İÇİNDEKİLER

Bilgi Testi Uygulaması

Wrap Widget

Row ve Colum'ların çocukları ekrana sığmazsa alta veya yana kaydırarak ekranda görünmeye devam etmesini sağlar. Bunun için yatay veya dikey mi kaymasını istediğinizi söylemelisiniz. Kullandığım bazı property'leri:

Direction: Axis.vertical - *yönü dikey yapar (default: horizontal)*

Spacing: – widgetler arası boşluk

runSpacing: – dikeyde boşluk

alignment: Wrap.Alignment.end/center - dizim sırasını değiştirir

AlertDialog Widget

ShowDialog() fonksiyonu çağrıldıktan sonra kullanılan alertdialog widgeti.

Class ve Obje Kullanımı

Soyutlama/Abstraction:

Class'lar soyut bir tanımlamadır. Classların içinde somut nesneler bulunur.

Bir sayfada bir kodun içinde birden fazla iş yapmak yerine, sayflarımızı farklı temel görevler için ayrı parçalara ayırıp birbirinden soyutlamalıyız.

Örneğin kod içindeki sabit tanımlamaları veya veri içeren bir liste yapısını class içerisinde ayrı bir dosyada tutmak gibi.

Paketleme/Encapsulation:

Bir nesnenin özelliklerinin dışarıdan kullanılmasını sınırlamak için kullanılır. Verileri başka bir kısımda ayırmamıza, yani soyutlama yapmamıza rağmen ana paketin içinde o veriye dışardan erişilip herhangi bir değişiklik yapılabiliyor ise programımız kullanışlı olmaz.

"_" ile değişkeni private yapabiliriz ve dışardan direkt erişim engellenir. Tanımlanan get fonksiyonu ile dışardan çağrılabilir. Get metotları return edilmeden yani işlemler yapılmadan önce kontrol yapabilmemize imkanı sağlar.

Test verisi ile ilgili tüm metot ve tanımlamaları bu classın içerisine paketledim ve dışarıdan erişilmesini engellemek için private yaptım.

Miras Alma / Inheritance:

Bir nesnenin özelliklerinin farklı nesneler tarafından da kullanılabilmesine olanak sağlayan OOP özelliğidir. Yazılan bir sınıf bir başka sınıf tarafından miras alınabilir. Bu işlem yapıldığı zaman temel alınan sınıfın tüm özellikleri yeni sınıfa aktarılır. Sınıfların parent child ilişkisi içerisinde olmalarıdır. (extends)

Override: bir sınıfa ait bir metodun, o sınıftan türetilmiş bir sınıf içerisinde aynı isimli bir metod tanımlanarak, bu metodun temel sınıftaki metodun yerine geçirmeye denir. Bu işlem,bir metodun aynı sınıftan türetilmiş farklı sınıflarda farklı işlere yaramasını sağlar.

Not: *"super" keywordü ata sınıfın özelliğini alıp kullanması için tanımlanır.*

Çok Biçimlilik / Polymorphism:

Metotların objeye göre farklı çıktılar üretmesi veya farklı işler yapmasıdır. Yani alışageldiğimiz gibi metotlara sabit görevler vermek yerine onlara çok biçimli (polimorf) davranacak şekilde bir esneklik vermektir.

Children[] ile widget tipinde bir çok child eklenebilir. Örneğin text, button .vs..

Not: Bir değişkene on pressed içerisinde parantez verildiği zaman, onpressed bir atama yapmak, bir değer döndürmek isteyeceğinden hata verir. Anonim bir fonksiton içerisnde parantez verilip parametre tanımlanan bir fonksiyon tanımlanırsa sorun olmayacaktır.

Beklenen Yaşam Süresi Uygulaması

Kullanıcıdan farklı widget ve şekiller kullanarak bilgi almaya çalışan bir uygulama.

Tema Kullanımına Giriş

Birden fazla sayfalı uygulamalar için her sayfanın teması benzer olacağından, bir yerde temel bir tema tanımlayıp diğer yerlerde de bu temayı kullanırız.

-Theme(): İstediğimiz widgetı bu widget ile sararak spesifik temalandırma yapabiliriz.

-ThemeData():

-Theme.of.(context).copyWith(accentColor:Colors.red):Widget ağacında kendisine tanımla- nan en yakın temaya gidip ulaşır ve temayı kopyalayıp ve getirir ama butonun rengi kırmızı olarak değişir.

Not: Dart da çoğu ifadeyi .toString() ile string veri yapısına dönüştürmek mümkün.

BoxDecoration Widget

Decoration property'e verilen widget tanımıdır. Borderradius gbi eşitli düzenlemeler içerir.

Expanded ve Flexible Widget

-flex

Widget Oluşturma (Composition) ve Özelleştirme

Widget extractiondan yararlanarak, bir constrtuctor ile renk ve child gibi property'ler ekleyerek bir widget oluşturabiliriz.

GestureDetector Widget

Çevrelediği widget'ın kullanıcının yaptığı kaydırma, basılı tutma gibi el işlemlerini algılamasını sağlar. Aynı zamanda o widget'ı tıklanabilir hale getirir.

-onTap: (){}

Slider Widget

Slider.adaptive() ile kullanılan platforma özgü slider tipi oluşturulur.

-min ve max:

-onChanged: kendisine atanan fonksiyon ile kullanıcının değişikliğini uygular.

-value: bir değişken tanımlanarak dinamik bir değer olması gerekir.

-divisions:

-labels:

Not: "Interpolation" kavramı temel olarak tanımlanmış iki değişkenin +(plus) işareti ile değil de parantezler içerisinde "$" işareti kullanımı ile birleştirilmesi işlemidir.

'${degiskenismim .round()}'

Not: String içinde $ gibi özel ifadeleri kullanmak için "\" kullanılır. "String \$"

OutlineButton & RotatedBox Widget

Widget & Metot Extraction

Refactor --> "Extract Flutter Widget" özelliği ile bir widgetı koparıp ayrı bir yerde yeni bir isimlendirme ile tanımlayıp kullanabiliriz. Bu tanımlamaya bir constrtuctor ile renk ve child gibi property'ler ekleyebiliriz ve böylelikle kendi özelliklerimizi oluşturduğumuz bir widget'ımız olur.

Refactor --> "Extract Metot" özelliği ile bir widgetı koparıp ayrı bir yerde yeni bir isimlendirme ile fonksiyon olarak tanımlayıp kullanabiliriz Widget extract edemediğmiz veya ihtiyacımız olduğu durumlarda metot extract edebiliriz.

Not: *StateManagment:* Hangi değişkeni nerde yöneteceğimizi, widget tree'yi nasıl oluşturacağımızı, değişkenleri birbirinden nasıl ayıracağımızı, hangi değişkene nereden ne zaman ulaşacağımızı ve değiştirme yetkisi vereceğimizi belirlememizi sağlayan, *arabirim denetim durumunun yönetimi'*dir.

Not: setState, State class'ının yanı örneğin stateless veya statefull sınıfının bir metodudur. Bir widget içinde bu widget'e ait setState metotunu kullanan başka bir widgetı bu sebeple extract edemeyiz.

Navigation: Sayfalar Arası Geçiş

Sayfalar flutter'da üst üste katman olarak yapışır.

Navigator sınıfı ekrana yeni bir katman yani route oluşturur veya o katmanı kaldırır.

-push: yapıştır

-pushReplacement: mevcut sayfayı kaldırarak yeni sayfa yapıştırır.

-pushAndRemoveUntil: yeni sayfayı getirirken önceki sayfaları kaldırır.

-pop: kaldır

-popUntil(): ilk katmana kadar tüm katmanları kaldırır.

-*context:* Flutter ağaç yapısının farkındadır fakat herhangi bir widgeta bunu bildirmek gerekir.

-Route: MaterialPageRoute yapısından oluşturulur. Bu yapının *builder* property'sini kullanırız.

Not: *Syntactic Sugar* Örneği: (context) {return RoutePink();} / (context) => RoutePink()

Navigation: Basit Veri Transferi

Bir sayfadan alınan veriler başka bir sayfada kullanılma ihtiyacı duyulabilir.

Not: *Flutter Null Safety içerir. Bu özellik devre dışı bırakılabilir.*

Navigation: Named Routes

Basit bir uygulamada sayfalar arası geçişlerde butonlara route özelliği verilebilir ancak büyük uygulamalarda bu yapı kod karmaşıklığına sebep olur ve kontrolü zorlaşır. Bu sebeple Navigator'ın *named routes* özelliğini kullanırız. Kullanıcının gidebileceği her sayfayı tanımlayıcı bir ad ile isimlendirip daha derli ve toplu bir yapı içerisinde tutabiliriz.

Not: *Map:* veri saklama yapısıdır. Key ve value değerlerini birbiri ile eşleştirir.

Not: *Object*: bütun veri tiplerini Dart obje olarak tutar. Tüm veri tiplerinin atasıdır.

Not: *Static:* herhangi bir obje üretilmese dahi programın her yerinden ulaşılabilmeyi sağlar.

Not: *Final:* daha sonra değiştirilemez bir değişken keyword'üdür.

Not: *const* compile time'da yani programımız henüz çalışmaya başlamadığı anda kullanılacak sabitler için, *final* ise run time'da çalışacak sabitler için tanımlanır.

Sayfaları map yapısı içerisinde tanımlayıp tutabiliriz:

Her widget içine static route name tanımlandıktan sonra:

-routes: {

HomePage.routeName: (context)=>HomePage() ,

… }

-pushNamed:

Not: -initialRoute: Uygulama ilk açıldığında gidilmesi gereken route'u ifade eder.

Navigation: ModalRoute.of() & onGenerateRoute

Bir sınıfa ait bir nesnenin bilgilerini içeren veri başka bir sayfaya pushNamed içerisinde gönderilir. Navigatordan gelen veriyi ilgili sayfa içerisinde *ModalRouteof()* ile çağırırız.

Materialapp sınfıına ait *onGenerateRoute* property'si routing işlemini yönetmemizi sağlar.

Navigator tarafından yollanan *settings* parametresini kullanır. Bu fonksiyon içerisinde switch case kullanılması uygundur.

Navigaton: Özet

Basit Navigasyon	Named Routing	Named Routing
push()	pushNamed()	pushNamed()
home:	initialRoute: routes: { *map* }	home: onGenerateRoute: ( ) { }
Constructor ile direkt veri transferi	Arguments: & ModalRoute.of()	Arguments: & Constructor

Get It Done, To-Do-List Uygulaması

Splash Screen ile açılıyor.

Settings kısmından tema değişikliği yapılabiliyor. Bu değişiklik provider paketi ile bütün widget ağacına yayınlanıyor ve state management bu şekilde yapılyor. State management provider paketi ile kullanıcı tuşa basıp yeni görev eklediği zaman, gorevlerin yapılıp yapılmadığını tikleyıp işaretleyebiliyor ve bu değişikliklier provider paketi ile sağlanıyor. Kullanıcı, kaydırma işlemi ile görev silebiliyor ve state yönetimi ile uygulamanın ilgili widget'ları kendini yeniden inşa edebiliyor. Tema ve görev kartı bilgilerini cihazın shared prefernces'ına kaydedebiliyoruz ve uygulama kapatıp açıldığı zaman kullanıcının yapmış olduğu işlemler kaybedilmiyor. Veri tabanı yönetimi en basit hali ıle shared preferences ile sağlanmıştır.

State Management Nedir? Neden Gerekir?

State, cihazın hafızasında bulunan her şeydir. Kullanıcının o anda ekranda ne görüp duyduğu ve bunları oluşturan tüm değişken ve property'lerin o anki durumlarıdır.

"State", bizim property'lere verdiğimiz değerler ve olusturduğumuz değişkenler.

"f" flutter'ın bizim tarafımızdan oluşturulan metotları.

"f" bizim ona verdiğimiz state'leri alan ve uygulamanın o anki statei olarak kullanan ve ekranı, arayüzü bize döndüren . Her state degıstıgında arayuz tekrar yenilenebilir

r.

State Management, *getx, provider, block* ve *redux* gibi tool'lar kullanılarak bizim tarafımızdan yapılır.

Flutter State'i ikiye ayırır:

1) Local State (Ephemeral State): Büyük bir widget tree içerisindeki özel bir local widget'taki herhangi bir özelliği değiştirme örneği verilebilir. Bu durumda tüm widgetlar yerine sadece değişiklik yapılan widget tekrar ekrana çizilir. setState veya Stateful Widget ile yapılabilir. Bu duruma local state yönetimi denir.

2) App State: Tum uygulamada etki yapabilen bir değişkeni değiştirerek tüm widget tree'nin etkilenmesine neden olma durumudur. Örneğin tema değiştirilmesi. Bunlar için state yönetimi gerekir.

Callback Kullanımı ile State Yönetimi

Neden ilave araca ve pakete ihtiyacımız var? State yönetimi gereklidir.

En tepede bulunan bir veriyi altlarda bir widget'a ulaştırmak için her widget içinde o veriyi constr ile tanımlayıp aktarmak gerekıyor.Veri aşağıda bulunsaydı, bazı widget'lara ulaşamazdı. Context, widget'ların kafasını kaldırıp nerelere ulaşılabilir olduğunu öğrenmesine yarar. En nihayetinde veriyi aktarırken hiç gerek olmayan widgetlara bile tek tek bu veriyi aktarmamız gerekicekti ve tüm widget ağacını tekrar çizmeye kalkacaktık. Callback'ler kullanarak bu yönetimi kullanmak verimli olmaz. State'i nasıl yöneteceğimize karar verdikten sonra bu state paketleri ile yönetimizi yaparız. Biz bu uygulamada Provider paketi kullanacağız.

Lifting State Up: State'ı yukarı taşımalıyız ki ağaçta aşağılardaki widget'lar da ulaşabiliyor olsun.

Callback() kapıdaki zil.

Zilcal() evin başka bir yerindeki bir fonksıyon.

Bu fonksiyona ulaşabilmek için kapıdaki zilin kullanılması gerekir. Dışarıdan erişim sağlandı.

Not: Callback içerisinde parantez kullanmayız çünkü yalnızca adresini ifade etmek isteriz.

Provider Paketi

State verısını tum wıdget dısında bıyere koyup wıdgetlerle baglantısını kesmemız gerekır. Arayuz ile veri içeren kısımların bırıbırıunden ayrı olması mımarı ıcın onemlı bı hususTur. Veriyi tüm widgetlerdan bagımsız bır sınıf ıcerısınde tutarak bır sınıf olusturmalıyız. Bu verı tum wıdget agacına veya ıhtıyac olan kısma yayınlanabılmesı gerekır. Bunu saglayacak sey provıder olacak. Sağlayıcı, widget agacına baglanabılen ve yayım yapabılen bır objedır.

Not: extends yerine with: tamamen mirascısı olmasın ama metotlarını ve ozlelıklerını kullanbılsın demektir. Böylece başka sınıfların da metotlarına ulaşabilir bir hale gelir. Örneğin, (Class ClassIsmım with IlkSınıf, IkıncıSınıf, UcuncuSInıf{}).

Change Notifier sınıfının notifyListeners() adında değişiklikleri bildirme özelliğini kullandık.

Statless widgetleri provider kullanarak değişmesini sağladık. Aynı zamanda yalnızca istediğimiz widgetlar rebuild oldu. Optimizasyon ve hızlı calısan bı uygulama ıcın onemlı bır durumdur.

Veri API den gelebilir ve bu gibi durumda future provider kullanırız.

Provider: Consumer Widget

Widget ağacının sadece ihtiyaç olunan kısmın rebuild olması için Consumer kullanılır.

Consumer nerede ise o kısım kendini build ediyor. Ayrıca tek tek provider of yazmak yerine Consumer widgetinin build parametresine verılen orta değeri kullanarak kodu kısaltmış oluruz.

Temalandırma ve Ana Sayfa

primarySwatch: ThemeData'nın bir property'idir. verilen rengin tonlarını kullanarak bir renk oluşturur.

Text typhography material.io sitesinden faydalanarak subtitle ve headlines ları kullandım.

Column içerisinde Flex özellikli 2 expanded widget ile ekranı paylaştırarak yapıyı oluşturdum..

Görev Kartları Oluşturma

ItemCard adında *Card* widgetı return eden bir widget oluşturdum. Card içerisinde child olarak *ListTile* widgetı kullandım ve bu widget ile text ve checkbox iconu ekledim.

ListView.builder kullanarak home sayfasında ItemCard() widgetımı eklemiş oldum.

Görevler Veritabanı Oluşturma

Provider Kullanımı

State bilgisini tutan ıtemdata sınıfına changenotfier özelliğini kazandırıdım.

Myapp ana widgetımı ChangeNotifierProvider ile sarmaladım ve ItemData verilerini yayınlattım.

Veriyi ekrana çizen listview builder bilgilerini provider ile dinamik bir forma ulaştırdım.

Item sayısını veren ilk expanded kısmı için text verisini ınterpolation ve provider ile dinamik hale getirdim.

Görev Tamamlama Ayarları

Uygulama stateımde, veritabanında bir şey değiştiği zaman ekran otomatik yenilenmeli.

toggleStatus ile durum değişikliğini işleyip checkboxların tıklanma animasyonunu sağladık.

Card widgetına elevation: isDone?1 : 5 ile check box tıklanınca yapışması animasyonunu kattık.

Yeni Görev Ekleme

Floating action buttona tıklandığında state verimi tutan listeme eklenen item ı eklemem gerekir.

BottomSheet class ını kullanarak animasyonlu pencere sağlandı. *Persistent:* kapanması için geri tusuna basmanız gerekır, arka ekrana dokununca kapanmaz.

Modal: pencere arka plana dokununca kapanıyor. Bu uygulamada *Model*'den yararlandık.

Kullanıcının TextField a girdiği input görevini text fieldın controller property'sinden yararlanrak tanımladığımız textController değişkeni içerisinde tuttuk.

Bu değişkeni items data kısmına yeni bir veri olarak provider ile (add butonuna basıldığı zaman gerçekleşmesi gerektiği için Flatbutton içerisindeki onpressed fonksiyonunun içerisinde) gönderdik.

Klavye boyutu farklılığından dolayı container a yalnızca alttan klavyenın kapladığı kadar padding verdim. *(bottom: MediaQuery.of(context).viewInsets.bottom)*

Text Field a satır atlayabilmesi için min ve max lines propert'lerini kullandık

Görev Silme

Cardlara *title* ile *key* vererek *Dissmisible* widget kullandım. Kaydırıp yok edilen card nesnesi veri tabanından ve widget ağacından da yok edilmeli. ListView.builder içinde removeAt kullanan deleteItem fonksyonunu provider ile kullandım. Aynı zamanda Dismissible widgetın onDismissed property'sine de bu fonksiyonu tanımladım.

Cardları GestureDetector widget ile sarmalayıp onTap, doubleTap, onVericalDragDown gibi property'lere deleteItem fonksiyonunu tanımlayabilir ve silinme animasyonunu değiştirebilirdim.

Setting Page ve MultiProvider kullanımı

Ayrı bir settings_page oluşturup body sine

SwitchCard adında Card Widgetı kullanan ayrı oluşturduğum bir statefull widget verdim. Child olarak *SwitchListTile* classını kullandım. Local bir stateManagement yönetebilmek için böyle yaptım.

Home page e ekledıgım settıngs butonunun onchanged property'sine Navigator.push metotunu kullanarak, tıklanıldığında SettingsPage sayfasına gitmesini sağladım.

SwitchListTile card ı ayrı bır yerde statefull wıdget olarak tanımlayarak local state Management yapmıs oldum. Butonun ıcında calısan her sey yalnızca o butonun setstate ını etkılıyor.

SwitchListTile üstünde iki ayrı Text tanımlayarak subtitle property'sine ternary operator ile ekledim.

Color_theme_data sayfasında yeni bir class oluşturup (with ChangeNotifier) iki adet tema tanımladım.

selectedThemeData değişkenime default olarak purple ThemeData atadım.

selectedThemeData yı değiştirebilecek bir fonksiyon tanımladım ve içerisindeki private değişkenimin get metodunu yazdım.

Widget agacının en tepesınden asagı MaterialApp ın görebileceği şekilde yayınlamam gerek.

1.ChangeNotifierProvider = listelerle ilgili veriyi alan ItemData.

2 ChangeNotifierProvider = tema rengi ile ilgili veriyi alan ItemData.

Birden fazla veri provide etmek için MultiProvider kullandım.

MaterialApp içerisindeki theme property'sini Provider ile değişip yayınlanmasını sağladım.

SwitchListTile içindeki butonuna, onchanged property'si içerisinde provider ile switchTheme metotunu ekleyerek fonksyonalite kazandırdım.

Final Refactoring

ListView.builderı consumer widget ile sarmalayıp tüm widgetn yanı scaffoldun yerıne sadece consumer kısmının provider ile rebuild olmasını sağladım. Local state yonetımı yaptım.

ListView.builder ı Align ile sarıp shrankwrap ve reverse ile eklenen liste elemanlarının üstten alta doğru eklenmesini sağladım.

İtem Data sınıfımdaki items değişkenimi private yapıp get ile aynı adlı kopyasını oluşturarak ve *UnmodifiableListView* kullanarak dışarıdan erişilemez hale getırdım.

Kodun daha rahat okunabilir olması için, appbar widgetını home_page kısmında metot olarak extract ettım.

Shared Preferences

Kullanıcı uygulamadan çıkıp tekrar gırdıgında lısteleri hazır olarak gelmeliydi.

Verının nasıl bır verı olduguna baglı olarak verıyı tutmak ıcın kullancagınız yontemler degısıklık gosterır. Uygulama içerisinde saklanılacak tema değimi ve login bilgileri gibi verileri tutan özel dosyalar vardır. Bu dosyaya kalıcı olarak kaydedılen verılere shared preferences denır. Krıtık bılgılerın burada saklanılması önerılmez. Bilgiye ulasılma ıslemı işlemciye gore daha yavas olacagından asynchronously yapı gerekır.

Provider ile basitçe bir state yönetimi yaptım.

Not: Parantez içerisindeki argüman gövde içerisinde kullanılmıyor ama yine de parantez içerisinde parametre olarak belirtilmesi gererkiyorsa "_" kullanabiliriz.

Shared Preferences + Provider Paketini Birlikte Kullanma

Provider ile yönetilen bilgimizin cihaz üzerinde kalıcı saklanması gerekiyor. Bu işlem için 2 adet fonksiyona ihtiyaç duyuyorum:

1. Fonksiyon: Cihazadaki SharedPreferences dosyasına git/aç ve yazma işlemi yap.

2.Fonksiyon: Yazılan veriyi oku.

SharedPrefernces objesi tanımladım. Bu objeyı kullanmamızı sağlayan *createSharedPrefObject* adınfa Future döndüren bir asyn fonksiyon oluşturdum. Bu fonksiyon getInstance() statik metotunu barındırır.

Uygulama açıldığında splash screen koymak, veriyi okuyup işleyebilmek için vakit kazandırır.

Not: Uygulama mimarisinde, her iş ve servisin uzmanı sınıflar farklı olmalı ve loosely-coupled, yani birbirlerine en zayıf şekilde bağlı olmalılardır. Arayüz ekranlarının olabildiğince bu tip servis ve verilerden ayırılması gerekir.

Anlık değeri kaydedip tutan *saveThemeToSharedPref* adında bir fonksiyon tanımladım. Buton ile her toggle yapıldığında bu fonksiyon çağrılır.

Uygulama ilk çalıştığında kayıtlı verinin okunması için *loadThemeFromSharedPref* adında veriyi uygulama ilk açıldığında hemen çağırıp getiren bir fonksiyon oluşturdum. Statefull widget olsaydı initState içerisinde çağırabilirdim. Fakat statless widget kullanıldığı için MaterialApp return edilmeden hemen önce Provider ile çağırdım.

Not: Syntactic Sugar Önerisi: "*??*", yazılan değerin null olması halinde devamına yazılan değeri atama işlemi yapar. Yani null koşulunu/kontrolünü gerçekleştirir.

Shared Preferences: main( ) async

Shared Prefernces'dan gidip çekilen veri önem taşıyor ve hiçbir şey oluşmadan önce o bilginin elimizde olmasını istiyorsak, main fonksiyonun içerisinde, runApp çalıştırılmadan önce asyn olarak veri çeken bir fonksiyon tanımlarım. Yani aslında main fonksiyonunu bir asyn fonksiyona çevirmiş olurum. Önce veri gelir, sonra runApp çalışır.

Not: => rotasyonu yalnızca bir satırlık returnler için kullanılan bir kısaltma işlemidir.

Main fonksiyonu içerisinde createSharedPrefObject fonksiyonumu await ile çağırdım. Bu kullanım için fonksiyonumun objesiz de kullanabilir bir yapıda olması gerekir çünkü henüz obje üretilmeden çağırıp kullanmak istiyorum. Objenin üretildiği yere static anahtar kelimesi kullanarak bu işlemi gerçekleştirdim. Bu işlemin gerçekleştirilmesi için öncesinde WidgetsFlutterBinding sınıfının ensureInitialized() metotunu kullanmamız gerekir.

Not: "static" anahtar kelimesi, bir sınıfın sahip olduğu verilerin farklı nesneleri arasındaki değerleri kalıcı hale getirmesine olanak tanır. Elimizdeki sınıfın herhangi bir obje oluşturulmadan kullanılabilecek, hemen ulaşılabilecek bir instances field tanımlayabilmemizi sağlar.

SwitchTileList Hata Düzeltme

Uygulamada settings kısmında ListTile ile Local StateManagement yapıp setState kullandık. Kullanıcının tema değiştirme butonuna tıklaması, yalnızca metni ve butonun görüntüsünü etkilemiyor. Tüm widget ağacını etkiliyor. Settings kısmını izole edemedik. SwitchCard Widgetımızın stateful olmasının bir anlamı kalmadı. Stateless yapıp set state i sildik.

Tema Bilgisini Kaydetme

Uygulamanın tema rengi bilgisini ve yapılacaklar listesinin elemanlarını cihazın shared prefernces'ına kaydetmek ve uygulama yeniden açıldığında bu bilginin tutuluyor olmuş olması. Kullanıcnın uygulamayı başlattığı an bu iki bilgi hazır olmuş olmalı.

createPrefObject, *saveThemeToSharedPref* ve *loadThemeFromSharedPref* adında sırasıyla, obje oluşturup getiren, objenin verisini kaydeden ve uygulama ilk açıldığında SharedPreferencesten veri alıp çekme görevleri olan üç adet fonksiyon oluşturdum.

Kullanıcı her switchTheme ile temayı değiştiğinde bu veriyi ShrPrf'a kaydedebiliriz. Her değişiklikte kayıt edilmesi daha güvenli olur. saveThemeToSharedPref(selected);

Main fonksiyonumuzu asyn bir hale getirip asyn bir fonksıyon cagırdım ve WidgetsFlutterBinding çağırdık.

MaterialApp return edilmeden loadThemeFromSharedPref fonksiyonum ile uygulama baslamadan once tema bılgısını edındırdım.

Uygulama async calısıp obje olusturup bu objenın loadThemeFromSharedPref metotu ile veri alınıp isPurple a atanacak ardından materialApp return edilip tema bilgisindeki selectedThemeData çağrılılcak, o da _selectedThemeData yı okuyup color_theme_data içerisinde tema kontrolünün yapıldığı yere gidicek.

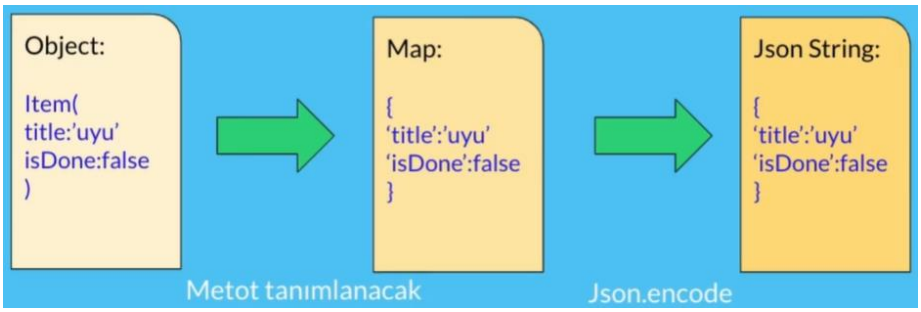
Nesneleri String'e Çevirme

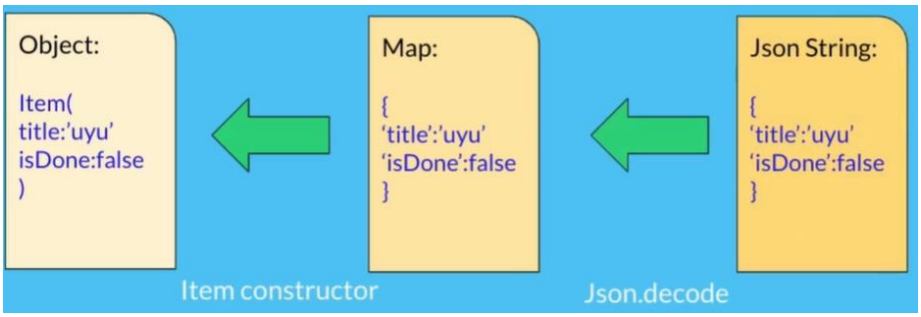
SharedPreferences a Item sınıfından oluşturulan nesnelerin title ve isDone property'si, instances variable'ı oluyor. SharedPrefernces sadece primitive ve stringlerden oluşan listeleri saklama becerisine sahip. Bu özel sınfıtan oluşan özel objeleri SharedPrefernces'a kaydetmek için her Item verimizi string olarak liste içerisinde saklamamız gerekir.

Her objeyi stringe çevrilip String tipinde bir listeye kaydedilip SharedPrefernces'a göndereceğim. Okuyacağım zaman ise string listesini tekrardan objeye dönüştürüp listviewbuilder'ın kullanabileceği Item lardan oluşan listeye dönüştürmem gerekir.

Herhangi bir objeyi stringe nasıl dönüştürürüz? JSON, veri değişimini kolaylaştıran bir metin biçimidir.

Yapacağımız işlem:





Gerekli tanımlamaları object_map_json dosyası altında yaptım.

Item objesının map hali için Map döndren, keyleri string ve value değerleri ise birden çok türde olabileceğinden dynamic tiğinde olan *toMap* adında bir çevirici fonksiyon tanımladım. Bu sınıftan oluşan her obje kendisne ait bu metotu kullanarak kendisinin map e dönüşmüş halini return edebildi.

Map'e dönüşmüş objeyi JSON'a dönüştürmek için json.encode ve json.decode kullandım.

Not: Herhangi bir sınıftan obje oluştururken Named Constructor ile istedğimiz kadar constructor oluşturabiliriz.

Item listesini kullanarak stringlerden oluşan bir liste oluşturup her bir elemanı alıp dönüştürerek başka bir liste içerisine yazdım. Her yeni eleman ekleme, silme ve işaretleme sonrası listeyi hemen oluşturup sharedpreferences'a kaydetme işlemini gerçekleştirdim. Uygulama ilk çalıştığında load metotu ile stringlerden olusan listeyi shared preferencstan alıp her öğeyi çevirip items listesi olusturup listview builder da bu items listesini kullandım.

Not: Dart'da herhangi bir değişkenin ardından . runtimeType metotu kullanılırsa uygulama çalıştıgında o objenin veri tipini öğrenebilirsiniz.

Görev Listesini Shared Preferences'a kaydetme

Consumer Widget

Widget agacının aynı kısmında birden fazla provider yayını dinlemek gerekebilir. Consumer Widgeti maksimum 6 yayına kadar, yayınların aynı anda dinlenebilmesini sağlar. Child property'si ile sadece özel olarak build edilecek kısımları belirtebiliriz.

Aynı isimli iki görev oluşurve biri silmek istenirse hata aldım. Sebebi her ikisinin de key değerinin aynı olmasıydı. Dismissible return eden item_card sayfasının key değerini Uniquekey() olarak değiştirince bu sorun flutter tarafından çözülmüş oldu.

Splash Screen

Splash Screen return eden ayrı bir SplashWidget oluşturdum. Splash Screen widgeti ve property'leri ile animasyonu hazırladım. Material app içerisinde bu widgeti return ettim.

Özet

State yönetimi local ve genel state yonetımı

State Management paketlerinin önemi için callbacklerle state yonetımı yapmaya calıstım.

Provider paketi ve nasıl kullanıldıgı

Provider, Changenotifier provider, multiprovider sınıflarını kullandım.

Hava Durumu Uygulaması

Arayüz Oluşturulması

API nedir?

API servis: OpenWeatherMap.org

http paketi

Future nedir?

Async & Await kullanımı

Json

Json ayrıştırma/parse

initState, setState, dispose: widget lifecycle metotları

Verilerin dinamik yenilenmesi

CircularProgressIndicator Widget'ı

Şehir Seçimi

Sayfalar arası basit veri döndürme

TextField input kontrolü / validation

Arkaplan otomatik güncelleme

Cihaz GPS verisi kullanma: Geolocator paketi

Try & Catch videosu hakkında

Try & Catch Anahtar Kelimeler

Hava Durumu icon simgeleri ekleme

5 Günlük tahmin kartları ekleme

ListView Widget Kullanımı

DailyWeatherCard'lara properties ekleme

Kart verisi çekme

DateTime & Duration Sınıfları

Gün bilgisini formatlama

Dart Liste Metotları

Sabit Uzunlukta Liste: Listedeki eleman sayısı belirli ve sabit olduğunda kullanılır.

Büyüyebilen Liste: [] ile oluşturulur ve add metodu ile üzerine ekleme yapılabilir.

Sabit Uzunlukta Liste Örneği:

```
final fixedLengthList = List<int>.filled(5, 0);
print(fixedLengthList); // [0, 0, 0, 0, 0]
fixedLengthList[0] = 87;
fixedLengthList.setAll(1, [1, 2, 3]);
print(fixedLengthList); // [87, 1, 2, 3, 0]
// Fixed length list length can't be changed or increased
fixedLengthList.length = 0;  // Error
fixedLengthList.add(499);    // Error
```

Büyüyebilen Liste Örneği:

```
final growableList = <String>['A', 'B']; // Creates growable list.
To add data to the growable list, use operator[]=, add or addAll.
growableList[0] = 'G';
print(growableList); // [G, B]
growableList.add('X');
growableList.addAll({'C', 'B'});
print(growableList); // [G, B, X, C, B]
```

Not: *Iterable Sınıfı* Dart da listelerin bir üst atası olan sınıfıdır.

Temel metotlar

High Order metotlar, fonksiyonun içine argüman olarak başka bir fonksiyon veren metotlardır.

Metotları ele alırken dikkat edilecek iki özellik:

1) Metodun return değeri var mı? Var ise tipi ne?

Örnek: ***bool*** any (***bool*** test (E element)) => bool return eden bir Yuksek mertebeli bir metot.

2) Üzerinde işlem yaptığı listede değişiklik yapıyor mu? Veya orijinal hali ile bırakıyor mu?

Not: List list1 = [‘a’, ‘b’, ...list2, ...list3] : list 2 listesinin elemanlarını list 1 listesinin devamına ekler.

Higher Order Metotlar ve Any Metodu

Any(): Listenin ilk elemanınından baslayarak alıp fonksıyon ıcerısınde donen degerle bir işlem yapar.

Örnek 1: bool testFunction(int i){return i==6;} list1.any ((item)=>testFunction(item))

Örnek 2: list1.any ((xx) {return item==6;}) / list1.any ((xx)=> item==6)

Every

Listedeki tüm elemanlar belirli bir testi geçer ise true döndürür. Any() yalnızca birinin olup olmamasına bakarken evert() belirtilen durumda olmasına bakıyor.

Expand

firstWhere

Testi karşılayan ilk elemanı döndürür.

forEach

map

Iterabel döndürür ve herhangi tip kabul eder. ListView ile kullanılabilir.

reduce & fold

int sum (int a, int b) => (a+b); print(ilk5.reduce((prev,next)=>sum(prev,next)));
// Çıktı: 15

reduce işe ilk başladığında prev ve next olarak iki veri alır. Listenin ilk iki elemanını kullanıyor. Bazen belirli bir başlangıç değeri vermemiz gerekir ise fold kullanılır.Fold başka veri tipi döndürebilir. Yani prev ve next değerlerinin tiplerinin farklı olduğunu düşünebiliyor.

int sum (int a, int b) => (a+b); print(ilk5.fold <int>(100, (prev,next) =>
sum(prev+next)));

//Çıktı: 115

Ürünler adlı bir sınıfımız olsun. Her ürünün ad ve fiyatı olsun. Kullanıcı ürün sınıfıından oluşan bir listeyi dolduruyor. Alışverişi tamamla kısmında toplam sepet tutarnı görüntülemek istiyoruz. Listeler int içermiyor, özel bir ürünler sınıfının objeleri var ve bunlar price değişkenine tanımlı değerleri tutuyor. forLoop ile toplanabilir. Fold ise tek satırda şu şekilde yapabilir:

Products.fold<int>(0, (prev,next) => prev + next.price) int (0) ve price bilgisi döndürür.

sort

int döndüren bir compare fonksiyonunu argüman olarak alır. Bu fonksiyon listemizden iki eleman alır. Sort ise bu elemanları sıralayıp dizer. Liste üzerindeki kıyaslama işlemi birbirlerinden farklı birden çok Sorting Algorithm'ler ile sağlanır. Sort un arkaplanda

listeyi nasıl sıraladığını bilmek için hangi sorting algoritmasını kullandığını bilmek gerekir.

indexWhere, lastIndexWhere, lastWhere, removeWhere, retainWhere, singleWhere, skipWhile

takeWhile & where

Firebase Authentication App

Firebase Hakkında

Firebase bir back-end service'dir. Bizim back-end servisine ihtiyacımız var. Firebase, sunduğu çözüm, doküman bolluğu ve yine Google desteği ile ön plana çıkmakta

Authentication

Firebase ın sunduğu hizmetlerden biridir**.** Kimlik doğrulamasıdır. Uygulamalarımızın database'ı olmasa bile kullanıcıların authentication olması için firebase'ın bu hizmetini kullanabiliriz.

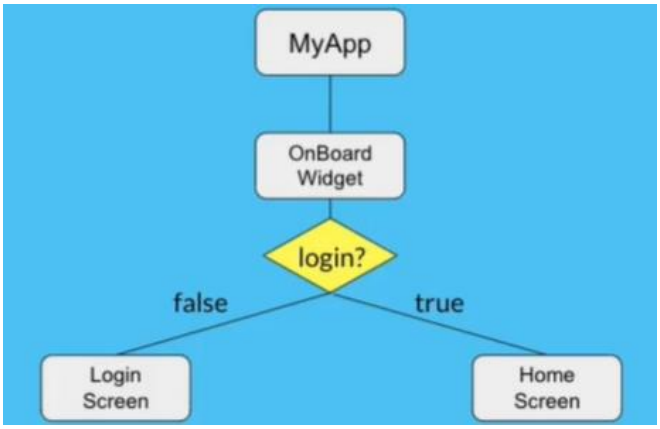
1) *Yerel Kimlik Doğrulama :* Örneğin parmak izi veya faceId

2) *Remote/Back-end sistem üzerinden kimlik doğrulama:* Giriş yap denilince o bilgiler başka bir yere yollanıp onaylama kodu ile giriş izni sağlanıyor.

Google, facebook, github gibi hesaplarla giriş seçeneğini firebase bize hazır metotlar ile kolay ve hızlı olarak sağlıyor.

Authentication: Widget Mimarisi

Genelde login bilgisi kalıcı olarak tutulur. Örnek bir Login işlemi widget şeması şu şekilde olabilir:



Firebase Projesi Oluşturmak

Firebase Paket Kurulumları ve Firebase Authentication Paketinin eklenmesi

Core paketi her zaman kurulmalı ki firebase başlatılabilsin.

```
firebase_core: ^2.4.0
cloud_firestore: ^4.2.0
firebase_auth: ^4.2.1
```

Firebase Başlatılması / Initialization - main () async yöntemi

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(MyApp());
}
```

Firebase Başlatılması / Initialization - FutureBuilder() yöntemi

await Firebase.initializeApp(); kısmını ve maindeki async fonksiyonu sildikten sonra Wmyapp Classı içerisine:

```
final Future<FirebaseApp> _initialization = Firebase.initializeApp();
```

Firebase kurulum testi

Sayfaların Oluşturulması

Button classı oluşturup tanımladığım değişkenleri atadım ve main kısmında kullandım.

Flutter signin button veya sign_button paketelri kullanılabilirdi hazır butonlar için.

Firebase Authentication

Bir sisteme ulaşmadan önce kimliğin doğrulanması işlemidir. Giriş yapmak için kullandığımız bilgiler uzak bir sistemde kontrol edildikten sonra işlenerek giriş yapılmasına izin verilir.

Firebase içerisindeki sign-in metotlardan kullanacağım üç provider ı enable ettim.

Anonim Authentication

Neden anonim giriş gereklidir? Bu geçici anonim hesaplar, uygulamanıza henüz kaydolmamış kullanıcıların güvenlik kuralları tarafından korunan verilerle çalışmasına izin vermek için kullanılabilir. Anonim bir kullanıcı uygulamanıza kaydolmaya karar verirse, gelecekteki oturumlarda korunan verileriyle çalışmaya devam edebilmesi için oturum açma kimlik bilgilerini anonim hesaba bağlayabilirsiniz.

signInAnonymously() ve signOut() fonksiyonlarını butonların onpressed inde kullandık. Fakat bu doğru bir kullanım değil.

Auth Servisi Eklenmesi

User sınıfı kullandım. Bu sınıf bir kullanıcının sahip olabileceği verileri, özellikleri user objesi ile döndürür.

Not: Flutterda tüm widgetlar bir sınıftır.

Provider ile Servis Kullanımı

Provider, başka sınıftaki objeleri ve metotları başka bir yerde yayınlayabilmemizi sağlar. Aynı zamanda servise ait objeleri de widget ağacına yayınlamamızı sağlar.

Oluşturduğum auth classını en tepede provide ederek diğer tüm widgetların bu sınıfın objelerine ve metotlarına erişebilmesini sağladım.

Kullanıcı, logut butonuna tıklayarak çıkış yaptı. Sign in anonymously butonu ile tekrar girdi. Her çıkış ve girişte kullanıcı id si farklı olarak yeniden tanımlandı. Console print ederek Id leri takip ettim.

Her giriş yapıldığında firebase consolundan eklenen kullanıcıları görüp yönetebiliyorum.

OnBoard Widget Eklenmesi

HomePage eklenmesi

authStateChanges metodu ile State Kontrolü

Kullanıcının login veya logout pozisyonunda olduğunu öğrenme

Kullanıcının login logout bilgisi bir değişken içinde tutulmamalı, firebase tarafından bize sağlanmalı.

Çoğu durumda, kullanıcınızın oturum açmış mı yoksa oturumu kapatmış mı olduğu gibi kimlik doğrulama durumu hakkında bilgi sahibi olmanız gerekir. Firebase Auth, bir Akış yoluyla bu duruma gerçek zamanlı olarak sahip olmanızı sağlar. *Stream*, bir kez çağrıldığında, kullanıcının geçerli kimlik doğrulama durumuna ilişkin anında bir olay sağlar ve ardından, kimlik doğrulama durumu değiştiğinde sonraki olayları sağlar.

Kimlik doğrulama durumu değişikliklerini dinlemek için üç yöntem vardır:

1. authStateChanges()
2. idTokenChanges()
3. userChanges()

Stream Nedir?

StreamBuilder, aktif olarak sürekli yeni event verilerini sağlayan bir Stream bağlantısının verilerini ekrana anlık olarak basmasını sağlayan bir widget'tır. Bir Akış ile etkileşimin en son anlık görüntüsüne dayanarak kendisini oluşturan widget.

Uzakta birileri bir şey yapcak, bilgi değişecek ve bu bilgi de bana gelecek.

Paketler bir yerlerden bize akıyor. Bana paket geldiği zaman bu pakete bakıp içindekinin durumuna göre bir işlem yapıyorum. Stream mantığında bize paketler içerisinde değişik veri tipleri gelebilir. Stream birden çok arka arkaya zamanı belli olmayan verilerin akışını ifade eder. Stream'ler Future'dan oluşan bir liste gibi düşünülebilir.

Stream programımıza nereden gelecek?

-API, Network veya Package üzerinden dışarıdan gelebilir.

-Stream döndüren bir async Fonksiyon yazılabilir. Stream<int> myFunct() async*{yield int;}

-StreamController adlı sınıf kullanılabilir.

Not: Stream de return yerine yield kullanılır. async yerine async* kullanılır.

Stream döndüren bir fonksiyon kullanarak 1 den 10 a kadar olan sayıların karesini 2 saniye ara ile aldım.

StreamController sınfını kullandım ve Streamden akan verileri dinledim. Bu kullanım öneriliyor.

Stream oluşturma veya gelen streamı alıp listen metodu ile subscription/abone olmayı ve streamden akan veriyi basitçe kullanmayı öğrenip uyguladım.

StreamSubscription metotları

Listen metodu ile streamı dinlediğimizde ona abone oluyoruz. Listen metodu ise Stream Subcription objesi döndürür.

Stream ile ilgili her işi controller ile yapıyoruz.

Event

onDone

onError

cancelOnError

setState ile Giriş yönlendirmesi

listen metodu ile bir kez çağrıldığı zaman stream bize o anda hemen kullanıcının loign durumunu getirir. Uygulama ilk çalıştığında bilgi gelir ardından stream kesilir. Her login-logout oldukça stream tekrar yayınlanır. Devamlı bir yayın olmaz, değişiklik oldukça olur.

Not: Statefull widget ilk çalıştığında build metodu çalışmadan önce state'i başlatırken yapılması gerekenler *iniState()* içerisine yazılır.

Not: "singleton" sadece bir tane obje üretilmesi gereken durumları ifade eder.

Gelen verinin durumuna göre hompage veya signinpage e gittik.

StreamBuilder Widget Kullanımı

AsyncSnapshot, streamden gelen verinin paketlendiği sınıf.

Auth servisi üzerinden firebase e bağlanmak için bir authStatus adlı metod oluşturdum. Çünkü widgetler üzerinden direkt bağlanmak istemiyorum. Provider yardımı ile auth objesi oluşturdum. Bu objenin authStatusunu StreamBuilder a stream olarak verdim. Yani streambuilder ın dinleyeceği stream ı auth objesinin authStatus

metodu verecek dedim. Stream den bir şeyler tetiklendikçe buildera verilen callback fonksiyonu çalışır. Bu callback fonksiyon context ve snapshot adlı iki değişmez, sabit argüman alıyor. Yapılan kontrol:

Streamden akan son verinin connetcionı aktif ise içerisindeki dataya bak. Data null değil ise HomePage null ise SignInPage döndür. Yani aktif kullanıcı var iste homepage, yok ise sign in page.

Streamden akan son verinin connetcionı aktif değil ise CircularProgresIndıcator göster.

Buton yükleme durumu

Sign in Anonymously butonuna basınca async bir işlem başlıyor ve bu işlemin gerçekleşmesi, sonuçlanması bekleniyor ancak daha sonuçlanmadan ben bu butona basmaya devam edebiliyorum. Böyle olmaması gerek. Home page gelene kadar bir suru giriş denemesi yapılabiliyor şuan.

İlk tıklandığında buton deaktif olmalı. Bunu isLoading değişkeni ile sağladım.

Buton Refactoring

Butonun onpressed ini Future döndüren başka bir fonksiyonda tuttum.

Email ve Şifre ile Giriş Sayfası

buildSignInForm(): Form widgetı kullandım ve metot olarak extract ettim. Widget olarak etmedim çünkü set state'ine erişebilmem için ana stateimin içerisinde kalması gerekiyor.

buildRegisterForm(): Kayıt Formu ekranı.

FlatButton yerine yeni gelen TextButton'ı kullandım

Return kısmına duruma göre hangi ekranın geleceğini belirmem gerek.

enum & Formların oluşturulması

Enum yani, Enumerations (Numaralandırmalar) string ifadelerle sayısal karşılaştırma ya da farklı işlemlerin gerektiği noktalarda yazılımcılar için daha da okunabilirlik sağlayan ve gelecekte anlaşılmasını zorlaştıran kod karmaşasını azaltmayı sağlayan yardımcı bir yapıdır. Değişkenlerin alabileceği değerlerin sabit (belli) olduğu durumlarda programı daha okunabilir hale getirmek için kullanılır. Programda birçok değişkene tek tek sayısal değer vermek yerine "enum" kullanılabilir.

Form decoration

Form validation

Kullanıcının girdiği veriler firebase sunucusuna gönderilmeden önce kontrol edilmeli. Bu sebeple tüm form ile ilgili elemanları Form widgetı ile sarmalamıştım. Böylece form içindeki elemanların durumlarını kontrol edebilir ve butonun fonksiyonunu bu duruma göre çalıştırabiliriz.

GlobalKey ile formKey oluşturdum. Formun durumunu kontrol edeceği için FormState sınıfını verdik. Bu formKey'i Form widgetıma key olarak atadım. Bu anahtarların form validate adlı metotları var. Bu metotlar, form içindeki elemanların kendilerini onaylayıp onaylamadığını kontrol ediyor. Her bir form alanına tek tek bakarak validate property'sine bakıyor.

Not: *Regular Expression:* Bu gibi girilen metinleri kontrol etme durumları ve kısıtları RegEx başlığı altında inceleniyor.

email_validator paketi

RegEx yerine basit bir email validation yöntemi kullandım.

Şifre, email ve onay formları için gerekli kontrolleri sağladım.

Yeni Kullanıcı Oluşturma, createUserWithEmailAndPassword()

Firebase ile iletişime geçiyoruz.

Firebase ın bize giriş için verdiği metotlar

UserCredential: Herhangi bir istek yapıldığında bu tipte bi istek döner

User: UserCredentialın alt sınıfı gibi düşünebiliriz.

Try catch yapısı kullandım.

createUserWithEmailAndPassword metodu ile kullanıcının girdiği kayıt sayfasındaki stringleri alıp bu metod ile firebase ulaşacağım.

Email Verification

Kayıt ol butonuna tıklandığında firebase yeni bir kullanıcı oluşturuyor. Bu olurken aynı zamanda kullanıcıya doğrulama linki gönderilsin. Bu işi user sınıfına ait verifyemail metodu ile biz tetikliyoruz.

```
if (!user.emailVerified) {
  await user.sendEmailVerification();
}
```

Firebase Mail Template

alertDialog eklenmesi

Kullanıcı Girişi, signInWithEmailAndPassword

Kayıt olan kullanıcıyı giriş yaptırdım. Öncelikle email verifiy edildi mi diye kontrol ettim.

Şifre Sıfırlama, Reset Password

Google Sign-In Ayarları

Google-sign-in paketini kurdum.

Googledan token verisi alınır ve bu tokenler firebase metodlarına arguman olarak verilir.

signInWithGoogle metodu

Google SignOut metodu

App bar daki signout ıconuna tıklayarak firebase den çıkış sağladık. Google tarafında da çıkış yapmam gerekir. Servisler kısmındaki signOut metoduma googledan da çıkmamı sağlayan bir metot ekledim.

Hata İşleme/Error Handling

Programlarımız çalışırken yani run time sırasında beklenmedik durumlar oluşabilir. Bunların önüne geçmek için hata fırlatıp yakalayan (try, catch) bir yapı oluşturmamız gerek. Async yani network işlemlerinde hata olasılığı daha yüksek olduğu için hata yönetimi bu durumlarda daha önemli olacaktır.

Hata Fırlatma (try) ve Hata Yakalama (catch)

Fonksiyona argüman olarak verilen değere 100den buyuk olursa *throw* ile bir hata fırlattıralım.

Uncaught Error: Yakalanmamış hata, ortaya birisi hata fırlattı ama bunu yakalayan bir yapı yok.

UnHandled Exception: Bir exception var ve bunun üstesinden gelecek herhangi bir şey yapılmadı. Havaya fırlatılan throw ile ilgilenen herhangi bir şey yok. Tehlikeli bir durumdur. Çünkü bu hatayı alıyorsak ön göremediğimiz veya yönetmediğimiz/yönetemediğimiz bir durum söz konusudur.

Fırlatılan şeyin veri tipi farketmez çünkü dart dilinde her şeyin türü objedir. Yine de fırlatılması için özel olarak yapılan Exception() sınıfı vardır. Exception(), herhangi bir hata durumunda kullanıcıya bilgi göstermek amacı ile kullanılır. Yani programatik olarak hata adresi verilir. Bunun yakalanması beklenir.

Fonksiyonu çalıştırmayı denemek (try) gerekir çünkü bu kısımda olağan bi hata olabilir.

Try, catch, on, finally, rethrow

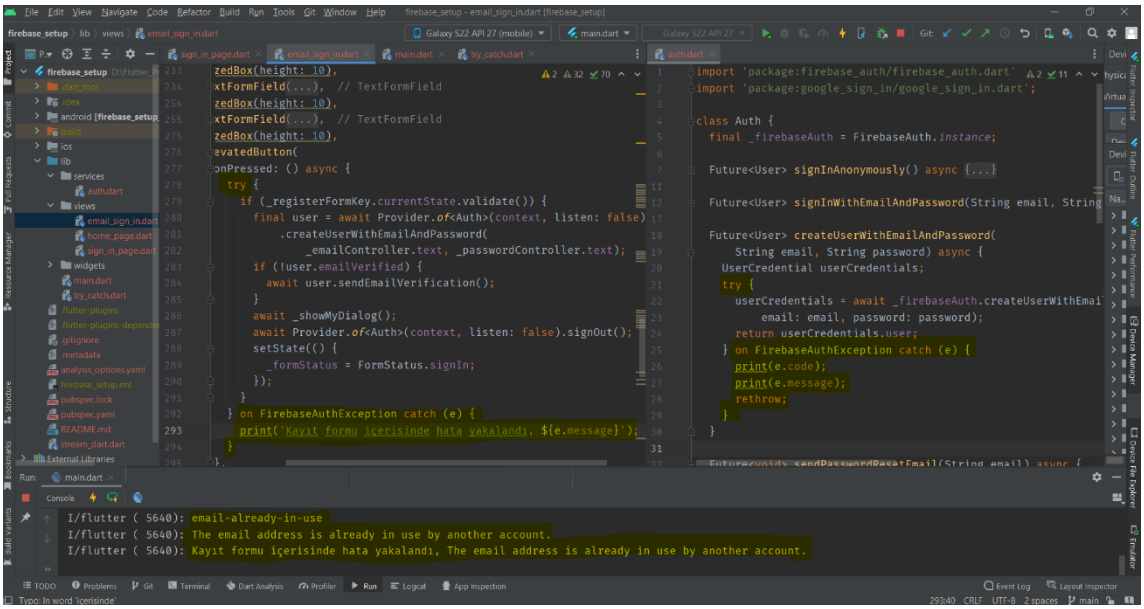
Örneğin bir authentication işlemi sırasında şu gibi bazı hatalar oluşabilir:

Network hatası veya authentication ile ilgili bir hata. Gelen hatanın ne olduğuna bakıp buna göre işlem yapmamız gerekiyor. Exception sınfından türetilen alt sınıflar bu işlemleri yapar.

Hata İşleme Pratik

Uygulamada giriş yapmak için üç tane buton bulunuyor ve bunlardan birinde bir hata oluşturup firebase den gelen hatayı aldım.

Önceden oluşturup giriş yaptığım hesabımla tekrar kayıt olmaya çalıştım. Hatayı yakaladığım için Unhandled Exception ile karşılaşmadım. Gelen hatayı rethrow ederek butonun fonksiyonunda kullandım.



Özet

Firebase kullanarak anonim giriş, e-mail ve password ile yeni kullanıcı oluşturma ve giriş yapma, password resetleme ve setState yapısı, try catch ile hata yönetimi,

Library App (Firebase Cloud Firestore & MVMM)

Cloud Firestore Veri Yapısı

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

Bu derste local veri tabanından bahsedilmedi, bulut veri tabanı göreceğiz.

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 sınfılar 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.

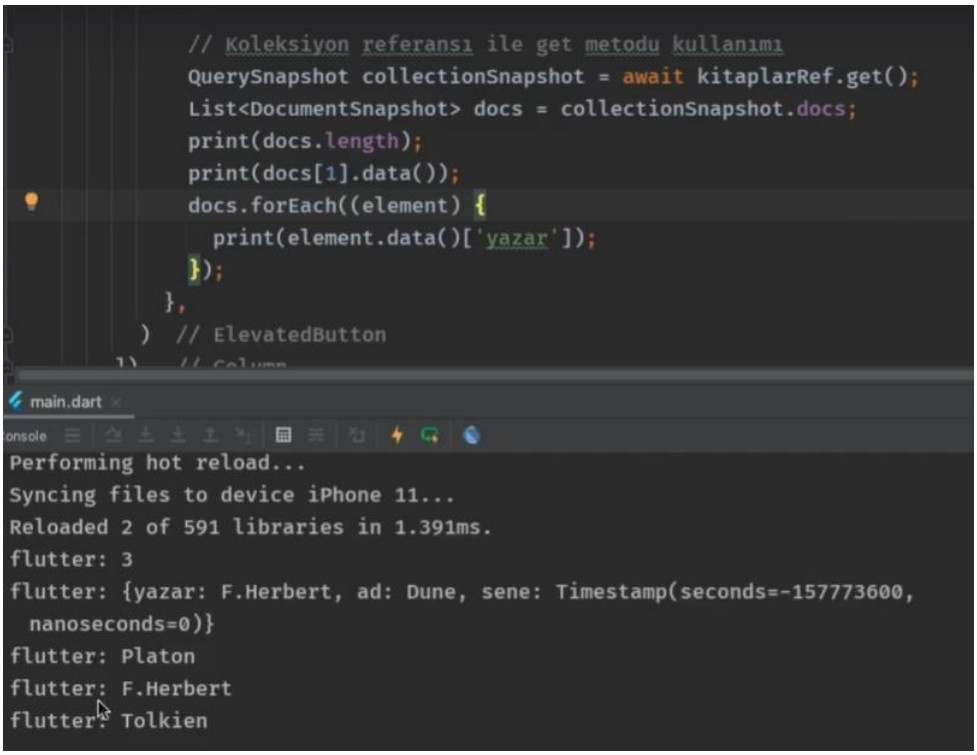
```
ElevatedButton(
  child: Text('GET DATA'),
  onPressed: () async {
    DocumentSnapshot documentSnapshot = await hobbitRef.get();
    Map<String, dynamic> data = documentSnapshot.data();
    print(data);
```

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

FirebaseFirestore _database = FirebaseFirestore.instance;

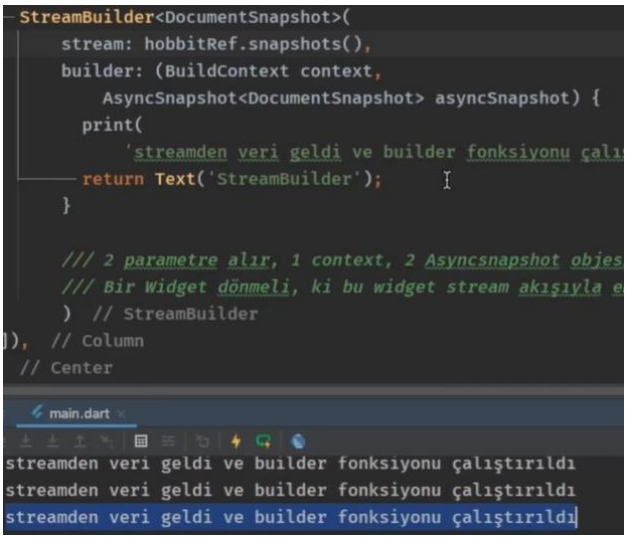
Collection	Document
CollectionReference *_database.collection('kitaplar');*	DocumentReference *_database.collection('kitaplar').doc('Dune');*
QuerySnaphot *_querySnapshot.doc → List<DocumentSnapshot>* *list[index].data() → Map<String,dynamic>*	DocumentSnaphot *_documentSnapshot.data() → Map<String,dynamic>*

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:

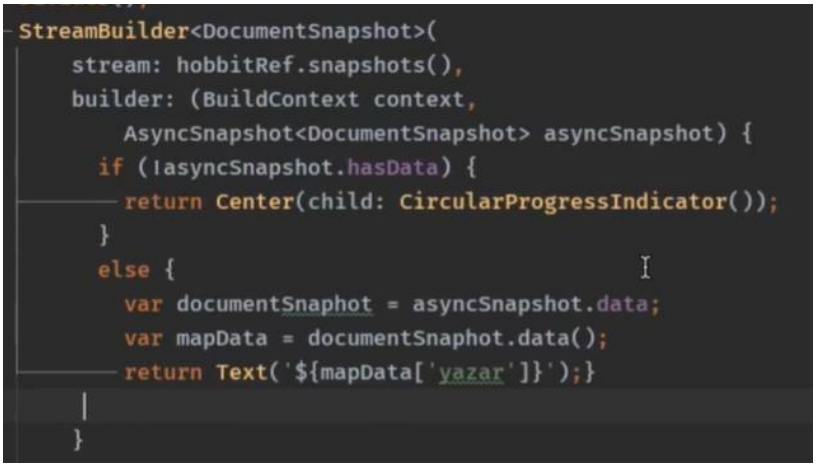
Dinlenilen streamdeki değişikliği firebase tarafından gelen bilgi ile StreamBuilder fark etti ve builder'a verilen metod çalışarak değişiklik arayuze de yansıtıldı



StreamBuilder ile doküman dinleme

AsyncSnapshot içinden DocumentSnapshot, DocumentSnapShot içerisinden de map çıkartılır.

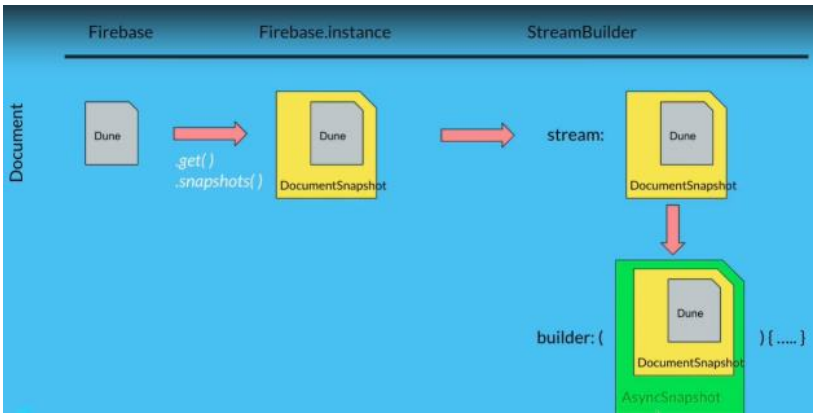
StreamBuilder içinde data henüz gelmedi ise CircProgIndıcator cevırılebilir.



"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

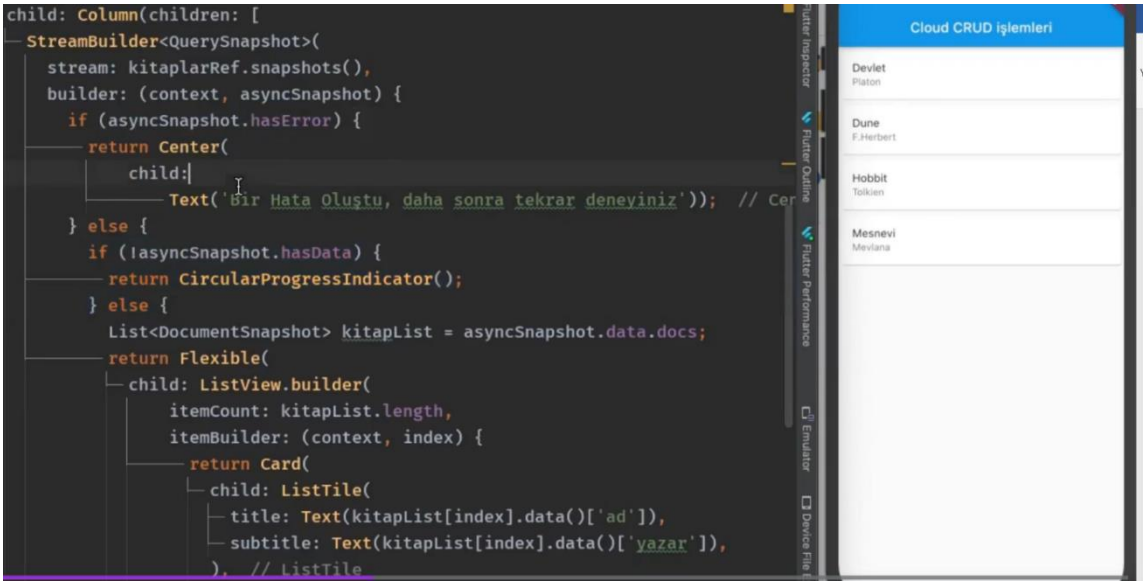
3 veri geldi ve kullanılabilir

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.

```
class MyApp extends StatelessWidget {
  final Future<FirebaseApp> _initialization = Firebase.initializeApp();
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return StreamProvider<QuerySnapshot>(
      initialData: null,
      create: (_)=>FirebaseFirestore.instance.collection('kitaplar').snapshots(),
      child: MaterialApp(
```

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

```
Consumer<QuerySnapshot>(
  builder: (context, querySnapshot, child) {
    if (querySnapshot == null) {
      return CircularProgressIndicator();
    } else {
      List<DocumentSnapshot> kitapList = querySnapshot.docs;
      return Flexible(
        child: ListView.builder(
          itemCount: kitapList.length,
          itemBuilder: (context, index) {
            return Card(
```

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:

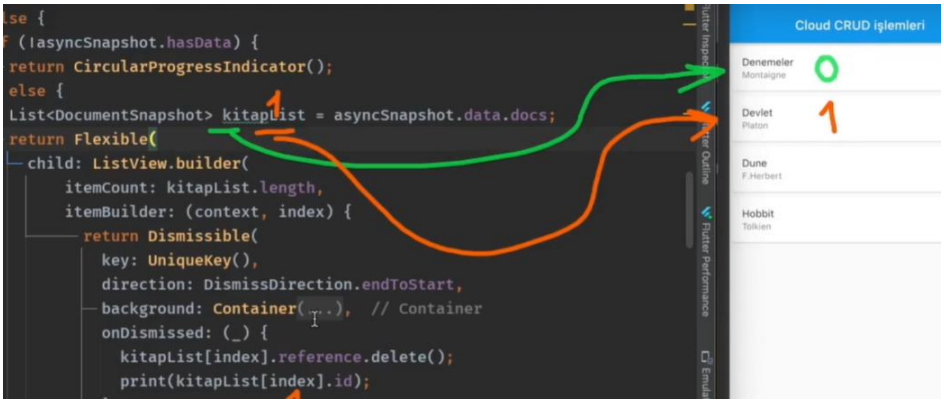
```
floatingActionButton: FloatingActionButton(
    onPressed: () async {
      //await kitaplarRef.doc('Denemeler').set(bookToAdd);
      await _database
          .collection('kayipKitaplar')
          .doc('Harry Potter')
          .set({'ad': 'Harry Potter', 'yazar': 'Rowling', 'sene': 2000});
 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

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:

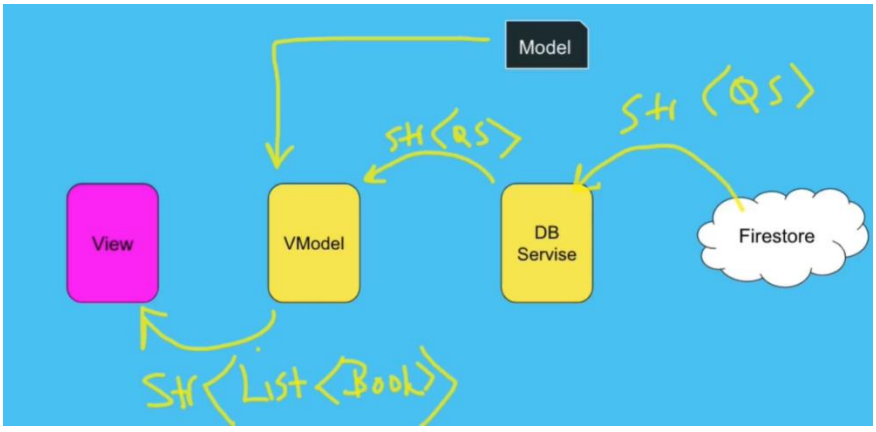
```
factory Book.fromMap(Map map) => Book(
    id:map['id'],
  bookName: map['bookName'],
  authorName: map['authorName'],
  publishDate: map['publishDate']
```

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

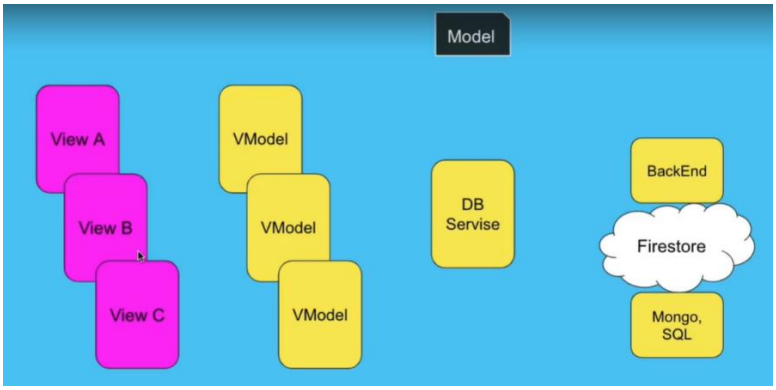


***View*:** Widgetleri tutan, birşeyler gösteren ve hesap yapmayı bilmeyen dumb bir kısım.

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.

```
/// objeden map olusturan

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.

```
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.