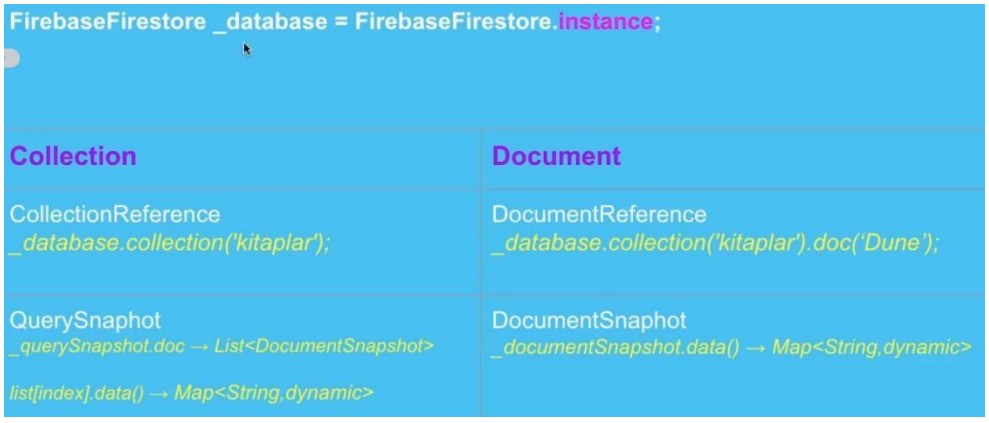
get() metodu ile manuel 'CollectionSnapshot' çekme documentSnapShot *DocumentSnapShot* döndürürken collectionSnapShot *QureySnapShot* döndürür.

.docs ile verilere erişip zarf açılır.

Çoğu zaman veri bu şekilde manuel olarak aktraılmaz. Otomatik olarak verinin akması gerekir.



GENEL BAKIŞ

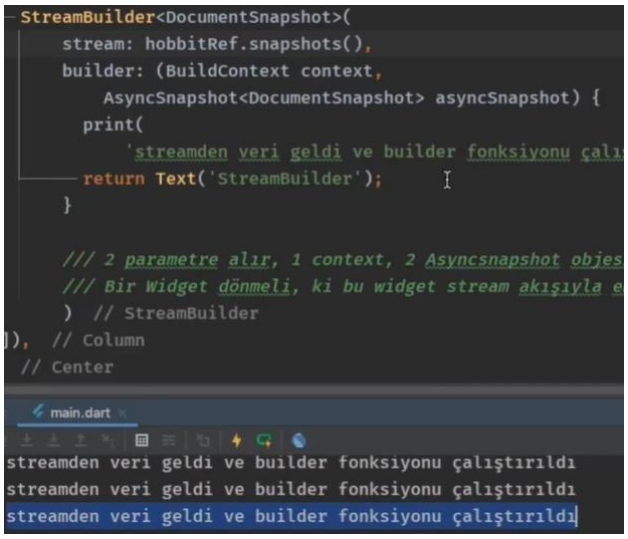


StreemBuilder Widget Kullanımı

Real time online veri ile çalışmayı firebase ile öğrendim. Veriyi get metodu ile manuel olarak çekmeyi ve veri tabanına abone olmayı öğrendim. Böylece veri tabanındaki değişiklik ile uygulama arayüzünün de değişmesini sağlayabiliyorum.

Cloud Storage de bir kitabın yazar ismini değişlik yapınca olan şeyler:

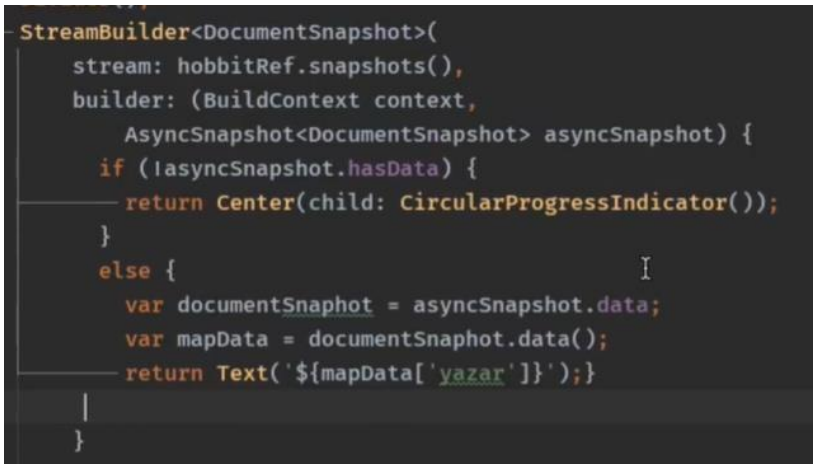
Dinlenilen streamdeki değişikliği firebase tarafından gelen bilgi ile StreamBuilder fark etti ve builder'a verilen metod çalışarak değişiklik arayuze de yansıtıldı



StreamBuilder ile doküman dinleme

AsyncSnapshot içinden DocumentSnapshot, DocumentSnapShot içerisinden de map çıkartılır.

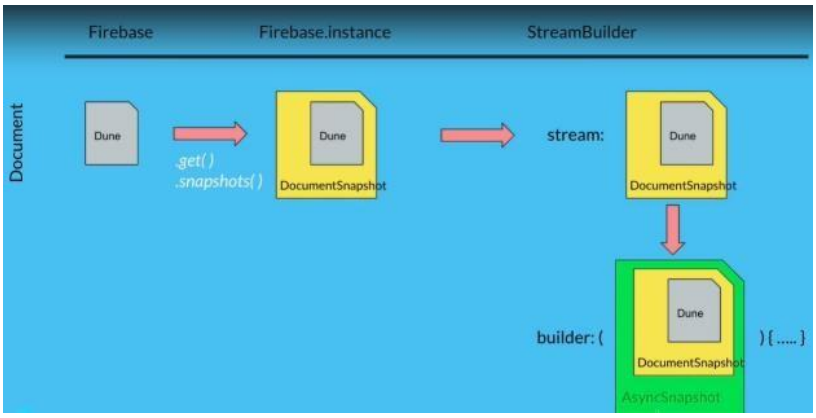
StreamBuilder içinde data henüz gelmedi ise CircProgIndıcator cevırılebilir.



“Dune” adında, Firebase de bulunan bir dökümana ulaşmaya çalışıyoruz.

Firebase.instance objesi ile dökümana gidip veriyi bir DocumentSnapShot içerisinde getirdik ve StreamBuilder’a verdik. StreamBuilder da paketlenmiş veriyi stream den alıp builderdaki fonksiyona AsyncSnapShot paketi içerisine koyup veriyor.

2 sınıf ile paketlendiği için 2 kez çıkarma işlemi gerekiyor veriyi kullanacaksak.



Stream akışının 3 durumu:

1 hata gelmiş olabilir, hata oluşmuş

2 veri bekleniyor

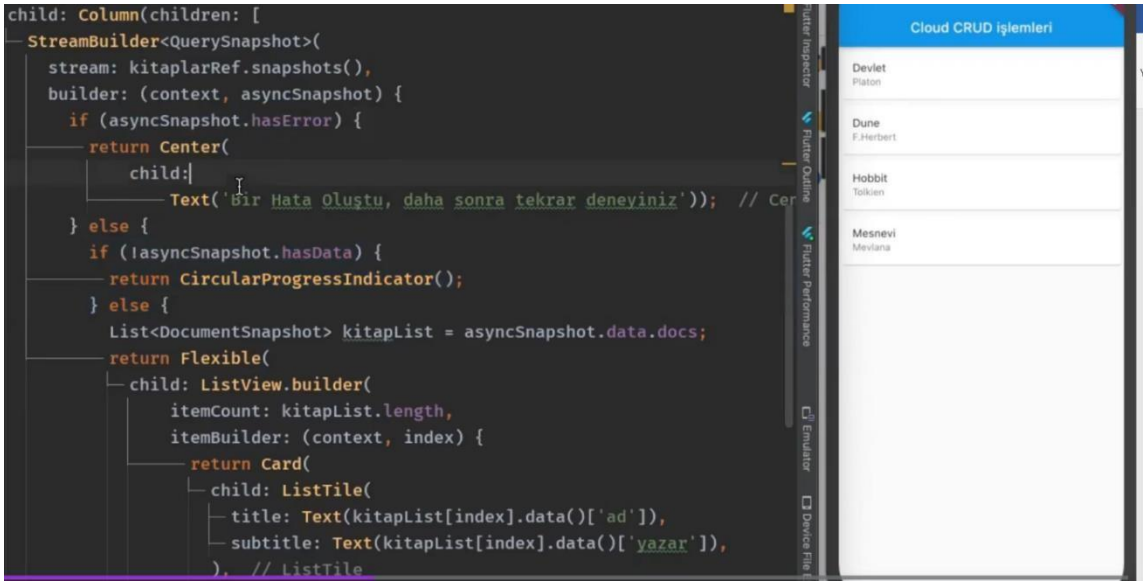
3 veri geldi ve kullanılabilir

Biz if içinde hasData ile datanın gelip gelmediğini kontrol ettik ama hatayı etmedik. Hata durumunu da kontrol etmek için şunu yaptık:

```
StreamBuilder<DocumentSnapshot>(
    stream: hobbitRef.snapshots(),
    builder: (BuildContext context,
        AsyncSnapshot<DocumentSnapshot> asyncSnapshot) {
      if (asyncSnapshot.hasError) {
        return Center(
          child: Text(
              'Bir Hata Oluştu, daha sonra tekrar deneyiniz.'));
      } else {
        if (!asyncSnapshot.hasData) {
          return Center(child: CircularProgressIndicator());
        } else {
          var documentSnaphot = asyncSnapshot.data;
          var mapData = documentSnaphot.data();
          return Text('${mapData['yazar']}');
        }
      }
    }
```

StreamBuilder ile koleksiyon dinleme

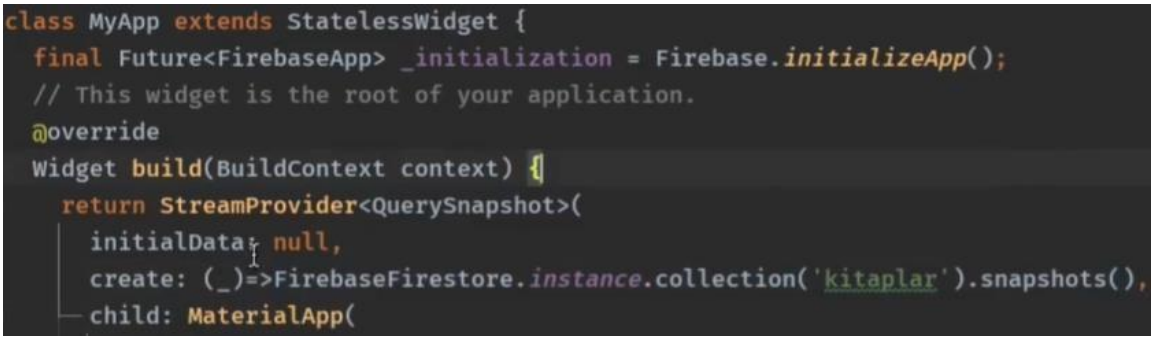
Streamden akan veriyle arayuzu guncellememızı saglar streambuilder



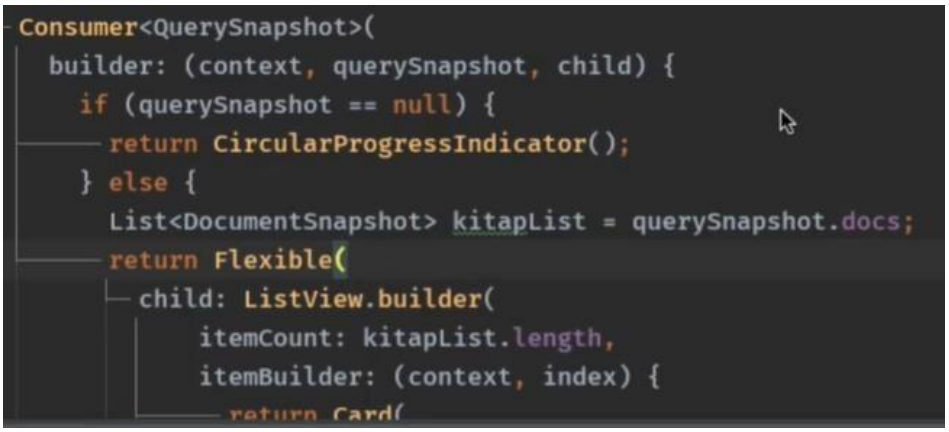
Alternatif metot: StreamProvider Kullanımı

StreamBuilder a alternatif olarak kullanılacak bir yol da provider paketidir. StreamProvider widgetı ile bunu Consumer ile birlikte kullanabiliriz.

MaterialApp i StreamProvider ile sarıp create ile dinleyeceği şeyi yazdık. İnitialData olarak, ilk gelen dataya başta null verdik.



Consumer tanımlayıp builder property'sine fonksiyonumu atadım. Kitap listesi verimi çektim. İf kontrolü yaparak circProgInd döndürdüm.



Doküman Ekleme - add( ) metodu

Bu kez konsol yerine, floatingactionbutton ile flutter üzerinden veri ekledim.

Fire base bize docoumentSnapShot içerisinde bir map yollarken biz firebase e sadece map yolluyoruz. booktoAdd adında bir map tanımladım.

Add metodu ile ekleme yaptım fakat bu kullanımda eklenilen dökümana auto-ıd verildi.

Verilerin tiplerini map içerisinde dikkat ederek yollamakta fayda var çünkü ileride sorgu için sıkıntılar oluşabilir.

```
floatingActionButton: FloatingActionButton(
    onPressed: () async {
      DocumentReference addedBook = await kitaplarRef.add(bookToAdd);
      print(addedBook.id);
      print(addedBook.path);
    },
    child: Icon(Icons.add)),  // FloatingActionButton
; // Scaffold
```

Doküman Ekleme / Güncelleme - set ( ) & update ( )

Bu metot ile istediğimiz ıd ile yollayabilir ve düzenleme yapabiliriz:

```
floatingActionButton: FloatingActionButton(
    onPressed: () async {
      //await kitaplarRef.doc('Denemeler').set(bookToAdd);
      await kitaplarRef.doc(bookToAdd['ad']).set(bookToAdd);

    },
```

```
floatingActionButton: FloatingActionButton(
    onPressed: () async {
      //await kitaplarRef.doc('Denemeler').set(bookToAdd);
      await kitaplarRef.doc(bookToAdd['ad']).update({'sene': 2000});
    },
```

Önceki satırlarda kitaplarRef adlı hazır koleksiyona eriştik. Şimdi ise kod ile yeni bir collection oluşturup içerisine veri attım. Fakat tabii bu kullanım yanlış. Kullanıcıdan alınmalı bu veriler. Firebase metotlarını uygulamak için böyle yaptım:

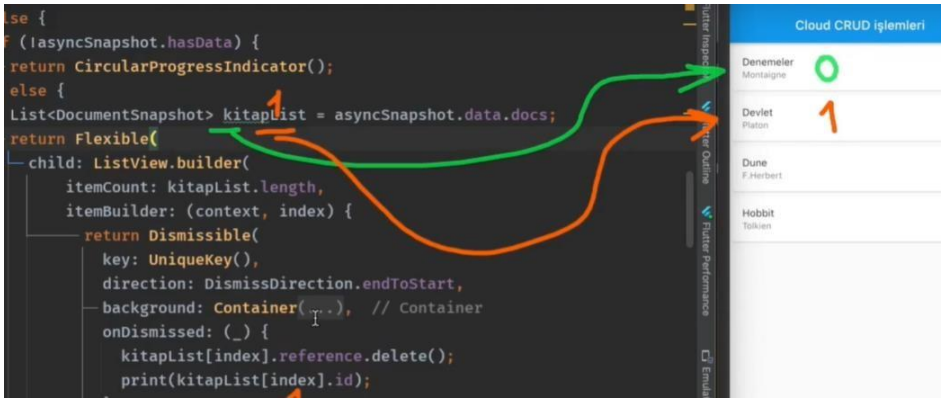
```
floatingActionButton: FloatingActionButton(
    onPressed: () async {
      //await kitaplarRef.doc('Denemeler').set(bookToAdd);
      await _database
          .collection('kayipKitaplar')
          .doc('Harry Potter')
          .set({'ad': 'Harry Potter', 'yazar': 'Rowling', 'sene': 2000});
    },
```

Bulamadığı için kendisi oluşturdu fakat ekranda bir değişiklik olmadı çünkü StreamBuilder ımız bu dizini dinlemiyor, kitapları dinliyor.

Doküman Silme - delete( )

Dissmisible widget kullandım, key ve direction property'sini verdim. onDissmissed property'sine silme fonksiyonunu tanımladım. confirmDismiss ile uyarı ekranı verilebilir.

Tümü yerine sadece bir alanı silmek için onDismissed kısmına delete yerine update({ 'sene': FieldValue.delete()})kullanılabilir.



MVVM için Refactoring

BooksView arayüz katmanım. O sebeple Flutter servislerinin yer aldığı kodları sildim. FloatActionButton içerisini temizledim.

BooksView e karşılık gelen bir booksViewModel dosyası oluşturdum.

Bu dosya içinde:

1) bookview ın state bilgisini tuttum. Kendini dinleyen widgetların güncellenmesi için Provider kullandım.
2) bookview arayüzünün ihtiyacı olan metot ve hesaplamaları yaptım
3) Gerekli servisler ile iletişim kurdum.

Book Model Sınıfı ekleme

Models dosyası oluşturup içerisine Book classı içeren book_model dosyasını ekledim.

Bu veri firebase den obje değil map olarak geleceği için:

Objeden map oluşturan metot:

```
Map<String, dynamic> toMap() => {
  'id':id,
  'bookName':bookName,
  'authorName':authorName,
  'publishdate':publishDate
};
```

Mapten obje oluşturan yapıcı:

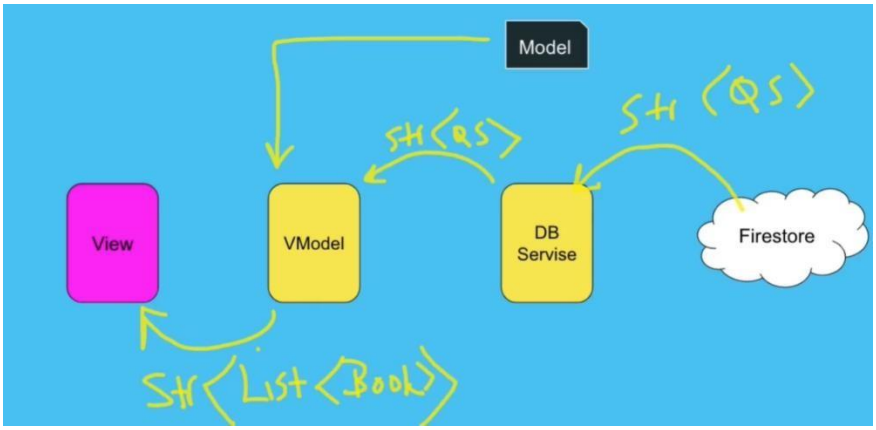
```
factory Book.fromMap(Map map) => Book(
    id:map['id'],
  bookName: map['bookName'],
  authorName: map['authorName'],
  publishDate: map['publishDate']
```

oluşturdum.

Not: “factory” anahtar kelimesi: yapıcıların başına eklenen bir keyword. Yapıcıda return kullanılıyorsa ve duruma göre obje oluşturuluyor ise kullanılır (return … diyip bir obje oluşturuluyor ise).

Database Servisi Oluşturma

View -- Vmodel -- DB Service

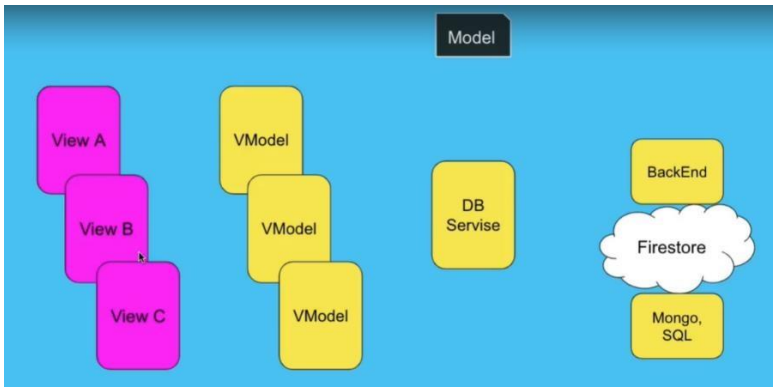


***View*:** Widgetleri tutan, birşeyler gösteren ve hesap yapmayı bilmeyen dumb bir kısım.

Vmodel: View ile Provider ile etileşim içerisinde oldu.

DB Service: Firestore’dan stream gelecek. Vmodel bu stream bilgisini alıp bir metot ile ilgili dönüşümü yaptıkran sonra View’a Stream List olarak gönderecek.

Neden bu DB Service gerekir? Başka veri tabanları kullanılabilir. Bu sebeple yalnızca DB Service de değişiklikler yapılacak veya minimum seviyede Vmodel lerde değişiklik olcak. Yani uygulamadaki güncelleme minimum düzeyde olması için.



Yeni servisler klasörü içinde Databse classını içeren database sınıfnı oluşturdum.

Firestore servisinden kitap verilerini stream olarak alıp bu hizmeti sağlatmak:

```
Stream<QuerySnapshot> getBookListFromApi(String referencePath
    ){
  return _firestore.collection(referencePath).snapshots();
}
```

Firestore üzerindeki bir veriyi silme hizmeti:

Firestore a yeni veri ekleme ve güncelleme hizmeti

VievModel güncelleme

Books_view_model içerisinde:

1) bookview state bilgisini tutmak

2) bookview arayüzünün ihtiyacı olan metot ve hesaplamaları yapmak

3) gerekli servislerle konuşmak

Veriyi gerekli şekilde dönüştürüp arayüze ileten bir metot oluşturdum. DB servis hizmetine erişebilmek için Database sınıfına erişmem gerek. Obje oluşturup kullanarak erişim sağladım.

DB Service deki verileri çekip getiren ve dönüştürüp arayüze ileten getBookList adında bir metot oluşturdum.

Bu metot ile; Stream e direkt ulaşmak yerine bu metot ile Provider vasıtası ile Firebasedeki verileri streamden ListBook olarak erişebiliyor olduk.

```
Stream<List<Book>> getBookList() {
  const String booksRef = 'books';

  /// stream<QuerySnapshot> --> Stream<List<DocumentSnapshot>>
  Stream<List<DocumentSnapshot>> streamListDocument = _database
      .getBookListFromApi(booksRef)
      .map((querySnapshot) => querySnapshot.docs);

  ///Stream<List<DocumentSnapshot>> --> Stream<List<Book>>
  Stream<List<Book>> streamListBook = streamListDocument.map(
      (listOfDocSnap) => listOfDocSnap
          .map((docSnap) => Book.fromMap(docSnap.data()))
          .toList());

  return streamListBook;
```

StreamBuilder güncelleme

Stream propertisini Provider ile güncelledim. Kendiisne gelen veriyi alıp asyncSnapshot ile sarıp Booklardan oluşan bir liste içerir.

Kitap verilerine artık string ile değil yapıcımdaki değişkenler ile eriştim.

asynSnapshot hata döndü, print ederek gördüm hatanın sebebini. getBookList metodumdaki Mapımı toList ile listeye çevirdim.

Delete metotlarını ekleme

Dismiss i çalışabilir hale getirip veritabanından sildirdim:

Databse sınıfında veriyi silme hizmetini sağlayan deleteDocument adında bir metot tanımladım.

Firebasedeki books a ref bilgisi ile git , şu id yi sil diye belirt demem gerekir.

```
Future<void> deleteDocument({String referecePath, String id})async {
  await _firestore.collection(referecePath).doc(id).delete();
}
```

ViewModelde arayüzün kullanacağı deleteBook metodu oluşturdum. Hangi kitabı bilmesini parametre ile belirttim:

```
Future<void> deleteBook(Book book)async{
  await _database.deleteDocument(referecePath:_collectionPath,id:book.id );
}
```

View da onDissmised da Provider ile metoda ulaştım:

```
onDismissed: (_) async {
  await Provider.of<BooksViewModel>(context,
          listen: false)
      .deleteBook(kitapList[index]);
  print(kitapList[index].id);
},
```

Veri Giriş Formu Ekleme

FloatingActButton ile Yeni form sayfası açsın ve veri oradan veri girelim. View klasoru içinde add_book_view ve add_book_view_model sayfası oluştrdum.

1. içinde stateful widget tanımladım.

-TextEditingController kullandıktan sonra sırasıyla dispose kullandım. (hafızada kullandığı yerle iligli bir işlem yapıyoruz)

-validator ile kullanıcının girdiği değer ile ilgili kontroller ve işlemler yaptım. -books_view ana sayfam içinde butonun onpressedine Navigator ile bu sayfaya gitmesini söyledim.

-FormKey tanımladım state tutması için ve key property'si ile Form a bu bilgiyi verdim.

-ElevatedButton onpressed içerisinde key kullanarak validate ile kontrolleri sağladım.

DateTime Picker Kullanımı

Kullanıcıdan tarih ve saat bilgisi almamızı sağlayan bir widgettır.

TextFieldlerin onTap property'sine showDatePicker ile tanımladım ve değişken içinde tuttum.

-Girilen text bilgisini değişkenimin toIso8601String metodu ile textfielda da görünmesini sağladım fakat görüntüsü hoşuma gitmedi.

Servis Oluşturma

Firebase *TimeStamp*, TextFieldım *String*, DateTime Picker ise *DateTime* tipi kullanıyor. Bu veri tiplerine dönüştürme işlemlerini yapmam gerekiyor.

Tarihle ilgili işlemleri yapmak için bir servis oluşturdum.

Calculator adında bı servis dosyası açtım. Calculator adında bir sınıf tanımladım.

Date Formatlama işlemini yapmamı sağlayan intl paketini kullanarak DateTime ı Stringe formatlayıp çeviren bir metot tanımladım.

```
class Calculator {
  /// DateTime zaman biçimini --> Stringe formatlayıp çeviren
  static String dateTimeToString(DateTime dateTime) {...}
}
```

TextFormFieldin onTap ine selectedDatei bu formata çevirerek yazdırmasını sağladım.

```
publishCtr.text =
    Calculator.dateTimeToString(_selectedDate);
```

Add Book Metotları

Database servısı ıcerısıne setBookData adında metot olusturdum.

1) Nereye hangi kolesiyona bu bilgi yazılacak

2) Yazılacak veriyi verip tipinin map objesi olması

```
/// firestore'a yeni veri ekleme ve güncelleme hizmeti
Future<void> setBookData(
    {String collectionPath, Map<String, dynamic> bookAsMap})async{
  await _firestore
      .collection(collectionPath)
      .doc(Book.fromMap(bookAsMap).id)
      .set(bookAsMap);
}
```

Add book view Model ı oluşturmustum ıcıne metot tanımladım.

Bu metodu addbookvıew cagıracak. Kullanıcıdan 3 verı alıyor. Arguman olarak verdım bu verıleri.

1) Formalanındakı verıler ıle book objesı olsuturacak.

-id olarak oluştuurlma zamanını verdim.

-publishDate property'si için dönüşüm gerekti. Calculator içinde bu donusumu yapıp burada kulandım.

```
Future<void> addNewBook(
    {String bookName, String authorName, DateTime publishDate}) async {
  /// Form alanındaki verileri ile önce bir book objesi oluşturulması
      Book newBook = Book(
          id: DateTime.now().toIso8601String(),
          bookName: bookName,
          authorName: authorName,
          publishDate: Calculator.datetimeToTimestamp(publishDate)
      );  // Book
```

2)Kitap bilgisini datbase servisi üzerinden Firestore a yazacak

Add Book View Güncelleme

1) Kullanıcı bilgileri ile addnewbook metodu çağrılacak

-addneewbook viewmodel içerisinde. Buradaki bilgi view a provider ile aktarılcak. Scaffoldu provider ile sarıp create ve builder ekleyerek oluşturdum.

-Provider üzerinden addnewbook a Provider.of yerine read() adında farklı bir format ile ulaştım.

-Zamanı tutan selectedDate metodunu daha genel bir yere çektim onPressed de ulaşabilmek için

2) Navigator.pop edilecek

Update Book Sayfaları

Database deki verileri değiştirebilecek bir yapımız olmalı.

Listtile traililng widgetina ıcon button ile yeni sayfa acılıp duzenleme yapılabilsin.

Views klasorune update_view ve update_view_model olarak iki dosya oluşturdum. Addbok un kopyalraını yapıstırıp duzeltmeleri yaptım.

Listile ın trailing onpresedıne navigator ile yeni sayfamın yoneltmesını yaptım. Acılan sayfada kitap

bilgisi gelmesini istiyorum.

Updatebookview içinde book objesi oluşturdum ve Navıgator push ıcında book verısını gırdım. TextAlanlarında gozukmesı ıcın bookCtr.text e bookname i, author name ve date verisini atadım.

Not: statein stateful içindeki veriye ulaşabilmesi için “widget” yazmamız gerekir.

Database sınıfımda setBookData kullanılacak. Updatebookviewmodel de update book adında metot olusturdum. Book objesını de ıstemesını sagladım argumanda. İd verisini içeriyor cunku. book.id yazabilmek için verdim. selectedDate in null kontrolünü dönüştürme ile sağladım.

Özet: views klasorumuz sureklı aynı yapılardan olusan vıew ve vıewmodel çiftlerinden meydana geldı!

flutter_slidable paketi kullanımı

Dissmisible widget yerine Cardları Slidable widgetti ile sarıp property’lerini kullandım.

Filtreleme TextField Alanı

Not: ListView ile çalışırken Flexible ve Expanded lar hayat kurtarıcıdır.

ListView Filtreleme Mantığı ve Uygulama

ListView neden içindeki eleman sayısını arttırıp azaltarak güncellenecek?

1) Streamden gelen veriden, Clouddan yani online bir değişiklik yapıyor olabilirim.

2) Arama yapma text field ına bir metin girilsin ve bu veri okunup bir metot çalışssın. Local ve offline olarak kullanıcının gördüğü şeyi değiştirmek istiyorum. Basit bir set state ile bu mümkün fakat bütun scaffoldu tetiklememeli. Yalnızca gereken kısmı ilgilendiren bi stateful widget kullanabiliriz.

-Books_view içindeki column ı olduğu gibi BuildListView adı ile stateful widget olarak extract ettim.

-Kullanıcının filtreleme işlemi yapıp yapmadığını anlık olarak tutabilen isFiltering adında bir değişken tanımladım.

-Filtreleme işlemi yapıldığında yalnızca filtrelenmiş verileri tutan filteredList adında geçici liste tanımladım.

-TextFieldın onChanged metodu ile, kullanıcı buraya tıkladı ise:

+ isFiltering true olacak.

+filteredList değişkenine sorgulama ile ekleme yaptırdım. SetState içerisine aldım -ListView.builderın

itemCount u kontrol yapıları ile isFiltering olarak güncelledim.

-büyük küçük harf uyumunu toLowerCase ile kaldırdım.

-klavye text field da olduğu için klavye sürekli açık kaldı. TextField onChange de else kısmı yani boş ise kısmına unFocus metodu içeren ufak bi kod satırı ekledim.

Ödünç Kayıtlar Sayfası Ekleme

Slidable widgetıma bir tane daha IconSlideAction ekledim ve bu BorrowListView sayfasını tetiklettim.

Ödünç Kayıt Modeli

Her bir ödünç kayıtın sınıfı olmalı. İsim soyisim tarih iade ve fotoğraf tutulacak.

Firebase konsoldan add field ile bir map yapısı oluşturulabilir.

Ödünç Kayıt Dart Sınıfı

Her bir kayıt bilgisini tutacak sınıfı oluşturdum.

Models içerisine borrow_info_model adında dart dosyası oluşturdum ve BorrowInfo adlı bir class tanımladım. toMap ve fromMap dönüştürücüleri tanımladım.

Dönüştürme işlemleri normalde manuel olarak değil, ***json_serializable*** paketi ile gerçekleştirilebiliyor. Bu paket ayrıca get ve set metotlarını doğru ve hızlı şekilde oluşturabiliyor.

Book Model güncelleme

Book objeleri içerisinde bir liste ve liste icerisinde borrowInfo objeleri olacak.

Veri tabanından stream ile bir veri geldiğinde book objeleri oluşturup bu objeleri kullanıyorduk. Arayüz bu book objelerini kullanıyordu. Yapmak istediğim şey: veri geldiğinde içinde bir liste olarak borrowInfoları da bulundurması.

Book sınıfı içinde id bookname gıbı degıskenler vardı. Artık bir liste ve liste içerisinde de book objeleri olacak. Bir sınıfın içinde başka sınıfları içeren bir yapı var***. Nested.***

Book sınıfına liste inst. var. ekledim.

Book model içerisindeki dönüştürücülere gerekli işlemleri yaptım.

```dart
/// objeden map olusturcan

Map<String, dynamic> toMap() {

  ///List<BookInfo> ---> List<Map>

  List<Map<String,dynamic>> borrows = this.borrows.map((borrowInfo) => borrowInfo.toMap()).toList();

  return {
    'id': id,
    'bookName': bookName,
    'authorName': authorName,
    'publishDate': publishDate,
    'borrows': borrows,
```

Dart keywords: *as* keywordu kullandım. Objenin tipinden emin olunduğu zaman kullanılır.

```dart
factory Book.fromMap(Map map) {

  var borrowListAsMap =map['borrows'] as List;
  List<BorrowInfo> borrows= borrowListAsMap.map((borrowAsMap) => BorrowInfo.fromMap(borrowAsMap)).toList();

  return Book(
      id: map['id'],
      bookName: map['bookName'],
      authorName: map['authorName'],
    publishDate: map['publishDate'],
    borrows: borrows,
  );
```

İç içe sınıflar var bu yapıda. Bu şekilde oluşturuluyor. Nested yapı.

addBook metodu güncelleme

Veri geldiğimde book objeleri oluşturulabiliyor otomatik.

Book obje içinde borrowınfo kayıtları geldi mi diye kontrol ettim.

Print (fullList.first.borrows.first.name)

- ⇨ Firestore verisini alıp donusturdum bu veriler de arayuze stream ile aktı.

Veri tabanına yeni kitap oluşturmak için ise addBook metoduna geldim ve boş bir borrows listesi koyup tanımladım.

Ödünç Kayıtları listeleme

Borrow_list_view içersinde build metodu içerisinde borrowList değişkeni tnaımlayıp bu verileri burada tuttum ve Scaffold bodsyine ListView.seperated koydum.

Kayıt Ekleme Form Sayfası

Firesbase Storage Kurulum Firebase_storage paketini

kullandım.

Firebase konsolunda storage kısmı var.

İf true değişikliğini yaptım kurrallar içersindeki.

Photos adlı klasor olusturup pc den bu dosyaya resım attım.

Uygulamada alınan fotoğrafları bu storage a atmak istiyorum.

Storage dosya erişimi

Using Cloud Storage dökümanı ile nasıl erişim sağlayacağımı öğrendim.

Konsolda storage dekı img dosyasına kodumda ulastım.

Dosyanın ***referencea*** ulasıp ***getDownloadUrl*** metodunu kullandım.

Her yenı odunc kayıtta bır foto pc hafızasında tutulup storage e yukelenecek. Storage bu url bılgısını bıze verıcek bız de BorrowInfo da photo url alanına yazcaz. Tuşa basınca olusan borrow objeleri oluşuyor bunların ıcınde photourl bılgısı var. **image_picker paketi**

cihaz kamerasını açmak veya galeriden resim seçmek

bu fotograf bılgısı File _image değişkenim içinde tutuluyor. Uygulama reset olunca storage de olmadıgı ıcın fotograf gıder.

Storage dosya yükleme getImage metodumun altına uploadImagetoStore adlı bir metot

tanımladım.

Storage üzerindeki dosya adını oluştur. Unıque olması gerektiği için fotoğrafın çekildiği tarihi ad olarak atadım.

Dosyayı gönder (putFile metodunu ve getDownloadURL metodunu kullandım)

Imaj boyut ve kalite ayarı

Resimler çok yer kapladıgı için resimleri sıkıştırmak gerekir.
Flutter_image_compress paketi bu işlem için kullanılabilir. Fakat benım kullandığım ImagePıcker paketi de bize resim üzerinde değişiklik yapabileceğimiz şeyler sunuyor.

photoUrl kayıtlarını alma photoUrl değişkeni tanımladım.

uploadImagetoStorage metoduna bu degıskenı return ettırdım.

Bu değişkene uploadImageToStorage metodunu async olarak atadım fakat kullanıcıdan bir resim geldiğine emin olmak için veya hiç resim yüklemediği durumlar için gerekli kontrolleri sağladıktan sonra.

Odunc kayıt butonuna basınca BorrowInfo objesı olusturduğuumuz kısma photouRlı tanımladım ve kullanmıs oldum.

SetState async olamaz. Await işlemler burada gerceklesemez

Yeni kayıttan vazgeçme senaryosu

Resım yukleyıp bilgileri doldurduktan sonra odunc kayıt ekle tusuna basmaktan vazgeçebiliriz. Bu gibi durumlarda fotoğraf boş yere firebase storage'a yüklenmış olur.

Kayıtın iptal edildiğini anlamamız gerekir. ModelBottomSheet in kapatıldığını anlamalıyız.

InkWell içinde newBorrowInfo objesi oluşturularak modalbottomsheet return ediliyordu. Bu Dönen değer eğer açılmazsa veya kapatılmazsa null döner. Fakat bu konrtrol ile photoUrl farklı yerlerde. Uygulama mimarisi dolayısı ıle bu ıslemı saglayamıyorum. Alternatıf olarak kullanıcının vazagecme senaryolarının hepsını engelledım. Vazgecme senaryosu ıcın ıptal butonu ekledım ve bu ıslemı kontrol edebilir hale geldim.

İşlem iptali fonksiyonu & WillPopScope Widgetı

Model bottom shıt arkaplana, gerı tusuna veya modelbottomsheet kaydırılarak default kapanma yontemlerını ıptal ettım. enableDrage ve isDissmisble property'lerini false yaptım.

Geri tusunu deaktıf etmeyı de BorrowForm u WillPopScope Widget ile sardım. Gerı tusunu overrıde ettık. İptal butonu ekledim ve navigator.pop kullandım

Store'dan dosya silmek

Store daki bir dosyaya erişip silme işlemi

View_model içerisinde oluşturdum metodumu bu sefer. Deletephoto adında. url den ref

bilgisine ulaştım.

Bu metoda arayuzde buton ıcerısınde ulasıp gerekli kontrol ile ulaştım ve kulandım.

Yeni sayfa açılırken context ile material app'e bağlanıyor.

Not: Null Coalescing Operator "??"

Not: *Provider'ın state management paketi olarak dezavantajı:* Her yeni widget dalı için ayrı context oluşturulur ve bu contextler yalnızca kendi içerisini ilgilendirir. Diğer contextlere erişim sağlayamaz.

Provider context vasıtası ile iletişim kurabilen bir paket.

Oysa GetX veya riverPod context ten, provider dan bağımsız çalışır. *image_crop paketi*

kullanılmadı!!

Abstraction ile servislerimi defalarca başka projelerimde kullanabilirim. Atomic design.