



DATABASE HIVE IN FLUTTER

2023



► Seyyed Amir Nima Ghaebi

فهرست مطالب

۳	مقدمه
۴	چرا دیتابیس Hive
۵	فصل اول : نصب و راه اندازی دیتابیس در فلاتر
۶	اضافه کردن دیتابیس به پروژه
۷	Initialize
۸	فصل دوم : Box چیست و نحوه استفاده از Box های مختلف
۹	مفهوم Box در دیتابیس Hive
۱۰	کاربرد Lazy box
۱۱	ذخیره کردن داده های حساس با استفاده از Encrypted box
۱۳	فصل سوم : ذخیره دیتا های جزئی و غیر آبجکتی
۱۴	Initialize
۱۵	Create
۱۶	Read
۱۷	Update and Delete
۱۸	نکاتی درباره ذخیره دیتا های غیر آبجکتی
۲۲	فصل چهارم : بررسی پیشرفته ویژگی های دیتابیس Hive
۲۳	ValueListenable در دیتابیس Hive
۲۶	Add method
۲۷	Auto increment & indices
۲۹	فصل پنجم : ذخیره آبجکت های شخصی سازی شده در دیتابیس Hive
۳۰	TypeAdapters چیست؟
۳۱	تعریف جدول در دیتابیس Hive
۳۲	ساخت کلاس TypeAdapter برای ذخیره کتاب
۳۶	Register Adapter
۳۷	Read and show in listview
۴۰	Add & Edit book
۴۵	Output

۴۶	فصل ششم : ساخت پروژه Todos ساده
۴۷	شروع پروژه
۴۸	ساخت کلاس Register Adapter & TypeAdapter
۵۱	Create , Read , Update , Delete
۵۷	ValueListenableBuilder در دیتابیس Hive
۵۹	باز نویسی متد dialogEditTodo
۶۰	فصل هفتم : بررسی مباحث پیشرفته و کاربردی دیتابیس Hive
۶۱	Default value
۶۲	Enums
۶۳	Relationships
۶۴	Snippets for VSCode
۶۶	پیدا کردن دیتا با استفاده از کلمه
۶۷	جستجو در دیتابیس و دیدن نتایج آن در ListView
۶۹	فصل هشتم : بررسی تمام متد های مهم و کاربردی پکیج Hive
۷۰	isEmpty & isNotEmpty
۷۰	hashCode
۷۱	Values
۷۱	isOpen
۷۲	Keys
۷۲	Lazy
۷۳	Length
۷۳	name
۷۴	toMap
۷۴	containsKey
۷۴	compact
۷۵	Clear
۷۵	deleteFromDisk
۷۶	فصل نهم : ذخیره فایل و تصاویر به دو روش در دیتابیس Hive
۷۷	ذخیره تصاویر به صورت بایت در دیتابیس
۷۹	ذخیره تصاویر به صورت آدرس در دیتابیس
۸۳	سخن پایانی
۸۴	منابع

یکی از مهم ترین کار هایی که در برنامه نویسی موبایل با اون سروکار داریم ذخیره اطلاعات به صورت دائمی در گوشی کاربر هستش.

بسیاری از اپلیکیشن ها به خصوص اپلیکیشن های موبایلی، به مدیریت اطلاعات روی تلفن موبایل نیاز دارند، به همین دلیل، برای ذخیره سازی این اطلاعات از دیتابیس ها (پایگاه های داده) استفاده می کنیم. این عمل به دو روش ذخیره اطلاعات روی دیتابیس آنلاین در سمت سرور و ذخیره اطلاعات به صورت لوکال (محلی) روی خود موبایل صورت می گیرد.

در این مقاله، به ذخیره اطلاعات به صورت لوکال می پردازیم. دیتابیس های مختلفی برای این کار وجود دارد که در این مقاله، از دیتابیس Hive استفاده می کنیم. این دیتابیس سریع، سبک و NoSQL است و برای اپلیکیشن های فلاتری و دارت توسعه یافته و قابل استفاده است.

فلاتر گزینه های مختلفی برای ذخیره سازی داده به صورت محلی در اختیار ما گذاشته که برنامه نویس بر اساس کاربرد اپلیکیشنی که در حال توسعه اون هست میتونه بین این گزینه ها یکی رو انتخاب کنه برای مثال دیتابیس Shared_Preferences یکجی مناسبی برای ذخیره ساختارهای کوچک Key-Value در فلاتر هست که یک مقدار رو به عنوان کلید میگیره و یک مقدار رو به عنوان مقدار اون کلید که مثلاً برای ذخیره مقدار روشن یا خاموش بودن دارک مود میتونه گزینه خوبی باشه.

یکی دیگه از گزینه ها، دیتابیس Sqflite هست که یکجی هست بر پایه Sqlite که در فلاتر استفاده میشه. این دیتابیس انتخاب خوبی هست برای ذخیره کردن داده ها در جداولی که دیتاها در اون جداول با هم ارتباط دارن و در ارتباطاتشون پیچیدگی زیادی وجود داره و ما در چنین مواقعی از این دیتابیس استفاده میکنیم.

ولی اگر ما میخوایم که از دیتابیس استفاده کنیم که پیچیدگی کد نویسی کمتری داشته باشه و سریع و امن باشه و وابستگی نیتو نیاز نباشه و بتونیم حتی در فلاتر وب هم از اون استفاده کنیم، دیتابیس Hive گزینه خیلی خوبی برای ما هست.

من در این مقاله میخوام صفر تا صد آموزش دیتابیس Hive رو به شما دوستان عزیز ارائه بدم خوشحال میشم تا آخر این مقاله ما من همراه باشید تا باهم این دیتابیس قوی رو در فریمورک فلاتر یاد بگیریم.

آیا این مقاله پیش نیازی هم دارد؟

بله ، شما برای اینکه بتونید از دیتابیس Hive استفاده کنید ، باید کمی با دارت و فلاتر کار کرده باشید.

چون یادگیری دیتابیس در زبان فلاتر دارای مباحث پیشرفته ای هست که شما برای درک این مباحث باید مقدمات دارت و فلاتر رو بدونید.

. چرا دیتابیس Hive

ما در فلاتر دیتابیس های مختلفی داریم که هر کدام از دیتابیس های مزایا و معایب خاصی دارن. دیتابیس سبکی هست از لحاظ حجمی و راه حل سریعی هست برای استفاده از ذخیره کردن ساختار Key-Value در اپلیکیشن و اینکه Cross-Platform هست و ما میتوانیم همزمان اون رو در (فلاتر موبایل، وب و دسکتاپ) استفاده کنیم که کراس پلتفرم بودنش مزیتی هست نسبت به دیتابیس Sqflite در فلاتر. و اینکه دیتابیس hive از نوع NOSQL دیتابیس برای فلاتر هست که نیازی به نوشتن دستورات Sql ای در فلاتر نیست و به طور اتوماتیک با استفاده از انوتیشن ها ساختارها رو برای ما میسازه

اما چرا دیتابیس هایو (Hive) ؟

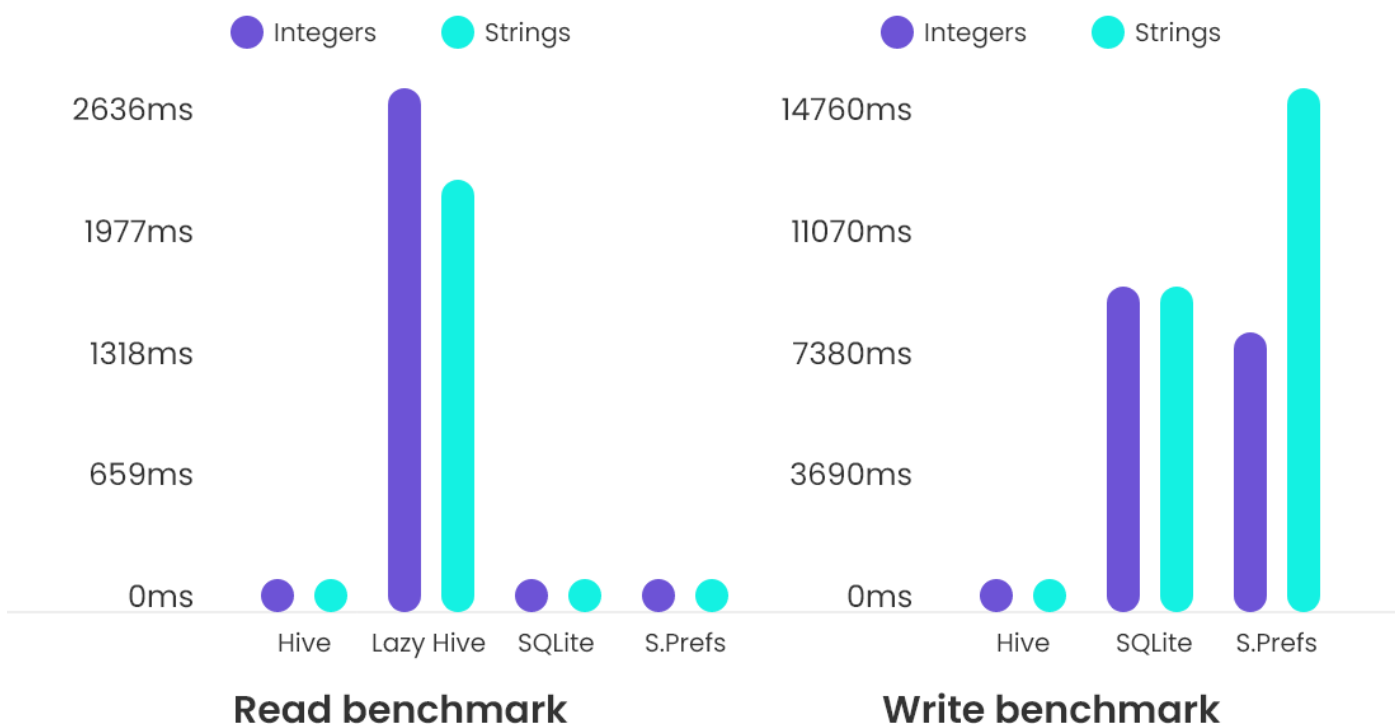
دیتابیس Hive یه دیتابیس NoSQL هست.

دیتابیس Hive یک دیتابیس فوق العاده سریع و سبک و امنی هست.

دیتابیس Hive در تمام سیستم عامل های Windows , Web , Mac , Linux , ios , Android قابل اجرا است.

این دیتابیس چقدر سریع است؟

همین طور که در تصویر زیر مشاهده میکنید، میتوانید در یک نگاه مقایسه ای بین سرعت معروف ترین دیتابیس های فلاتر داشته باشید.

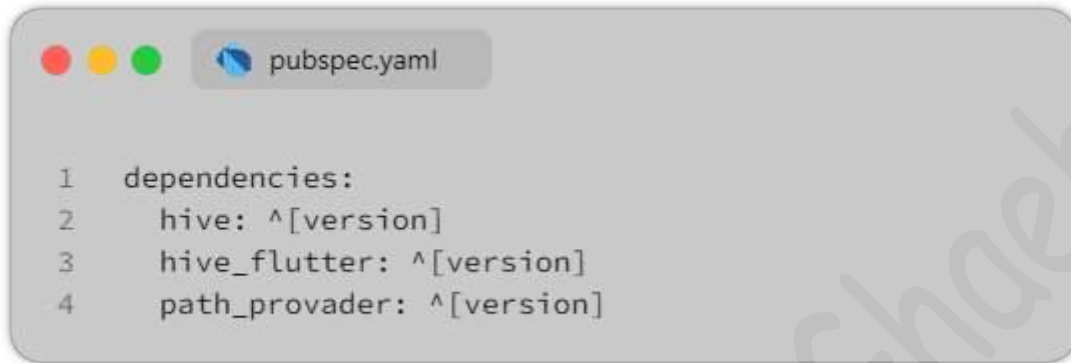




فصل اول : نصب و راه اندازی دیتابیس در فلاتر

. اضافه کردن دیتابیس به پروژه

اولین کاری که باید انجام بدیم برای استفاده از دیتابیس Hive اضافه کردن پکیج های مورد نیاز است. به همین منظور پکیج های زیر رو به فایل **pubspec.yaml** اضافه کنید.



```
1 dependencies:
2   hive: ^[version]
3   hive_flutter: ^[version]
4   path_provider: ^[version]
```

و سپس دستور flutter pub get را در ترمینال اجرا کنید. میتوانید آخرین ورژن پکیج های بالا رو از لینک های زیر بدست آورید.

Hive : <https://pub.dev/packages/hive>

Hive_Flutter : https://pub.dev/packages/hive_flutter

Path_Provader : https://pub.dev/packages/path_provider

Initialize .

برای راه اندازی دیتابیس Hive کافیست کد `await Hive.initFlutter();` رو در متد `main` اضافه کنید.

```
main.dart

1 import 'package:hive_flutter/adapters.dart';
2
3 void main() {
4   Hive.initFlutter();
5   runApp(const MyApp());
6 }
```

ولی برای اینکه بتونید از دیتابیس در پلتفرم های اندروید و IOS استفاده کنید، باید به صورت زیر راه اندازی رو انجام بدید.

```
main.dart

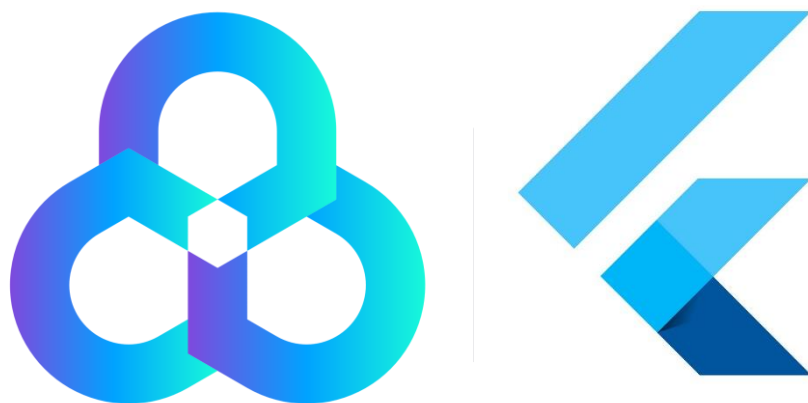
1 import 'package:hive_flutter/adapters.dart';
2 import 'package:path_provider/path_provider.dart';
3
4 void main() async {
5   WidgetsFlutterBinding.ensureInitialized();
6   final directory = await getApplicationDocumentsDirectory();
7   Hive.init(directory.path);
8   runApp(const MyApp());
9 }
```

در کد بالا و در لاین ۶ ما اومدیم و از پکیج `path_provider` استفاده کردیم و مسیر پوشه اپ خودمون رو در گوشی کاربر پیدا کردیم و آدرس رو در متغیر `directory` ریختیم.

و در لاین ۷ اومدیم و راه اندازی دیتابیس رو انجام دادیم و آدرس ذخیره سازی رو بهش پاس دادیم.

شما میتوانید از پکیج `device_info_plus` و یا استیت منیجمنت `GetX` استفاده کنید و تشخیص بدید که آیا اپ در گوشی اجرا میشه و یا در وب.

و بعد از تشخیص درست میتونید دیتابیس رو `Initialize` کنید.



فصل دوم : BOX چیست و نحوه استفاده از BOX های مختلف

. مفهوم Box در دیتابیس Hive

دیتابیس Hive از یک مفهومی به اسم Boxes برای ذخیره کردن دیتا در دیتابیس استفاده میکنه، اگه قبلا با دیتابیس Sql کار کرده باشین با مفهوم Table در اون آشنا هستین و یک Box مفهومی شبیه به Table هست با این تفاوت که Box ها فاقد ساختار دقیقی هستن، Box ها انعطاف پذیر هستن و اون ها فقط میتونن ارتباطات ساده بین دیتا ها رو هندل کنن.

قبل از اینکه دسترسی پیدا کنیم به داده های ذخیره شده یک جعبه ما باید اون Box یا جعبه رو باز کنیم، این کار کل محتوای داخل جعبه رو در مموری بارگزاری میکنه و باعث میشه خیلی راحت و سریع به محتوای داخل جعبه قابل دسترسی باشند.

این دو خط پایین یک مثال ساده از این هست که چطور جعبه ای به نام peopleBox رو باز میکنیم و پارامتر name این جعبه رو دریافت میکنیم.



```
main.dart

1 var box = await Hive.openBox('peopleBox');
2 //add 👉
3 box.put('name', 'Amir');
4 //raed 👉
5 String name = box.get('name');
```

فسنق! Cpy! وب هان غ فله ان بن رن! وب هان ذم فله سنق! Cpy! وب اوقال نك ولس بن م! Mb(z! cpy! مي! Fodszqf e! cpy! ان صو;

. کاربرد Lazy box

همون طور که گفتیم جعبه های معمولی داده های مارو از حافظه محلی در مموری بارگیری میکنند و جعبه ما باز میمونه، اما وقتی داده های سنگینی داشته باشیم این راه خوبی برای دسترسی به داده ها نیست و حجم زیادی از داده وارد مموری ما بشه ممکنه باعث وجود خطا در برنامه بشه و حافظه زیادی رو اشغال کنه، به این خاطر ما از Lazy box استفاده میکنیم.

در Lazy box ، فقط کلید ها یا Key برای خواندن و ذخیره کردن داخل مموری بارگزاری میشن، و ما میتونیم با استفاده از کلید داده مورد نظرمون رو از داخل جعبه فراخوانی کنیم. دوطرفه زیر رو مشاهده کنید به این طریق میتونیم از Lazy box استفاده کنیم.



```
1 var lazyBox = await Hive.openLazyBox('hugePeopleBox');
2 //add
3 await lazyBox.put('name', 'Amir')
4 //read
5 String name = await lazyBox.get('name');
```

ما برای دسترسی به محتویات داخل جعبه های معمولی نیاز به استفاده از مفهوم await نداریم ولی در Lazy box باید استفاده کنیم به خاطر اینکه داده ها و محتوای ما در حال حاضر در مموری نیستند و فقط با استفاده از کلیدها قابل دسترسی هستند.

. ذخیره کردن داده های حساس با استفاده از Encrypted box

شما ممکنه بخواید داده حساسی از کاربر رو داخل دیتابیستون ذخیره کنید که به راحتی قابل دسترسی نباشه اینجا هست که مفهوم Encrypted Box به ما کمک میکنه که این کارو انجام بدیم.

دیتابیس Hive از مفهوم رمزگذاری AES-۲۵۶ در خارج از Box یا جعبه پشتیبانی میکنه همراه با یک تابع کمکی برای تولید یک کلید رمزگذاری با استفاده از الگوریتم Fortuna.

برای ذخیره کردن ایمن کلید رمزگذاری شده بهتره از پکیج flutter_secure_storage استفاده کنید.

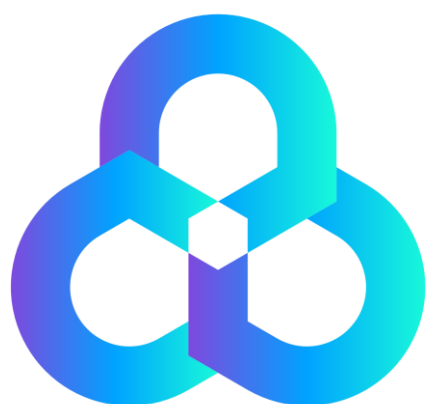
اینجا یک مثال از ساخت و بازکردن یک Encrypted Box رو مشاهده میکنید.

```
1  const secureStorage = FlutterSecureStorage();
2  final encryprionKey = await secureStorage.read(key: 'key');
3
4  if (encryprionKey == null) {
5    final key = Hive.generateSecureKey();
6    await secureStorage.write(
7      key: 'key',
8      value: base64UrlEncode(key),
9    );
10 }
11
12 final key = await secureStorage.read(key: 'key');
13 final encryptionKey = base64Url.decode(key!);
14 print('Encryption key: $encryptionKey');
15
16 await Hive.openBox(
17   'securedBox',
18   encryptionCipher: HiveAesCipher(encryptionKey),
19 );
```

سُورَةُ هٰكِيْمٍ طَلَّهِ! زَسْرِيَالِ ثَبَا! وَبَسْ! حَفْطِ وَطْ نَبَلْ قِيَهْ رَقْنِ لَنْ ثَرْ وَبِشَالِشْ ثَقْبِيْ گَهْ ر/



```
1 final encryptedBox = Hive.box('securedBox');
2
3 _getData() {
4   setState(() {
5     data = encryptedBox.get('secret');
6   });
7   log('Fetched data');
8 }
9
10 _putData() async {
11   await encryptedBox.put('secret', 'Test secret key');
12   log('Stored data');
13 }
```



فصل سوم : ذخیره دیتا های جزئی و غیر آبجکتی

. Initialize

برای راه اندازی دیتابیس مثل [دفعه قبل](#) به صورت زیر عمل میکنیم.

```
main.dart

1 import 'package:hive_flutter/adapters.dart';
2 import 'package:path_provider/path_provider.dart';
3
4 void main() async {
5   WidgetsFlutterBinding.ensureInitialized();
6   final directory = await getApplicationDocumentsDirectory();
7   Hive.init(directory.path);
8   runApp(const MyApp());
9 }
```

البته اگر میخوايد پروژه رو روی پلتفرم وب اجرا کنید باید از دستور [initFlutter](#) استفاده کنید.

الان دیتابیس ما آماده شده و میتوانیم دیتا های غیر آبجکتی رو درون دیتابیس به صورت دائمی ذخیره کنیم.

برای ذخیره دیتا های غیر آبجکتی حتما باید دیتا هارو به صورت key-value ذخیره کنیم.

خب الان که دیتابیس آماده شده موافقید بریم و عملیات CRUD رو انجام بدیم.



Create .

اولین چیزی که در دیتابیس با اون سروکار داریم ذخیره اطلاعات است. در کد زیر یک مثال ساده ای از ذخیره اطلاعات در دیتابیس رو مشاهده میکنید.



```
1 final String boxName = 'user_box';
2
3 void save_data() async {
4   var box = await Hive.openBox(boxName);
5   box.put('firstName', 'Amir');
6   box.put('email', 'ga7089036@gmail.com');
7   box.put('password', 'amir1235');
8
9   box.close();
10  print('Saved Data');
11 }
```

در لاین ۱ اومدیم اسم باکسمون رو داخل یک متغیری از جنس String ریختیم تا همجا از این نام استفاده کنیم، زیرا اگه به اشتباه اسم باکس غلط املایی داشته باشه دیتا ها هرگز از دیتابیس خونده نمیشن. پس حتما از یک نام استفاده کنید.

بعد در لاین ۳ اومدیم یه متد ساختیم که هر موقع فراخوانی شد دیتا های ما ذخیره بشن. (یادتون نره حتما باید async باشه متدمون).

در لاین ۴ اومدیم و یه متغیر ساختیم و داخل یه box باز کردیم ، نام این باکس هم متغیری که بالا در لاین ۱ تعریف کردیم گذاشتیم.

در لاین های ۵ تا ۷ اومدیم و کلید اسم ، ایمیل و پسورد را ذخیره کردیم که به ترتیب مقادیری برای مثال درونشون ذخیره کردیم.

در لاین ۹ حتما حتما باکس رو close کنید، اگه این کار رو نکنید به مرور زمان پروژه شما مقدار زیادی از رم گوشی رو اشغال میکنه و پرفورمنس اپ پایین میاد.

در لاین ۱۰ هم اگه که اروری ندهد کلمه Saved را چاپ میکند و این به این معنی است که دیتا ها با موفقیت ذخیره شده است.

بعد از ذخیره اطلاعات ما نیاز داریم که به اطلاعات ذخیره شده دسترسی داشته باشیم و در پروژه خود از آن استفاده کنیم.

در کد زیر یک مثال ساده ای از خواندن اطلاعات در دیتابیس رو مشاهده میکنید.



```

1  final String boxName = 'user_box';
2
3  void load_data() async {
4    var box = await Hive.openBox(boxName);
5    String name = box.get('firstName');
6    String email = box.get('email');
7    String password = box.get('password');
8
9    box.close();
10   print('Load Data = ${name} - ${email} - ${password}');
11 }

```

در لاین ۱ اومدیم اسم باکسمون رو داخل یک متغیری از جنس String ریختیم تا همه جا از این نام استفاده کنیم، این نام باید همونی باشه که موقع ذخیره اطلاعات ازش استفاده کردیم.

بعد در لاین ۳ اومدیم یه متد ساختیم که هر موقع فراخوانی شد دیتا های ما رو از دیتابیس بخونه. (یادتون نره حتما باید async باشه متدمون)

در لاین ۴ اومدیم و یه متغیر ساختیم و داخل یه box باز کردیم ، نام این باکس هم متغیری که بالا در لاین ۱ تعریف کردیم گذاشتیم.

در لاین های ۵ تا ۷ اومدیم سه تا متغیر از جنس String تعریف کردیم و متد get تونستیم اطلاعاتی که ذخیره شده بودن رو توی متغیر ها بریزیم.

در لاین ۹ حتما حتما باکس رو close کنید، اگه این کار رو نکنید به مرور زمان پروژه شما مقدار زیادی از رم گوشی رو اشغال میکنه و پرفورمنس اپ پایین میاد.

در لاین ۱۰ هم اگه که اروری ندهد کلمه Load Data را همراه با دیتا های خونده شده چاپ میکند و این به این معنی است که دیتا ها با موفقیت خونده شده و با موفقیت نمایش داده میشن.

Update and Delete .

خب بعد از ذخیره و خواندن اطلاعات شاید نیاز باشه اون هارو ویرایش یا حتی حذف کنیم. برای این کار میتونید از کد های زیر استفاده کنید.

```
main.dart

1 final String boxName = 'user_box';
2
3 void update_data() async {
4     var box = await Hive.openBox(boxName);
5
6     box.put('firstName', 'Seyyed Amir Nima Ghaebi');
7     box.put('password', '1234567890');
8
9     box.close();
10    print('Updata Data');
11 }
```

برای آپدیت دیتا ها باید دوباره از متد put استفاده کنیم. اگه کلید ما از قبل وجود داشته باشد مثل کلید `firstName` متد put میاد و مقدار اون رو آپدیت میکنه. پس فرقی نداری میتونید از متد [save_dada](#) هم استفاده کنید برای حذف دیتا هم میتونید از کد های زیر استفاده کنید. در پایین تمام کلید هارو پاک کردیم شما میتونید هر کلیدی که دوست دارید رو پاک کنید.

```
main.dart

1 final String boxName = 'user_box';
2
3 void delete_data() async {
4     var box = await Hive.openBox(boxName);
5
6     await box.delete('firstName');
7     await box.delete('email');
8     await box.delete('password');
9
10    box.close();
11    print('Delete Data');
12 }
```

برای حذف دیتا هامون کافیه متد delete رو صدا بزنیم و بهش کلیدی که میخوایم حذف بشه رو بدیم.

. نکاتی درباره ذخیره دیتا های غیر آبجکتی

. از نکته های دیتابیس Hive نوع داده ای ذخیره سازی هستش.
که خوشبختانه این دیتابیس هیچ محدودیتی نداره و از ما یک متغیر dynamic میگیره.
این به این معنی که ما میتونیم هر متغیری با هم نوعی که خواستیم رو در دیتابیس ذخیره کنیم.
در زیر یک مثال کامل از ذخیره سازی همه متغیر هارو مشاهده میکنید:

```
main.dart

1 final String boxName = 'user_box';
2
3 void save_data() async {
4   var box = await Hive.openBox(boxName);
5
6   box.put('int', 123);
7   box.put('double', 10.23);
8   box.put('string', 'String');
9   box.put('bool', true);
10  box.put('list', ['A', 'B', 'C', 'D', 'E', 'F', 'G']);
11  box.put('map', {'Key': 'Value', 'Key2': 'Value2', 'Key3': 'Value3'});
12
13  box.close();
14  print('Saved Data');
15 }
```

در مثال بالا میبینید که ما تقریباً تمام متغیر هارو درون دیتابیس ذخیره کردیم.
شما میتونید از متغیر هایی با نوع DateTime و Uint8List هم در دیتابیس ذخیره کنید.
. نکته بعدی اینکه برای اینکه بخواید دیتا هارو کاربر وارد کنه به راحتی کافیه که کنترلر ویجت TextField رو به متد put بدید.

```
main.dart

1 final String boxName = 'user_box';
2
3 void save_data() async {
4   var box = await Hive.openBox(boxName);
5
6   box.put('firstName', _firstNameController.text);
7   box.put('email', _emailController.text);
8   box.put('password', _passwordController.text);
9
10  box.close();
11  print('Saved Data');
12 }
```

. نکته بعدی اینکه احتمال داره شما دیتا هایی که ذخیره کردید رو با متد load_data بخونید.
بعد دیتا هارو حذف کنید.

و دوباره با متد load_data دیتا هارو بخونید.

خب چون قبلا دیتا هارو حذف کردیم پروژه به ارور برخورد میکنه.
چرا؟ چون رفته تو دیتابیس و کلیدی پیدا نکرده و به ما Null برمیگردونه.
برای حل این ارور دو راه حل وجود داره که در ادامه با هم یاد میگیریم.

✓ راه حل اول :

شما میتونید با عملگر ?? به پروژه بگید که اگر متد get مقدار Null رو برگردوند بیا و یه متنی نمایش بده.

```
main.dart

1 final String boxName = 'user_box';
2
3 void load_data() async {
4   var box = await Hive.openBox(boxName);
5
6   _firstNameController.text = box.get('firstName') ?? 'Not found';
7   _emailController.text = box.get('email') ?? 'Not found';
8   _passwordController.text = box.get('password') ?? 'Not found';
9
10  box.close();
11  setState(() {});
12  print('Load Data');
13 }
```

به کد های بالا نگاه کنید.

در لاین های ۶ تا ۸ اومدیم و گفتیم که متن ویجت TextField من مساوی باشه با مقدار کلید firstname در دیتابیس .

بعد با عملگر ?? گفتیم که اگر دیتا ای که متد get به ما برمیگردونه null بود؟ بیاد و متن ویجت TextField من رو مساوی با متن Not Found بشه.

و در لاین ۱۱ اومدیم متد setState رو فراخوانی کردیم تا تمام تغییرات ما اعمال بشه.

✓ راه حل دوم :

راه حل دوم راحت تر هستش و پیشنهاد میکنم از این راه حل استفاده کنید.

در متد get یک پارامتری وجود داره تحت عنوان defaultValue.

پارامتر defaultValue چیکار میکنه؟ میاد از ما یه متن میگیره و اگر دیتایی که برمیگردونه null باشه میاد و متنی که ما بهش داده بودیم رو خودکار برمیگردونه.

برای درک راحت تر به مثال پایین نگاه کنید.

```
1 final String boxName = 'user_box';
2
3 void load_data() async {
4   var box = await Hive.openBox(boxName);
5
6   _firstNameController.text = box.get('firstName', defaultValue: 'Amir');
7   _emailController.text = box.get('email', defaultValue: '');
8   _passwordController.text = box.get('password', defaultValue: '');
9
10  box.close();
11  setState(() {});
12  print('Load Data');
13 }
```

همینطور که میبینید در متد get اومدیم از پارامتر defaultValue استفاده کردیم.

در لاین ۶ مقدار پارامتر رو مساوی با متن Amir قرار دادیم.

و در لاین های بعدی این مقدار رو برابر با یه رشته خالی گذاشتیم.

پس اگر ما دیتا هارو حذف کنیم و متد load_data رو فراخوانی کنیم.

چون دیتا ها حذف شدن متد get به ما null برمیگردونه.

پس اومدیم ما پارامتر defaultValue یه متنی بهش دادیم.

اگر دیتا ها null باشه TextField اسم مساوی با Amir میشه.

و دو TextField بعدی تغییری نمیکنن.

این به سلیقه خودتون بستگی داره که دوست داشته باشید متنی داخل TextField قرار بدید یا اون رو خالی بزارید.

و در آخر setState فراموش نشه ، بخاطر اینکه متن های TextField ها تغییر کنه.

. و نکته بعدی اینکه شما میتونید به جای استفاده از چند متد put یک بار از متد putAll استفاده کنید.

```
main.dart

1 final String boxName = 'user_box';
2
3 void save_data() async {
4   var box = await Hive.openBox(boxName);
5
6   box.putAll({'name': 'Amir', 42: 'life'});
7
8   box.close();
9   print('Saved Data');
10 }
```

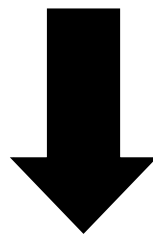
همین طور که میبینید در بالا دو پارامتر name که مقدارش Amir هست و بعدی که ۴۲ که مقدارش life هست رو مشاهده میکنید .

الان این متد حکم دو متد put را دارد.

و در نهایت شما میتوانید در کلید یک دیتا int بدید همین طور که میبینید یکی از دیتا ها کلید ۴۲ را دارد.

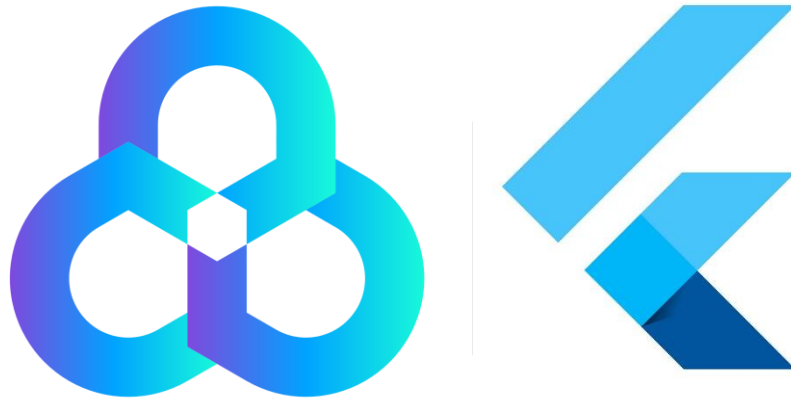
```
main.dart

1 box.put('name' , 'Amir');
2 box.put(42 , 'life');
```



```
main.dart

1 box.putAll({'name': 'Amir', 42: 'life'});
```



فصل چهارم : بررسی پیشرفته ویژگی های دیتابیس Hive

. ValueListenable در دیتابیس Hive

گاهی وقت ها نیاز هست که بیایم و به یکی از دیتا های توی دیتابیس مون دائما گوش بدیم و هر موقعی که تغییری توی اون دیتا ایجاد شد بیایم و همون لحظه توی الی هم اون تغییر و اعمال کنیم.

برای همین ما میتونم از ویژگی listenable باکس هامون استفاده کنیم تا این چالش رو پیاده سازی کنیم.

```
1 import 'package:flutter/material.dart';
2
3 import 'package:hive/hive.dart';
4 import 'package:hive_flutter/hive_flutter.dart';
5
6 void main() async {
7   await Hive.initFlutter();
8   await Hive.openBox('settings');
9   runApp(MyApp());
10 }
11
12 class MyApp extends StatelessWidget {
13   @override
14   Widget build(BuildContext context) {
15     return MaterialApp(
16       title: 'Demo Settings',
17       home: Scaffold(
18         body: ValueListenableBuilder<Box>(
19           valueListenable: Hive.box('settings').listenable(),
20           builder: (context, box, widget) {
21             return Center(
22               child: Switch(
23                 value: box.get('darkMode', defaultValue: false),
24                 onChanged: (val) {
25                   box.put('darkMode', val);
26                 },
27             ),
28           );
29         },
30       ),
31     );
32   }
33 }
34 }
```

توضیحات کد های بالا در صفحه بعد.

در کد های بالا یه مثال خیلی ساده از صفحه تنظیمات زدیم که برای مثال حالت دارک مود و لایت مود رو در دیتابیس ذخیره کردیم ، میخوام همین که کاربر حالت دارک مود رو تغییر داد توی دیتابیس هم تغییر کنه.

. در لاین های ۷ و ۸ اومدیم و دیتابیس رو Initialize کردیم و بعد اومدیم box مون رو باز کردیم، برای این box رو داخل متد main باز کردیم که دیگه نیازی نباشه هرجا خواستیم از استفاده کنیم ون رو بازش کنیم.

. در لاین ۱۷ اومدیم و Scaffold خودمون رو ساختیم.

و در قسمت Body از ویجت ValueListenableBuilder استفاده کردیم که نوع ویجت از نوع Box هست.

ویجت ValueListenableBuilder دو پارامتر داره:

. اولی valueListenable هستش ، این پارامتر میاد از ما یه مقداری میگیره و دائما به ان مقدار گوش میکنه. همین که اون مقدار تغییری کنه میاد و پارامتر دوم رو یعنی builder رو فراخوانی میکنه.

. پارامتر دومی (builder) میاد از ما یه ویجت میگیره و با استفاده از دیتا هایی که ما بهش میدیم اون ویجت رو میسازه.

یعنی هر بار که مقدار ما تغییر میکنه متد builder میاد و ویجتی که بهش دادیم رو با مقدر جدید میسازه.

خب ، همین طور که میبینید در قسمت valueListenable اومدیم و box مون رو بهش دادیم و یادتون نره حتما متد listenable باکس رو تایپ کنید.

زیرا متد listenable میاد و هر تغییری که در box ما رخ میده رو به پارامتر valueListenable میده و ویجت ValueListenableBuilder میفهمه تغییری تو box ما رخ داده پس متد builder رو فراخوانی میکنه.

. در قسمت builder اومدیم و ویجت Switch رو بهش دادیم.

در ویجت Switch گفتیم که مقدرات برابر با دیتایی که داخل دیتابیس داریم باشه، اگر دیتا null باشه اون رو false در نظر بگیر.

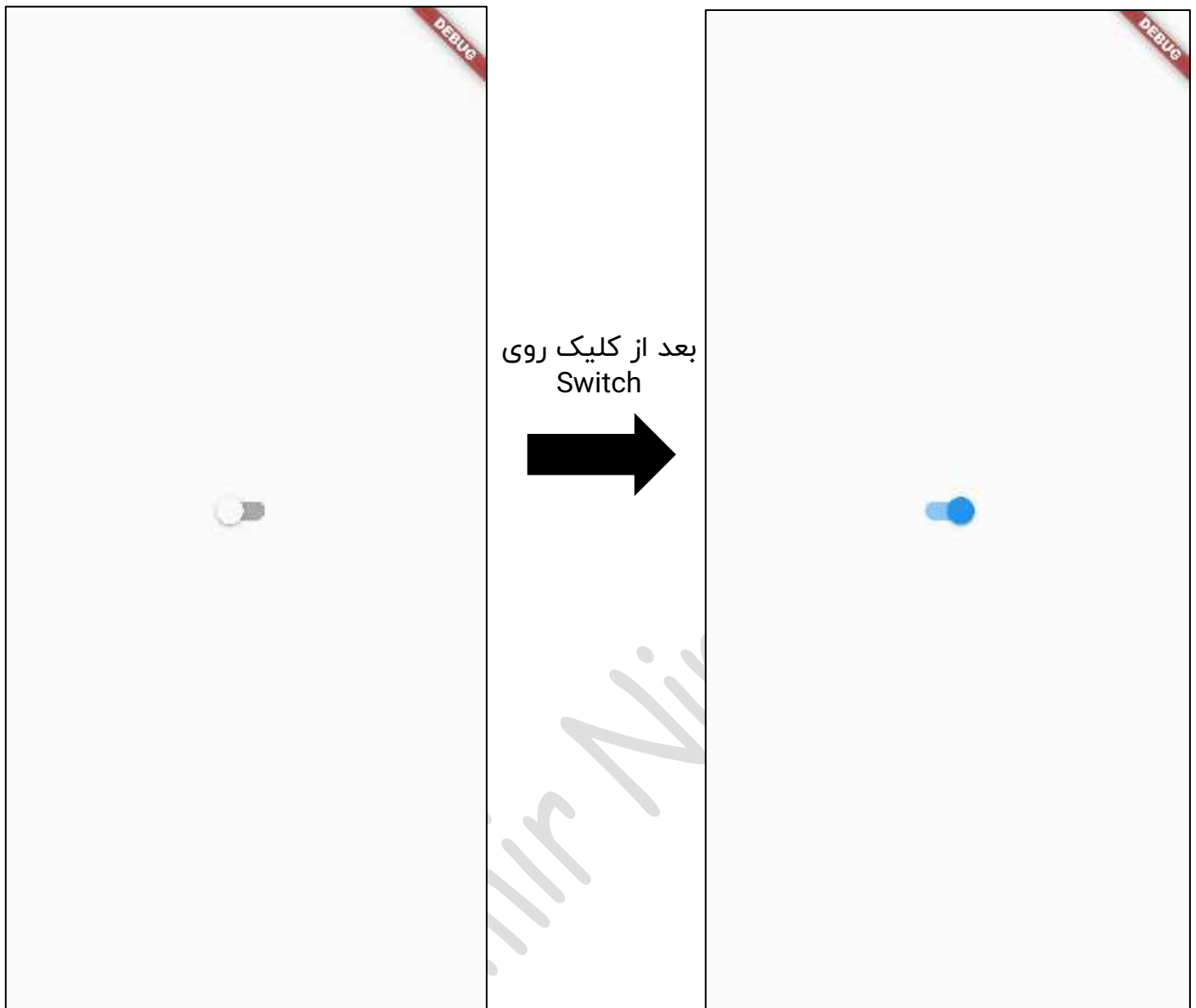
و در متد onChange هم بیا و مقدار کلید darkMode رو آپدیت کن به مقدار جدید یعنی val.

وقتی که کاربر روی ویجت Switch میزنه مقدار جدید داخل box در کلید darkMode ریخته میشه . پس تغییرات صورت میگیره .

پس در نتیجه متد builder فراخوانی میشه.

همه اینا بدون هیچ setState انجام میشه.

خروجی کار رو در صفحه بعد میتوانید مشاهده کنید.



این خروجی کار بود.

به این نکته توجه کنید که استفاده از ValueListenableBuilder خیلی روی سرعت اپ تأثیر میزازه و سرعت بیشتری داره.

ولی نباید زیاد ازش استفاده کرد، استفاده زیاد باعث کندی اپ میشه!

Add method .

پیش از این ها ما با متد put میومدیم و یه فیلد جدیدی به دیتابیس اضافه میکردیم که دارای دو پارامتر key و value بود.

یعنی برای هر فیلدی که میخواستیم به دیتابیس اضافه کنیم یه کلید میدادیم و مقداری که میخواستیم رو بهش میدادیم.

و برای دسترسی به اون مقدار میومدیم از متد get استفاده میکردیم و کلیدی که میخواستیم رو بهش میدادیم.

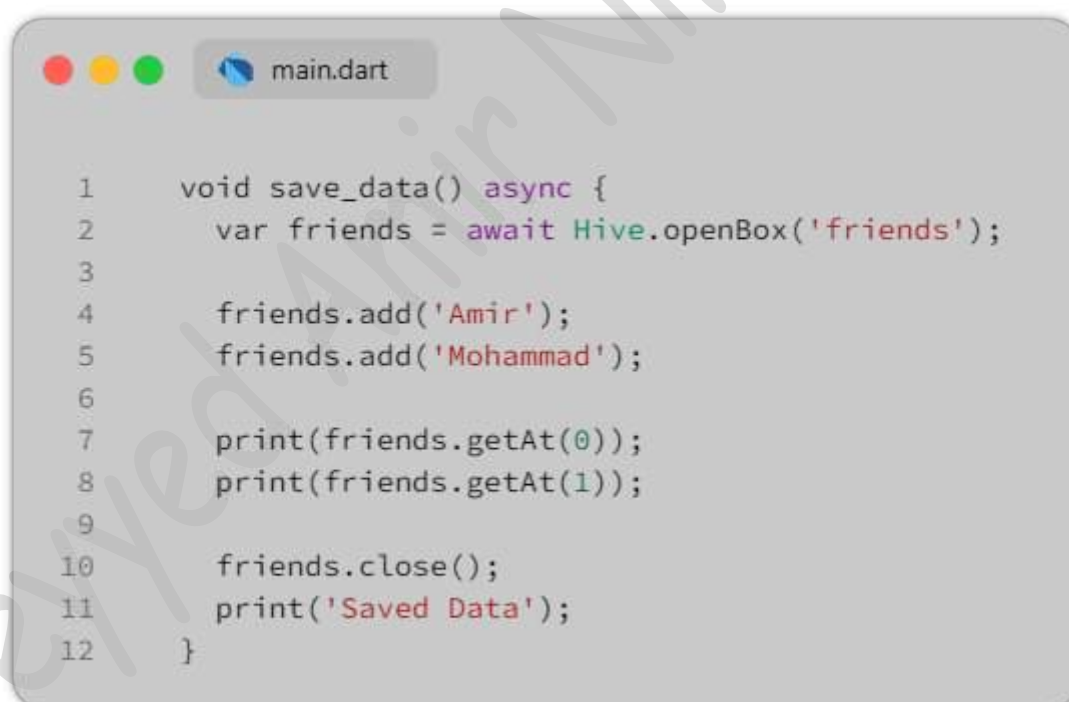
اما پکیج دیتابیس Hive یه متد دیگه ای برای اضافه کردن دیتا به دیتابیس داره.

این متد جدید که میخوایم باهم یاد بگیریم اسمش add() هست.

این متد add چه فرقی با put داره؟

متد add فقط از ما پارامتر value رو میگیره و دیگه نیازی به پارامتر key نداره.

متد add خودش مقدار key رو به صورت اتوماتیک به فیلد ها میده. در تصویر پایین میتواندی مثال ساده از این متد رو ببینید. (کاربر این متد فقط به اینجا ختم نمیشه ، جلو تر از کاربرد های دیگه متد هم استفاده میکنیم).



```
1 void save_data() async {
2   var friends = await Hive.openBox('friends');
3
4   friends.add('Amir');
5   friends.add('Mohammad');
6
7   print(friends.getAt(0));
8   print(friends.getAt(1));
9
10  friends.close();
11  print('Saved Data');
12 }
```

همین طور که میبینید در لاین ۴ و ۵ اومد دو اسم رو به دیتابیس اضافه کردیم.

این رو در نظر بگیرید در متد add ما هم key رو داریم و هم index که خود متد به صورت خودکار این دو پارامتر رو در دیتابیس میزاره.

الان فیلد Amir در دیتابیس index صفر رو داره.

و فیلد Mohammad در دیتابیس index یک رو داره.

اگر بخواید به فیلد هایی که متد add به دیتابیس اضافه کرده دسترسی داشته باشید ، میتونید از متد getAt استفاده کنید.

متد getAt از ما به index میگیره.

اگر برای مثال index صفر رو بهش بدیم میره تو دیتابیس و Amir رو برمیگردونه.

Auto increment & indices .

برای درک بهتر فرق بین key و index میتونید از کد زیر استفاده کنید.

```
1 void save_data() async {
2   var friends = await Hive.openBox('friends');
3
4   friends.add('Lisa'); // index 0, key 0
5   friends.add('Dave'); // index 1, key 1
6   friends.put(123, 'Marco'); // index 2, key 123
7   friends.add('Paul'); // index 3, key 124
8
9   print(friends.getAt(0));
10  print(friends.get(0));
11
12  print(friends.getAt(1));
13  print(friends.get(1));
14
15  print(friends.getAt(2));
16  print(friends.get(123));
17
18  print(friends.getAt(3));
19  print(friends.get(124));
20
21  friends.close();
22  print('Saved Data');
23 }
```

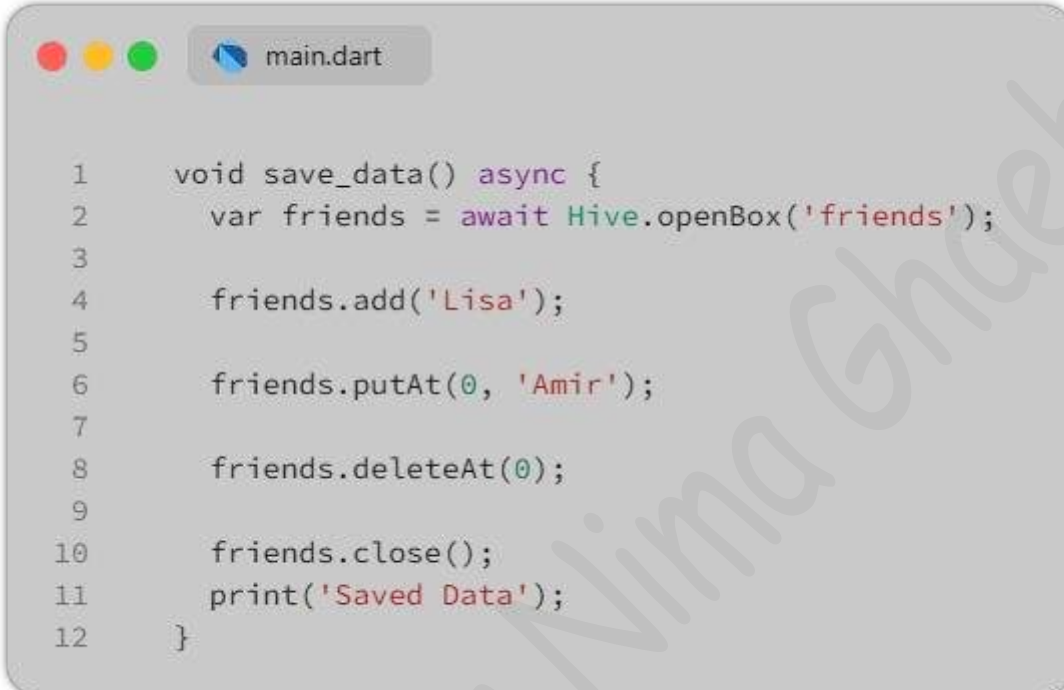
لطفا به کامنت هایی که در کد وجود داره دقت کنید.

میبینید که index در هر فیلد فقط به دونه افزایش یافته هست.

اما برای key متد add میره میبینه که آخرین key در دیتابیس ثبت شده چیه و از اون به بعد ادامه میده.

در لاین های ۹ تا ۱۹ اومدیم با متد getAt به فیلد ها دسترسی پیدا کردیم و هم با get به فیلد ها با key.

شما میتوانید با متد putAt دیتاهارو ویرایش و حتی اضافه کنید.
در متد deleteAt هم از ما index رو دریافت میکنه و اون فیلد رو حذف میکنه.



```
1 void save_data() async {  
2   var friends = await Hive.openBox('friends');  
3  
4   friends.add('Lisa');  
5  
6   friends.putAt(0, 'Amir');  
7  
8   friends.deleteAt(0);  
9  
10  friends.close();  
11  print('Saved Data');  
12 }
```



فصل پنجم : ذخیره آبجکت های شخصی سازی شده در دیتابیس Hive

. TypeAdapters چیست؟

در [فصل سوم](#) یاد گرفتیم که ما میتونیم در دیتابیس همه نوع متغیری بدون هیچ اروری ذخیره کنیم.

نوع متغیر	کد ذخیره سازی
Int	<code>box.put('key' , ۱۲۳);</code>
Double	<code>box.put('key' , ۸.۸۹);</code>
String	<code>box.put('key' , 'Amir');</code>
Bool	<code>box.put('key' , false);</code>
List	<code>box.put('list', ['A', 'B', 'C', 'D', 'E', 'F', 'G']);</code>
Map	<code>box.put('map', {'Key': 'Value', 'Key۲': 'Value۲', 'Key۳': 'Value۳'});</code>
Datetime	<code>var now = new DateTime.now(); box.put('key' , now);</code>
Uint8List	<code>Uint8List bytes = ---; box.put('key' , bytes);</code>

جدول بالا تمام متغیر هایی که دیتابیس پشتیبانی میکنه رو نشون میده.

ولی گاهی وقت ها ما به متغیری نیاز داریم غیر از اینا

برای مثال من میخوام کتاب هایی که دارم رو توی دیتابیس وارد کنم و پروژه کتابخانه آفلاین رو برای مثال بسازم.

خب من که نمیتونم از متغیر های بالا برای ذخیره کتابم استفاده کنم!

پس باید چیکار کنم؟

دیتابیس هایو میتونه از ویژگی TypeAdapters استفاده کنه تا این کارو انجام بده.

TypeAdapters چی هستش؟

به زبان ساده TypeAdapters در واقع یک کلاسی هست که ما اون رو میسازیم و شخصی سازی میکنیم و از اون کلاس به عنوان یه متغیر استفاده میکنیم برای ذخیره دیتا هایی که به طور پیشفرض در زبان دارت نیستن.

. تعریف جدول در دیتابیس Hive

اگر قبلا با دیتابیس های دیگه ای مثل SQL Server کار کرده باشید ، حتما با جدول ها آشنا هستید. در دیتابیس ها ما باید با استفاده از جدول هایی که میسازیم دیتا هارو ذخیره کنیم. برای ذخیره کتاب ها ما به جدول زیر نیاز داریم:

Books				
ID	Name	Writer	Publication Date	Price
۰				
۱				
۲				
۳				
۴				
۵				
...

در جدول بالا هر کتاب دارای یک نام ، نویسنده ، زمان انتشار ، قیمت هستش.

پس هر کتابی که بخوایم به این جدول اضافه کنیم باید این چهار تا فیلد بالا رو حتما داشته باشه. اما یه مشکلی هست!

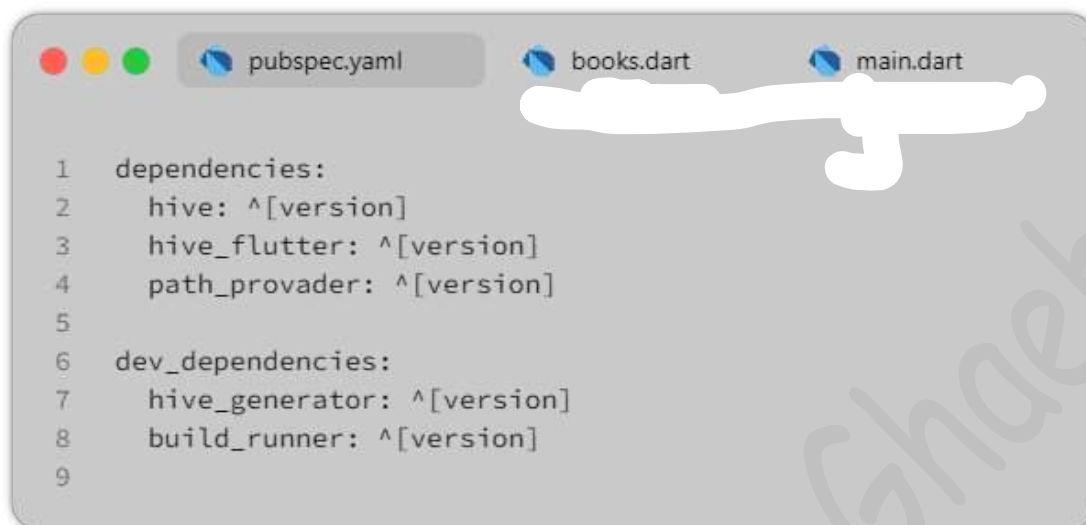
دیتابیس Hive یک دیتابیس NoSql هستش و هیچ جدولی نداره.

راه حل چیه؟

ما باید این جدول بالارو با کد های فلاتری بسازیم ، در اصل کلاس TypeAdapter همون جدول ما هستش. پس بدون وقت تلف کردن بریم برای ساخت کلاس.

. ساخت کلاس TypeAdapter برای ذخیره کتاب

خب برای شروع پروژه ، باید پکیج های زیر رو در فایل **pubspec.yaml** اضافه کنید



```
1 dependencies:
2   hive: ^[version]
3   hive_flutter: ^[version]
4   path_provider: ^[version]
5
6 dev_dependencies:
7   hive_generator: ^[version]
8   build_runner: ^[version]
9
```

قبل از اینکه بریم سراغ پروژه درباره پکیج های بالا نیز یه توضیح بدم.

. پکیج hive که اصلی ترین چیزی هست که نیاز داریم که از اسمش که معلومه خود دیتابیس مون هستش.

. پکیج hive_flutter برای راحتی کار با دیتابیس در فلاتر هستش.

. از پکیج path_provider برای شناسایی آدرس پوشه پروژه هستش ، هر برنامه ای در گوشی کاربر یه پوشه اختصاصی برای خودش داره که در گوشی های و پلتفرم های مختلف این آدرس پوشه متغییر هست ما با پکیج path_provider میایم و آدرس این پوشه های اپ رو پیدا میکنیم. این پوشه ها مخصوص خود اپ هست و نه کاربر و نه اپ های دیگه مثل گالری به این پوشه هیچ دسترسی ندارن.

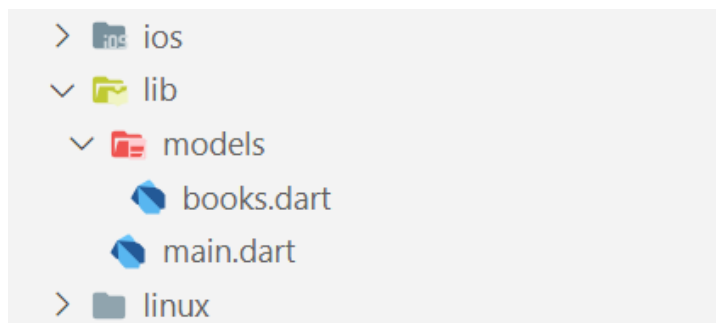
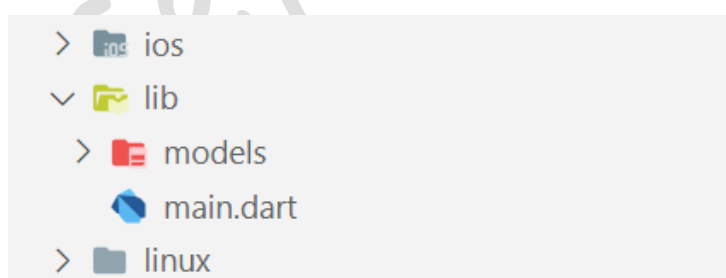
. از پکیج های hive_generator و build_runner برای ساخت کلاس های شخصی سازی شده باری ذخیره اطلاعات استفاده میکنیم که در این فصل و فصل بعدی با این کلاس ها آشنا خواهید شد.

و سپس دستور flutter pub get را در ترمینال اجرا کنید.

حالا بریم سراغ پوشه بندی پروژه مون که تنها یه پوشه اضافه میشه (:

من در پوشه lib اومدم و یه پوشه به اسم models ساختم.

داخل پوشه models یه فایل به نام book.dart ساختم ، درواقع در این فایل میخوایم کلاس مون رو بسازیم.



در صفحه بعد میایم و کد های داخل فایل books.dart داریم رو توضیح میدیم.

در فایل book.dart کد های زیر رو نوشتیم:

```
pubspec.yaml  books.dart  main.dart

1  import 'package:hive/hive.dart';
2
3  part 'books.g.dart';
4
5  @HiveType(typeId: 1)
6  class Books extends HiveObject {
7    @HiveField(0)
8    String? name;
9
10   @HiveField(1)
11   String? writer;
12
13   @HiveField(2)
14   String? publicationDate;
15
16   @HiveField(3)
17   String? price;
18
19   Books({
20     required this.name,
21     required this.writer,
22     required this.publicationDate,
23     required this.price,
24   });
25 }
26
```

. در لاین ۱ که اومدیم پکیج هایو رو اضافه کردیم.

. کدی که در لاین ۳ نوشتیم رو در ادامه توضیح میدم.

. در لاین ۶ اومدیم کلاس مون رو به اسم Books ساختیم و به اون گفتیم که از کلاس HiveObject ارث بری بکنه. که حتما لازمه برای اینکه بتونیم ازش استفاده کنیم.

. در لاین ۵ و دقیقا بالای کلاس مون اومدیم از عبارت HiveType استفاده کردیم و این دقیقا آیدی جدول مون رو تعیین میکنه. برای مثال اگه من یه جدول برای سبد خرید بخوام اضافه کنم آیدی باید بشه ۲ مگر نه با جدول اول به هم میخوره.

. داخل کلاس از عبارت HiveField استفاده کردیم ، این دقیقا همون فیلد هایی که تو جدول بالا داشتیم هست یعنی همون Name , Writer , Publication Date , Price توی جدول.

. در کلاس نیازی به فیلد ID نداریم.

خب قسمت فیلد ها هم تقریبا کامل کردیم.

. برای مثال در لاین ۱۶ و ۱۷ اومدیم از عبارت HiveField استفاده کردیم تا Hive بدونه این متغیر یکی از فیلد های جدول مون هست. یادتون باشه id فیلد ها تکراری نباشه.

. و در لاین ۱۷ اومدیم یه رشته که با عملگر ? ، Nullable کردیم چون ممکنه null باشه.

. و در نهایت اسمش رو داریم چون فیلد برای قیمت اسشو گذاشتیم Price ولی شما هر اسمی که دوست دارید رو بزارید.

. در لاین های ۱۹ تا ۲۴ اومدیم و سازنده کلاس رو تعریف کردیم ، برای اینکه بتونیم به فیلد های جدول مون دسترسی داشته باشیم.

پس کلاس مون آمادهست!

. اما بریم سراغ لاین ۳ ، الان این کلاسی که ما ساختیم رو پکیج Hive نمیشناسه ! پس باید کاری کنیم که پکیج Hive این کلاسی که ساختیم رو به عنوان یه TypeAdapter قبول کنه و ما بتونیم ازش استفاده کنم.

. خب برای این کار از پکیج build_runner استفاده میکنیم تا کلاس مارو آماده کنه.

در لاین ۳ اومدیم و این کارو انجام دادیم.

از عبارت part استفاده کردیم و آدرس فایلی که باید بسازه رو بهش دادیم .

. خب قطعا ارور میده چون آدرسی که میدیم اصلا وجود نداره ولی نگران نباشید ارور در ادامه کار حل میشه. آدرس فایل رو چی بدیم ؟ باید آدرس همین فایل book.dart رو بدیدم و در آخر اسم فایل قبل از فرمت فایل یه g.dart بزاریم.



```
7
8
9  part 'books.g.dart';
10
11 @HiveType(typeId: 1)
```

همین طور که در تصویر بالا میبینید اسم ها یکی هستن و تنها فرقی اضافه شدم g. به اول فرمت فایل هستش.

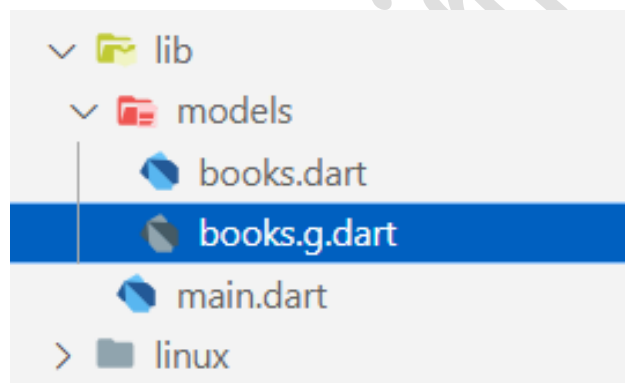
تا اینجا کار خیلی خوب پیش رفتیم ، در ادامه کار میایم با استفاده از پکیج build_runner فایل جدید رو میسازیم.

برای ساخت فایل جدید (همین فایل books.g.dart) ، ترمینال IDE که استفاده میکنید رو باز کنید.
برای ساخت فایل دستور `flutter packages pub run build_runner build` رو در ترمینال اجرا کنید.
منتظر بمونید تا فایل رو ایجاد کنه.
خروجی کار به این شکل میشه:

```
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell + - [ ] [X] ... ^ X

PS C:\Users\Amir\Documents\project\project> flutter packages pub run build_runner build
Deprecated. Use `dart run` instead.
[INFO] Generating build script completed, took 1.8s
[INFO] Reading cached asset graph completed, took 257ms
[INFO] Checking for updates since last build completed, took 19.9s
[INFO] Running build completed, took 19.0s
[INFO] Caching finalized dependency graph completed, took 265ms
[INFO] Succeeded after 19.3s with 35 outputs (35 actions)
PS C:\Users\Amir\Documents\project\project>
```

در صورتی که هیچ اروری نده مثل خروجی بالا به صورت خودکار کنار همین فایل یه فایل جدیدی به اسم books.g.dart ساخته.



تا اینجا کار تقریباً تمام کارای دیتابیس انجام شد ، توی فایل جدید یه همچین کد هایی رو میبینید.

```
lib > models > books.g.dart
1  // GENERATED CODE - DO NOT MODIFY BY HAND
2
3  part of 'books.dart';
4
5  // *****
6  // TypeAdapterGenerator
7  // *****
8
9  class BooksAdapter extends TypeAdapter<Books> {
10   @override
11   final int typeId = 1;
12
13   @override
```

خب ما با این فایل هیچ کاری نداریم ، همین طور که در لاین ۱ میبینید خود هایو هم گفته که به هیچ وجه به کد های این فایل دست نزنید ، تنها چیزی میخوایم، اسم کلاس هستش.
فقط اسم کلاس BookAdapter کپی کنید و به مرحله بعد برید.

Register Adapter .

خب تا الان تونستیم کلاس خودمون رو بسازیم و Adapter هم با کمک پکیج build_runner ایجاد کردیم. الان باید دیتابیس مون رو Initialize کنیم و Adapter که کپی کردیم رو ثبت کنیم ، تا بتونیم ازش استفاده کنیم.

```
main.dart

1 import 'package:flutter/material.dart';
2 import 'package:hive_flutter/adapters.dart';
3 import 'package:path_provider/path_provider.dart';
4 import 'package:project/models/books.dart';
5
6 void main() async {
7   WidgetsFlutterBinding.ensureInitialized();
8   final directory = await getApplicationDocumentsDirectory();
9   Hive.init(directory.path);
10
11   Hive.registerAdapter(BooksAdapter());
12   await Hive.openBox<Books>('book_box');
13
14   runApp(const MyApp());
15 }
```

تنها چیزی که اضافه شده در لاین ۱۱ هست که اومدیم از متد registerAdapter برای ثبت آداپتور که ساختیم استفاده کردیم.

در لاین ۱۲ هم اومدیم box مون رو ساختیم تا دیگه نیازی نباشه هی بازش کنیم و هی ببندیم.

و همیشه یادتون باشه که برای این نوع Box هایی که دیتاشون شخصی سازی شده و فرق میکنه حتما با علامت <> بگید باید از چه نوعی باشه. باید اسم کلاسی که تو فایل book.dart ساختید رو بدید.

التبه اگر میخواید از دیتابیس در پلتفرم های غیر از اندروید و IOS استفاده کنید باید به صورت زیر هایو رو Initialize کنید.

```
main.dart

1 import 'package:flutter/material.dart';
2 import 'package:hive_flutter/adapters.dart';
3 import 'package:project/models/books.dart';
4
5 void main() async {
6   WidgetsFlutterBinding.ensureInitialized();
7   Hive.initFlutter();
8
9   Hive.registerAdapter(BooksAdapter());
10  await Hive.openBox<Books>('book_box');
11
12  runApp(const MyApp());
13 }
```



```

40 class _HomePageState extends State<HomePage> {
41   @override
42   Widget build(BuildContext context) {
43     final box = Hive.box<Books>(box_Name);
44     return Scaffold(
45       appBar: AppBar(
46         title: const Text('Library'),
47         centerTitle: true,
48       ), // AppBar
49       floatingActionButton: FloatingActionButton(
50         onPressed: () {
51           Navigator.of(context).push(
52             MaterialPageRoute(
53               builder: (context) => AddPage(index: null),
54             ), // MaterialPageRoute
55           );
56         },
57         child: const Icon(
58           Icons.add,
59           color: Colors.white,
60         ), // Icon
61       ), // FloatingActionButton
62       body: box.values.isEmpty

```

. خب کد های بالا کاملاً مشخصه برای چیزای جدیدی که اضافه شده توضیحات زیر رو بخونید.

. در کد بالا و در لاین ۴۳ اومدیم و یه Box ساختیم .

. در لاین ۴۹ تا ۶۱ اومدیم یه floatingActionButton ساختیم که در صورتی که کلیک بشه میشه به صفحه AddPage که در ادامه باهم میسازیمش.

به لاین ۵۳ دقت کنید، اومدیم و index به صفحه add فرستادیم که در ادامه توضیح میدم برای چی این کارو کردیم.

این از این بریم سراغ body و تکمیلش کنیم.

```

62         body: box.values.isEmpty
63         ? const Center(child: Text('You have no book!'))
64         : ListView.builder(
65             itemCount: box.values.length,
66             itemBuilder: (context, index) {
67                 final Books books = box.values.toList()[index];
68                 return ListTile(
69                     onTap: () {
70                         Navigator.of(context).push(
71                             MaterialPageRoute(
72                                 builder: (context) => AddPage(index: index),
73                                 ), // MaterialPageRoute
74                     );
75                 },
76                 title: Text(books.name.toString()),
77                 subtitle: Text(books.writer.toString()),
78                 leading: CircleAvatar(
79                     backgroundColor: Colors.grey.shade200,
80                     child: Center(
81                         child: Text(
82                             index.toString(),
83                             style: const TextStyle(fontSize: 20),
84                         ), // Text
85                     ), // Center
86                 ), // CircleAvatar
87                 trailing: IconButton(
88                     onPressed: () {
89                         Hive.box<Books>(box_Name).deleteAt(index);
90                         setState(() {});
91                     },
92                     icon: const Icon(
93                         Icons.delete,
94                         color: Colors.grey,
95                     ), // Icon
96                 ), // IconButton
97             ); // ListTile
98         },
99     ), // ListView.builder

```

. در لاین ۶۲ اومدیم و اون باکس من که در لاین ۴۳ ساختیم مقداری داخلش هست یا نه.

اگر مقداری نباشه داخل Box میایم و متن شما کتابی ندارید رو وسط صفحه نشون میدیم.

. ولی اگه در box مون دیتایی وجود داشته باشه میایم و یه ListView.builder میسازیم.

. در لاین ۶۵ باید طول دیتا های باکس یا همون کتاب هامون رو به listView بدیم که اون با توجه به تعداد کتاب ها بیاد و لیست ایجاد کنه.

. از لاین ۶۶ به بعد باید بیایم و لیستمون رو بسازیم.

. در لاین ۶۷ اومدیم یه متغیر از نوع Books با نام books ساختیم و درونش اومدیم یه کتاب قرار دادیم که این کتاب index برای با index لیست ویو داره ، یعنی اگه listView بخواد ردیف اول رو بسازه index برابر با ۰ هست ، و میره تو دیتابیس و کتاب هارو میاره و اون کتابی که index برای با ۰ هست رو میاره. و همین طور index افزایش داده میشه و دونه دونه هر کتاب به لیست اضافه میشه.

. و در لاین ۶۸ اومدیم و یه نمونه از هر کتاب رو ساختیم که در نهایت این ردیف ها کنار هم گذاشته میشه و لیست مارو میسازه.

. اومدیم و از ویجت ListTile استفاده کردیم.

. در پراپرتی onTap اومدیم و گفتیم در صورتی روی یکی از کتاب ها کلی بشه به صفحه AddPage بره و به صفحه add ما پارامتر index رو براش فرستادیم. اگر دقت کرده باشید ما در لاین ۵۳ اومدیم و پارامتر index رو براش null فرستادیم. در ادامه میگم چرا به ای پارامتر نیاز داریم.

. در لاین ۷۶ اومدیم و عنوان ویجت ListTile رو برابر با اسم کتاب قرار دادیم.

. و در قسمت subtitle اومدیم و نام نویسنده رو قرار دادیم.

. در لاین های ۷۸ تا ۸۶ اومدیم و در پراپرتی leading از ویجت CircleAvatar استفاده کردیم برای نمایش index کتاب ها ، همین کار خاصی انجام ندادیم.

. در لاین های ۸۷ تا ۹۶ اومدیم و از پراپرتی trailing استفاده کردیم و بهش یه IconButton داریم و گفتیم در صورتی کلیک بشه بیاد و کتاب مارو از دیتابیس حذف کنه.

. در لاین های ۸۹ و ۹۰ میبینید که اومدیم و کتاب رو با استفاده از متد deleteAt حذف کردیم و index بهش دادیم . متد میره داخل دیتابیس میگردد و اون کتابی index برابر با index که ما فرستادیم باشه رو حذف میکنه ، خب کتاب از دیتابیس حذف میشه ولی در خط بعدی حتما از setState استفاده کنید تا لیست ما از اول ساخته بشه ، اگر setState رو انجام ندید اون کتابی که حذف شده همچنان در لیست خواهد ماند و باید برنامه رو ببندیم و دوباره باز کنیم که کتاب از لیست ما هم حذف بشه.

خب تا الان تونستیم کتاب هارو درون لیست نمایش بدیم و هر کدوم از کتاب هایی که خواستیم رو حذف کنیم.

در مرحله بعدی میریم که به box مون کتاب های جدید اضافه کنیم و اون هارو ویرایش کنیم.


```

lib > add.dart
1  import 'package:flutter/material.dart';
2  import 'package:hive/hive.dart';
3  import 'package:project/main.dart';
4  import 'package:project/models/books.dart';
5
6  // ignore: must_be_immutable
7  class AddPage extends StatefulWidget {
8      int? index;
9
10     AddPage({required this.index, super.key});
11
12     @override
13     State<AddPage> createState() => _AddPageState();
14 }
15
16 class _AddPageState extends State<AddPage> {
17     @override
18     Widget build(BuildContext context) {
19         final _nameController = TextEditingController();
20         final _writerController = TextEditingController();
21         final _dateController = TextEditingController();
22         final _priceController = TextEditingController();
23
24         if (widget.index != null) {
25             final box = Hive.box<Books>(box_Name);
26             final Books books = box.values.toList()[widget.index!];
27
28             _nameController.text = books.name.toString();
29             _writerController.text = books.writer.toString();
30             _dateController.text = books.publicationDate.toString();
31             _priceController.text = books.price.toString();
32         }
33
34         return Scaffold(

```

اینم از کلاس AddPage بدون اتلاف وقت بریم سراغ توضیحات کد

. در لاین ۸ اومدیم و پارامتر index مون رو تعریف کردیم.

این پارامتر به چه دردی میخوره؟ ببینید ما میخوام عملیات اضافه کرد و ویرایش رو در این صفحه انجام بدیم پس میایم از این پارامتر استفاده میکنیم ، اگر مقدارش null بود میفهمیم که کاربر میخواد کتاب اضافه کنه ولی اگه مقدار index نال نبود میفهمیم قراره یه کتاب ویرایش بشه پس با index که داریم میریم داخل دیتابیس و دنبال کتابی که با این index برابره میگردیم و مشخصات کتاب رو به کاربر نشون میدیم تا کاربر هر فیلدی خواست رو ویرایش کنه.

. در لاین های ۱۹ تا ۲۲ اومدیم و چهار تا کنترلر برای textField هامون ساختیم ، یه کنترلر برای اسم کتاب ، یکی برای نویسنده ، یکی برای تاریخ انتشار و دیگری برای قیمت کتاب هستش.

. در لاین ۲۴ اومدیم و گفتیم که اگر index که دریافت کردیم null نبود پس قراره کتابی ویرایش بشه!

پس میایم و که box میسازیم و در لاین ۲۶ میایم یه متغیر از نوع Books میسازیم و میگی که برو به دیتابیس و اون کتابی که index با index من برابر هست رو بیار و بریز داخل متغیر books من.

. در لاین های ۲۸ تا ۳۱ میایم و متن TextField هامون رو برابر با مشخصات کتابی که داریم میکنیم.

. توجه کنید این کد هایی که در لاین های ۲۵ تا ۳۱ هست در صورتی اجرا میشن که index کتابی برای کلاس ارسال شده باشه در صورتی که index ما Null باشه یعنی قراره کتابی اضافه بشه پس نیازی به اجرا کد های نیست چون اصلا index نداریم پس اگر کد های اجرا بشه نمیتونه کتابی پیدا کنه چون index ما مقدارش null هست .

و در ادامه Scaffold خودمون رو تکمیل میکنیم.

```

34     return Scaffold(
35       appBar: AppBar(
36         title: const Text('Add New Book'),
37         centerTitle: true,
38         actions: [
39           IconButton(
40             onPressed: () {
41               if (widget.index == null) {
42                 Hive.box<Books>(box_Name).add(
43                   Books(
44                     name: _nameController.text,
45                     writer: _writerController.text,
46                     publicationDate: _dateController.text,
47                     price: _priceController.text,
48                   ), // Books
49                 );
50               } else {
51                 Hive.box<Books>(box_Name).putAt(
52                   widget.index!,
53                   Books(
54                     name: _nameController.text,
55                     writer: _writerController.text,
56                     publicationDate: _dateController.text,
57                     price: _priceController.text,
58                   ), // Books
59                 );
60               }
61             },
62             Navigator.pushReplacement(
63               context,
64               MaterialPageRoute(
65                 builder: (context) => const HomePage(),
66               ), // MaterialPageRoute
67             );
68           ],
69           icon: const Icon(Icons.check),
70         ), // IconButton
71       ],
72     ), // AppBar
73     body: Padding(

```

خب در Scaffold ما یه appBar داریم

در پارامتر action اومدیم و یه دکمه اضافه کردیم که آیکنون check رو داره ✓ خب میخوایم وقتی رو دکمه کلیک شد کار های زیر رو انجام بده.

اول از همه باید چک کنیم ببینم آیا قراره عملیات add انجام بشه یا put برای همین در لاین ۴۱ اومدیم و گفتیم در صورتی که index من null بود یعنی قراره کتاب add بشه ولی اگر نه کتاب قراره put بشه.

خب اگر null بود باید کتاب رو add کنیم.

در لاین ۵۱ اومدیم و box مون رو با همون اسم قبلی ساختیم و متد add رو صدا زدیم.
اومدیم و از کلاس Books یه نمونه ساختیم و اون رو به متد add دادیم.
در لاین های ۵۴ تا ۵۷ اومدیم و نام و دیگر مشخصات کتاب را دادیم.
و در نهایت کتاب ما با موفقیت به box اضافه میشه.

خب حالا نوبت put هست.

دوباره در لاین ۵۱ اومدیم و یه box با همون اسم قبلی ساختیم اگه اسم ها برابر نباشه ارور میده.
و متد putAt رو صدا زدیم.

در پارامتر اول index کتابی که باید ویرایش بشه رو بهش دادیم.
و در پارامتر دوم مثل قبلی یه نمونه از کلاس books ساختیم و مشخصات جدید کتاب رو هم بهش دادیم.
در نهایت کتاب ما با مقادیر جدید ویرایش میشه.

خب چه کتاب جدیدی اضافه کنیم چه کتابی رو ویرایش کنیم در لاین های ۶۲ تا ۶۷ اومدیم و عملیات برگشت به صفحه Home رو انجام دادیم.

اینم از این الان باهم میریم که ادامه Body رو تکمیل کنیم.

```

73 body: Padding(
74   padding: const EdgeInsets.all(13.0),
75   child: Column(
76     children: [
77       const SizedBox(
78         height: 25,
79       ), // SizedBox
80       TextField(
81         controller: _nameController,
82         decoration: const InputDecoration(hintText: 'Book Name'),
83       ), // TextField
84       const SizedBox(
85         height: 15,
86       ), // SizedBox
87       TextField(
88         controller: _writerController,
89         decoration: const InputDecoration(hintText: 'Writer'),
90       ), // TextField
91       const SizedBox(
92         height: 15,
93       ), // SizedBox
94       TextField(
95         controller: _dateController,
96         decoration: const InputDecoration(hintText: 'Date'),
97       ), // TextField
98       const SizedBox(
99         height: 15,
100      ), // SizedBox
101      TextField(
102        controller: _priceController,
103        decoration: const InputDecoration(hintText: 'Price'),
104      ), // TextField
105    ],
106  ), // Column

```

در body به جز textfield ها چیزی نداریم.

پس فقط یکی از textfield ها رو توضیح میدم.

در لاین های ۸۰ تا ۸۳ اومدیم و یه textfield تعریف کردیم.

در پارامتر اول یعنی controller اومدیم و کنترلر مربوط به هر textfield رو دادیم .

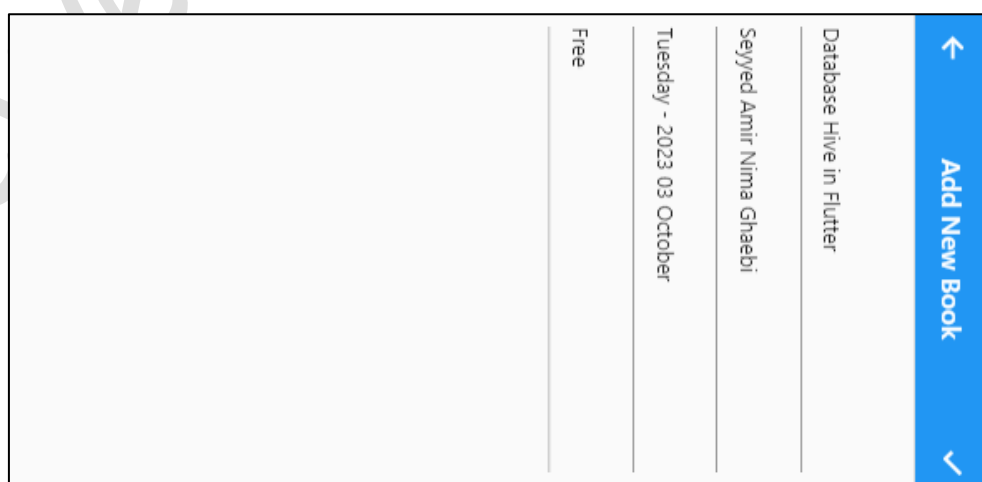
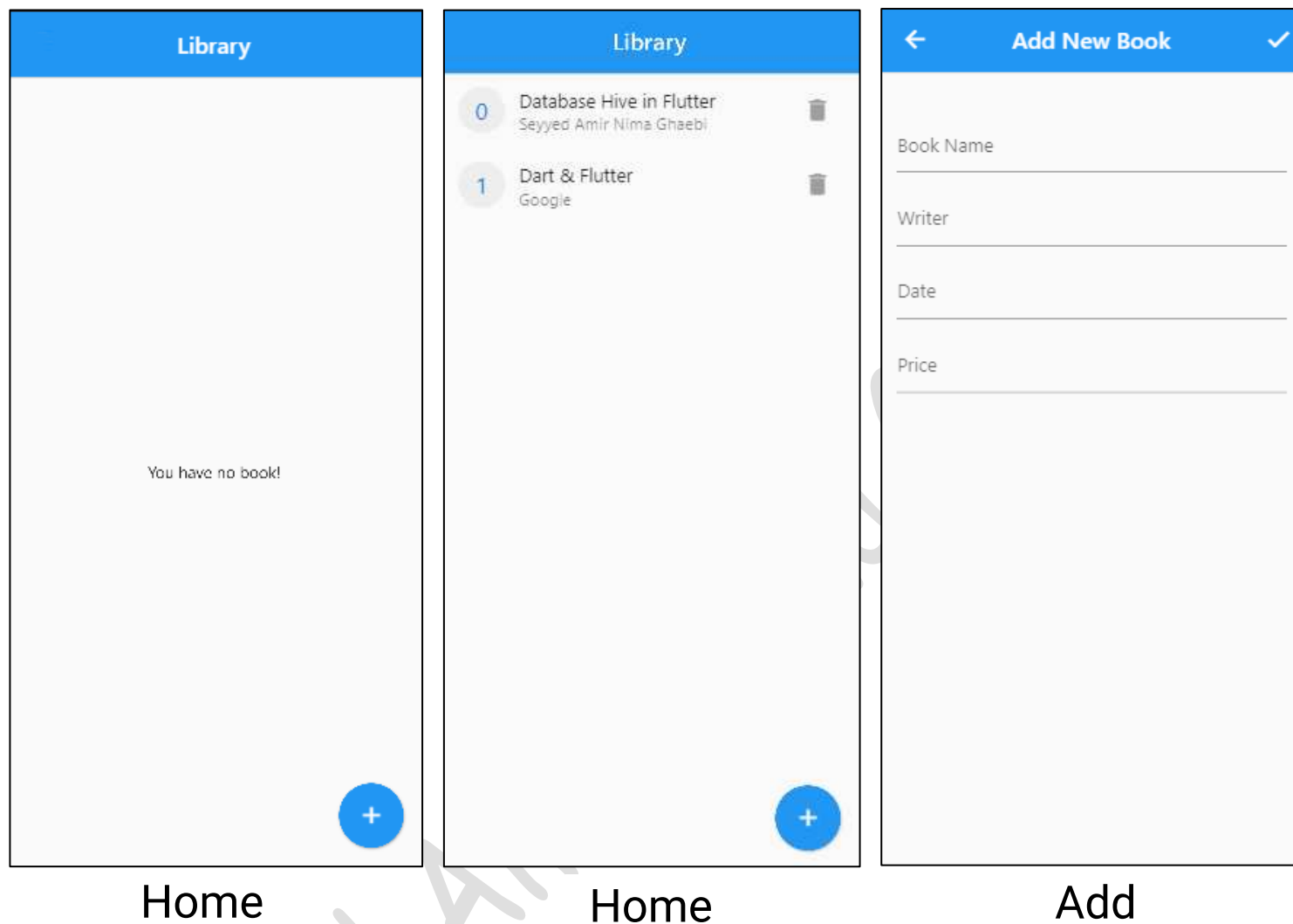
در پارامتر دوم یعنی decoration اومدیم و یه hintText به هر textfield دادیم.

همین و بین هر textfield اومدیم و با ویجت SizedBox به فاصله ای دادیم.

در صفحه بعدی میتوانید خروجی پروژه ای که نوشتیم رو ببینید.

Output .

خروجی پروژه کوچک کتابخانه به شکل زیر هست :



Edit

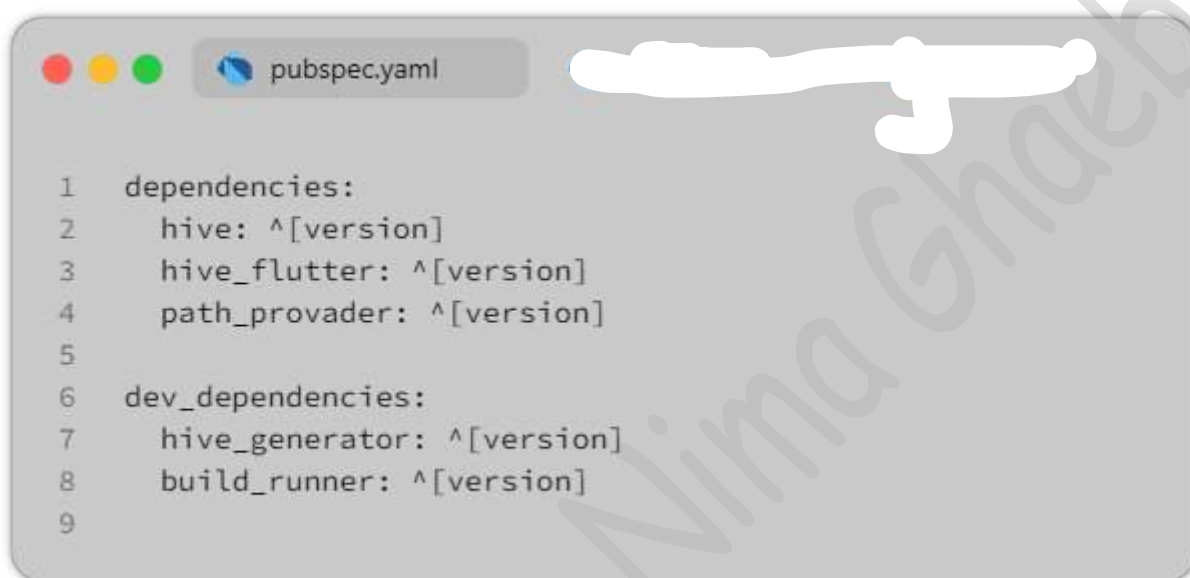


فصل ششم : ساخت پروژه Todos ساده

. شروع پروژه

دوستان برای یادگیری بهتر و مرور مطالبی که با هم یاد گرفتیم تصمیم گرفتم یه پروژه کوچک هم باهم بنویسیم تا این دیتابیس خوب رو باهم کامل یاد بگیریم.

خب برای شروع پروژه ، مثل قبل برای استفاده از دیتابیس باید پکیج Hive و دیگر پکیج هایی که لازم داریم رو به پروژه فلاتری مون در فایل **pubspec.yaml** اضافه کنید.

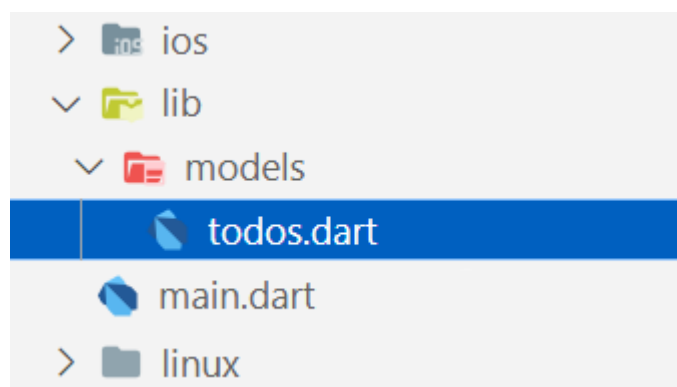
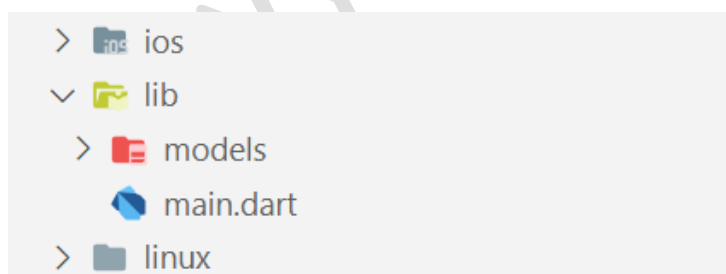


یادتون نره به جای کلمه version برید به سایت pub.dev و آخرین ورژن های پکیج های بالا رو کپی کنید و جایگزین کنید.

و سپس دستور flutter pub get را در ترمینال اجرا کنید.

در قسمت پوشه های پروژه مثل فصل پیش فقط یک پوشه models برای فایل های دیتابیس مون در پوشه lib ایجاد میکنیم.

داخل پوشه models یه فایل به نام todos.dart ساختیم .



در صفحه بعد میایم و کد های داخل فایل todos.dart داریم رو توضیح میدیم.

. ساخت کلاس Register Adapter & TypeAdapter

خب در فایل todos میایم و کدهای زیر رو تایپ میکنیم.

توجه کنید اسم todos کاملا اختیاری شما هر اسمی که دوست دارید و به پروژه تون مربوط هست رو تایپ کنید.

```
main.dart  todos.dart

1  import 'package:hive/hive.dart';
2
3  part 'todos.g.dart';
4
5  @HiveType(typeId: 1)
6  class Todos extends HiveObject {
7    @HiveField(0)
8    String? title;
9    @HiveField(1)
10   String? text;
11   @HiveField(2)
12   String? date;
13   @HiveField(3)
14   bool? isComplete;
15
16   Todos({
17     required this.title,
18     required this.text,
19     required this.date,
20     required this.isComplete,
21   });
22 }
```

. در لاین ۳ آدرس کلاسی که دو پکیج های hive_generator و build_runner برای ذخیره سازی میخواند بسازه رو بهش دادیم و قطعا ارور میده چون این فایل فعلا ساخته نشده. اسم فایل هم باید دقیقا برابر با اسم کلاس خودمون باشه یعنی اسم فایل خودمون هست todos.dart اسم فایل جدید در آدرس باید بشه todos.g.dart.

. در لاین ۵ اومدیم و آیدی جدول خودمون رو با عبارت HiveType دادیم.

. در لاین ۶ کلاس خودمون رو ساختیم باز هم اسم این کلاس اختیاری هست و کاملا به خودتون بستگی داره و بهش گفتیم که از کلاس HiveObject ارث بری کنه تا بتونیم از قابلیت های دیتابیس استفاده کنیم.

. در لاین های ۷ تا ۱۴ اومدم و فیلدهای دیتابیس رو تعریف کردیم که اول با عبارت HiveField گفتیم که میخوایم یه فیلد اضافه کنیم بلاوه آیدی و در خط بعد نوع متغیر و نام متغیر رو تعریف کردیم.

. در لاین های ۱۶ تا ۲۱ هم اومدیم و یه سازنده برای کلاسمون ساختیم تا بتونیم به فیلد های دیتابیس مون دسترسی داشته باشیم.

برای ساخت کلاس جدید که پکیج Hive اون رو بتونه بشناسه ترمینال IDE که استفاده میکنید رو باز کنید. برای ساخت فایل دستور `flutter packages pub run build_runner build` رو در ترمینال اجرا کنید.

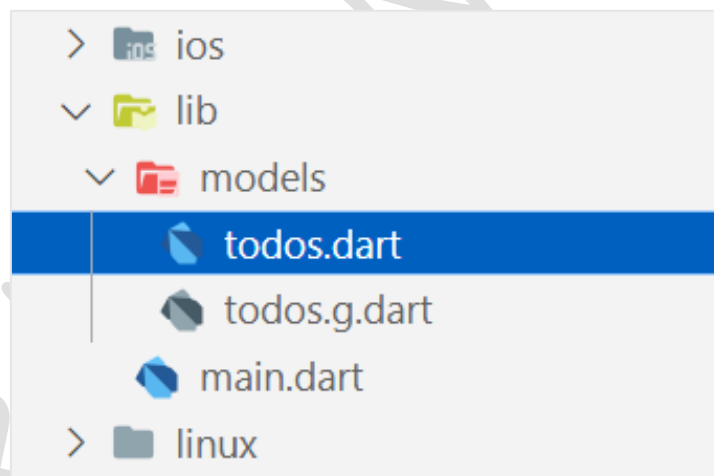
منتظر بمونید تا فایل رو ایجاد کنه.

خروجی کار به این شکل میشه:

```
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell + - [ ] [X] ... ^ X

PS C:\Users\Amir\Documents\project\project> flutter packages pub run build_runner build
Deprecated. Use `dart run` instead.
[INFO] Generating build script completed, took 1.8s
[INFO] Reading cached asset graph completed, took 257ms
[INFO] Checking for updates since last build completed, took 19.9s
[INFO] Running build completed, took 19.0s
[INFO] Caching finalized dependency graph completed, took 265ms
[INFO] Succeeded after 19.3s with 35 outputs (35 actions)
PS C:\Users\Amir\Documents\project\project> █
```

در صورتی که هیچ اروری نده مثل خروجی بالا به صورت خودکار کنار همین فایل یه فایل جدیدی به اسم `todos.g.dart` ساخته.



خب دیگه با پوشه `models` کاری نداریم. وارد فایل `todos.g.dart` و اسم کلاس رو کپی کنید و به مرحله بعد برید ، مراقب باشید هیچ تغییری در فایل ایجاد نکنید و در آینده به مشکل نخورید.

```
main.dart todos.dart todos.g.dart X
lib > models > todos.g.dart > ToDosAdapter
1 // GENERATED CODE - DO NOT MODIFY BY HAND
2
3 part of 'todos.dart';
4
5 // *****
6 // TypeAdapterGenerator
7 // *****
8
9 class ToDosAdapter extends TypeAdapter<ToDos> {
10   @override
11   final int typeId = 1;
```

نوبتی هم باشه نوبت اینیشیالایز کردن هایو و ریجستر کردن کلاسمون هستش.

```
main.dart

1  import 'package:flutter/material.dart';
2  import 'package:hive_flutter/adapters.dart';
3  import 'package:project/models/todos.dart';
4
5  const box_Name = 'todos_box';
6
7  void main() async {
8    Hive.initFlutter();
9
10   Hive.registerAdapter(TodosAdapter());
11   await Hive.openBox<Todos>(box_Name);
12
13   runApp(const MyApp());
14 }
```

مثل فصل های قبل :

. اول import هارو انجام دادیم.

. بعد در لاین ۵ اسم box رو ریختیم توی یه متغیر تا همه جا از این متغیر استفاده کنیم ، برای اینکه احتمال داره خطای املائی پیش بیاد و نام box رو بنزیم که تا حالا وجود نداشته.

. در لاین ۸ اومدیم دیتابیس هایو رو Initialize کردیم، تا بتونیم ازش استفاده کنیم ، Initialize کردن دیتابیس شامل پیدا کردن آدرس ذخیره سازی دیتابیس هم میشه.

. البته اگه میخواید پروژه رو توی اندروید و IOS اجرا کنید، باید Initialize رو با متد Hive.initFlutter انجام بدید و مسیر ذخیره سازی رو با پکیج path_provider به متد بدید.(که البته در فصل های پیش اشاره کردم به این موضوع).

. در لاین ۱۰ کلاسی که ساخته بودیم و نامش رو کپی کرده بودیم رو با متد Hive.registerAdapter() ثبت کردیم.

. در لاین ۱۱ هم box مون رو باز کردیم تا بتونیم توش دیتا هارو ذخیره کنیم و اونارو بخونیم.

و در لاین ۱۳ هم اپ اجرا میشود.

Create , Read , Update , Delete .

خب اولین کلاسی که باید اجرا بشه HomePage هستش:

```
15
16 class MyApp extends StatelessWidget {
17   const MyApp({super.key});
18
19   @override
20   Widget build(BuildContext context) {
21     return const MaterialApp(
22       title: 'Todos App',
23       debugShowCheckedModeBanner: false,
24       home: HomePage(),
25     ); // MaterialApp
26   }
27 }
28
29 class HomePage extends StatefulWidget {
30   const HomePage({super.key});
31
32   @override
33   State<HomePage> createState() => _HomePageState();
34 }
35
36 class _HomePageState extends State<HomePage> {
37   final _titleController = TextEditingController();
38   final _textController = TextEditingController();
39   final box = Hive.box<Todos>(box_Name);
40   @override
41   void dispose() {
42     box.close();
43     super.dispose();
44   }
```

. در این ۳۷ و ۳۸ اومدیم دو تا کنترلر برای TextField هامون تعریف کردیم، که یکی برای عنوان تودو هستش و دیگری برای متن تودو هستش.

. در لاین ۳۹ اومدیم box خودمون رو تعریف کردیم و اون رو داخل یه متغیری ریختیم تا همه جا بتونیم ازش استفاده کنیم. مگر نه میتونید به صورت محلی تعریف کنید و هرجا که لازم داشته باشید دوباره یه نمونه بسازید ولی این روش بالا بهتره و پیشنهاد میکنم از ان روش استفاده کنید.

. در لاین های ۴۰ تا ۴۴ اومدیم و متد dispose رو صدا زدیم. و در لاین ۴۲ گفتیم اگر کاربر از این صفحه خارج شد بیا و box من هم رو ببند تا رم گوشی اشکال نشه.

```

47 Widget build(BuildContext context) {
48   return Scaffold(
49     backgroundColor: Colors.grey.shade200,
50     body: box.values.isEmpty
51       ? const Center(child: Text('You have no todo'))
52       : ListView.builder(
53         itemCount: box.values.length,
54         itemBuilder: (context, index) {
55           final Todos todos = box.values.toList()[index];
56           return Padding(
57             padding: const EdgeInsets.all(8.0),
58             child: ListTile(
59               tileColor: Colors.white,
60               title: Text(
61                 todos.title.toString(),
62                 style: TextStyle(
63                   decoration: todos.isComplete!
64                     ? TextDecoration.lineThrough
65                     : TextDecoration.none,
66                 ), // TextStyle
67             ), // Text
68             subtitle: Text(
69               todos.text.toString(),
70               maxLines: 2,
71               overflow: TextOverflow.ellipsis,
72               style: TextStyle(
73                 decoration: todos.isComplete!
74                   ? TextDecoration.lineThrough
75                   : TextDecoration.none,
76               ), // TextStyle
77             ), // Text
78             leading: Checkbox(

```

الان که همچی آمادست بریم سراغ نمایش دیتا هامون.

. در لاین ۵۰ اومدیم و گفتیم اگر در box من مقداری وجود نداشت وسط صفحه بنویس : شما تودو ندارید، ولی اگر تو box مقداری بود بیا برای م یه لیست بساز.

. در لاین ۵۳ باید طول مقادیر box و بهش بدیم ، برای مثال اگه ما در دیتابیس ۳ تا تودو داریم باید بگیم ۳ تا که ما برابر با طول مقادیر box مون قرار دادیم.

. در لاین ۵۴ باید بیایم و لیست مون رو بسازیم ، در لاین ۵۵ اومدیم یه متغیر ساختیم که مقدارش برابر با مقادیر box مون هست ولی اون مقداری که index برابر با index لیست ما باشه.

. در لاین ۵۶ هم اومدیم یه Padding دادیم تا در دیواره های گوشه فاصله داشته باشه، اومدیم و یه ListTile تعریف کردیم ، متن این listTile ما برابر با متن که در باکسمون هست ، دقت کنید کل Box رو به عنوان متن نمیدیم اگر یادتون باشه در لاین ۵۵ اومدیم و هر دیتا رو ریختیم توی متغیرمون.

. در قسمت subtitle هم اومدیم و ۲ خط از توضیحات اون تودو رو قرار دادیم، در صورتی که از دو خط بیشتر بشه یه ... نمایش داده میشه آخرش.

. و اگر تودو انجام شده باشه یه خط روی عنوان و توضیحات کشیده میشه.

بریم ادامه کد هارو براتون توضیح بدم.

```

78         leading: Checkbox(
79             value: todos.isComplete,
80             shape: const OvalBorder(
81                 side: BorderSide(
82                     style: BorderStyle.solid,
83                 )), // BorderSide // OvalBorder
84             onChanged: (val) async {
85                 todos.isComplete = !todos.isComplete!;
86                 await todos.save();
87                 setState(() {});
88             },
89         ), // Checkbox
90         trailing: IconButton(
91             onPressed: () {
92                 box.deleteAt(index);
93                 setState(() {});
94             },
95             icon: const Icon(Icons.delete),
96         ), // IconButton
97         onTap: () {
98             _dialogEditTodo(context, index);
99         },
100     ), // ListTile
101 ); // Padding
102 },
103 ), // ListView.builder
104 floatingActionButton: FloatingActionButton(
105     onPressed: () {
106         _dialogAddTodo(context);
107     },
108     child: const Icon(Icons.add),
109 ), // FloatingActionButton
110 ); // Scaffold
111 }

```

. در قسمت leading یعنی از لاین ۷۸ تا ۸۹ اومدیم و یه CheckBox قرار دادیم در صورتی کار انجام شده بود این چک باکس فعال و در صورتی که اون تودو انجام نشده بود چک باک غیر فعال میشه ، و با زدن وی چک باکس حالت چک باک تغییر میکنه.

. در قسمت trailing یعنی از لاین های ۹۰ تا ۹۶ اومدیم و یه باتن گذاشتیم برای حذف کردن تودو و پس از حذف کردن متد setState رو فراخوانی کردیم تا لیست و صفحه مون از اول لود بشه .

. در لاین های ۹۷ تا ۹۹ اومدیم از متد onTap ویجت listTile استفاده کردیم و گفتیم اگر کاربر روی یک ز تودو ها زد یعنی میخواد اون تودو رو ویرایش کنه ، پس متد ویرایش رو صدا میزنیم و index اون تودو که میخواد ویرایش بشه رو صدا زدیم.

. در پایین یه floatingActionButton داریم که برای اضافه کردن تودو به کار میره. در صورتی روی این دکمه کلیک بشه متد dialogAddTodo صدا زده میشه.

و تمام ، این بود از ظاهر اپ ، در صفحات بعدی این دو متد Add و Edit رو براتون توضیح میدم.


```

113 _dialogAddTodo(BuildContext context) async {
114   return showDialog(
115     context: context,
116     builder: (context) {
117       return AlertDialog(
118         title: const Text('Add Todos'),
119         content: SizedBox(
120           height: 100,
121           child: Column(
122             children: [
123               TextField(
124                 controller: _titleController,
125                 decoration: const InputDecoration(hintText: "Enter title"),
126               ), // TextField
127               TextField(
128                 controller: _textController,
129                 decoration: const InputDecoration(hintText: "Enter text"),
130               ), // TextField
131             ],
132           ), // Column
133         ), // SizedBox
134         actions: <Widget>[
135           IconButton(
136             onPressed: () {
137               Navigator.of(context).pop();
138               _titleController.clear();
139               _textController.clear();
140             },
141             icon: const Icon(Icons.arrow_back_outlined),
142           ), // IconButton
143           IconButton(
144             onPressed: () {
145               final now = DateTime.now();
146               box.add(
147                 Todos(
148                   title: _titleController.text,
149                   text: _textController.text,
150                   date: now.toString(),
151                   isComplete: false,
152                 ), // Todos
153               );
154               Navigator.of(context).pop();
155               _titleController.clear();
156               _textController.clear();
157               setState(() {});
158             },
159             icon: const Icon(Icons.check),
160           ), // IconButton
161         ], // <Widget>[]
162       ); // AlertDialog

```

. وقتی این متد فراخوانی میشه میاد و یه Dialog رو به ا نمایش میده که شامل دو textField هست.

. به TextField ها کنترلر های خودشون رو دادیم.

. و در قسمت actions دو تا دکمه ساختیم، اولی برای برگشت و انصراف از اضافه کردن و دومی برای تایید و اضافه کردن تودو.

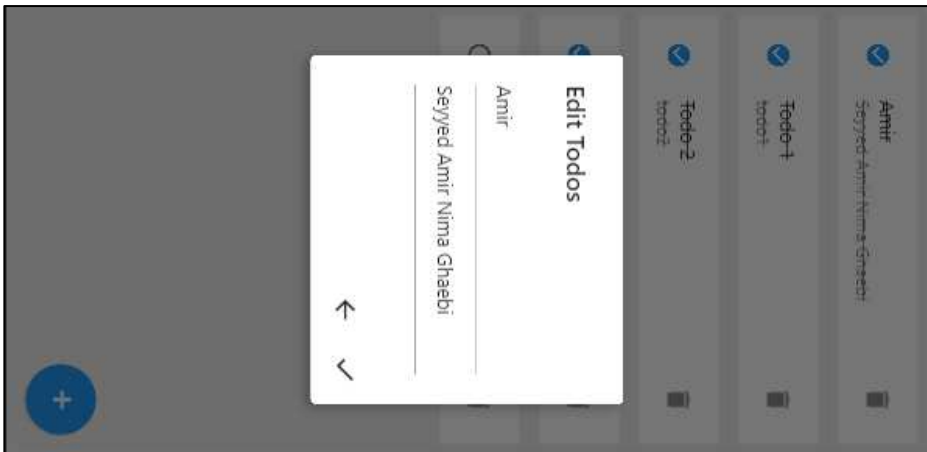
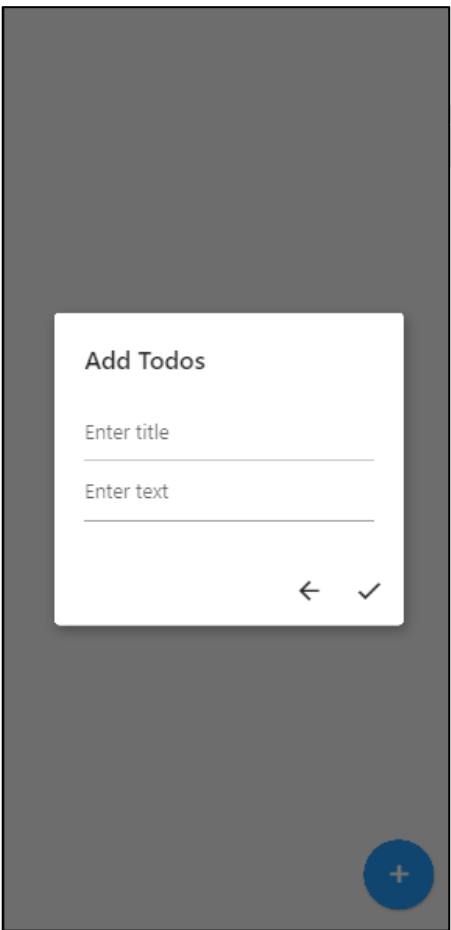
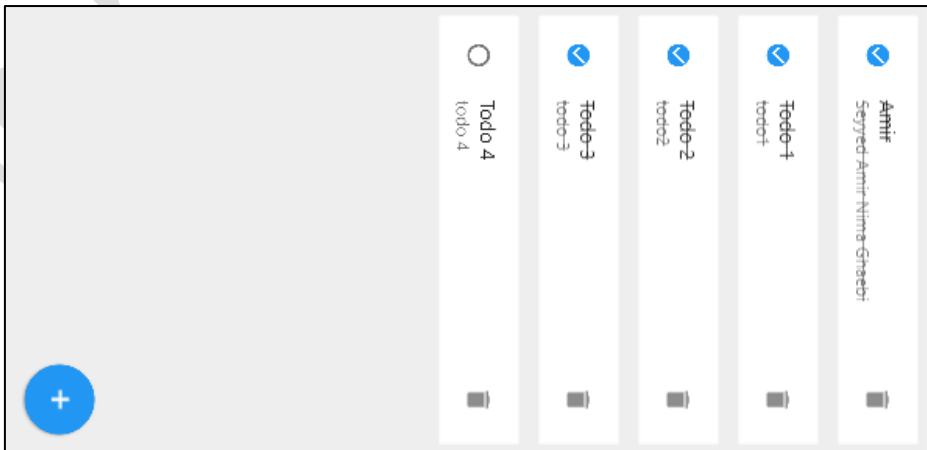
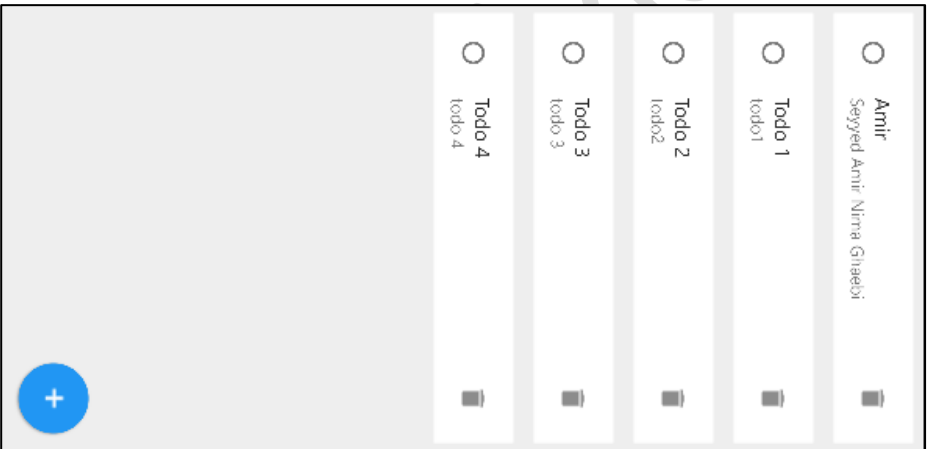
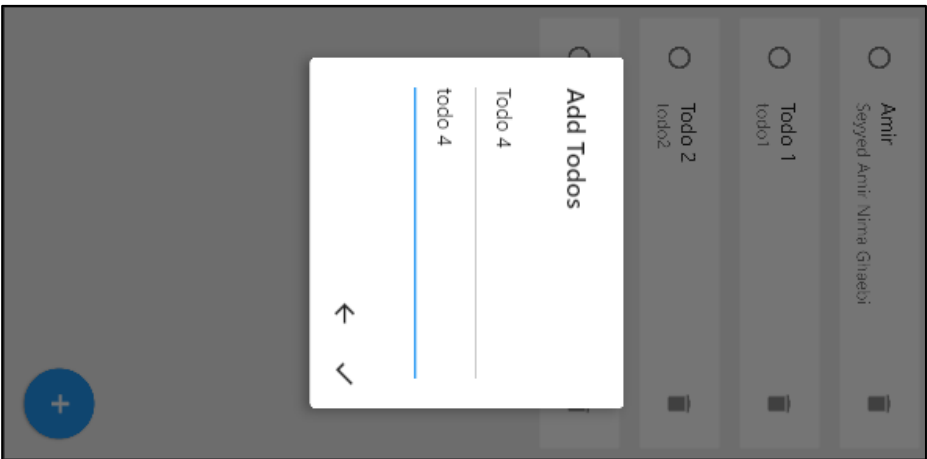
. در لاین ۱۴۵ اومدیم زمانی که تودو ثبت میشه رو داخل یه متغیر ریختیم و در لاین های ۱۴۶ تا ۱۵۳ اومدیم مثل قبل از متد add برای ذخیره دیتای جدید استفاده کردیم ، در لاین ۱۵۴ دیالوگ رو بستیم و در لاین های ۱۵۵ و ۱۵۶ اومدیم و کنترلر هارو پاک کردیم تا بشه دوره متن وارد کرد و در آرم یه setSatet برای اینکه تودو که اضافه شده در لیست ما دیده بشه.

```

167 _dialogEditTodo(BuildContext context, int index) async {
168   final Todos todos = box.values.toList()[index];
169   _titleController.text = todos.title.toString();
170   _textController.text = todos.text.toString();
171   return showDialog(
172     context: context,
173     builder: (context) {
174       return AlertDialog(
175         title: const Text('Edit Todos'),
176         content: SizedBox(
177           height: 188,
178           child: Column(
179             children: [
180               TextField(
181                 controller: _titleController,
182                 decoration: const InputDecoration(hintText: "Enter new title"),
183               ), // TextField
184               TextField(
185                 controller: _textController,
186                 decoration: const InputDecoration(hintText: "Enter new text"),
187               ), // TextField
188             ],
189           ), // Column
190         ), // SizedBox
191         actions: <Widget>[
192           IconButton(
193             onPressed: () {
194               Navigator.of(context).pop();
195               _titleController.clear();
196               _textController.clear();
197             },
198             icon: const Icon(Icons.arrow_back_outlined),
199           ), // IconButton
200           IconButton(
201             onPressed: () {
202               final now = DateTime.now();
203               box.putAt(
204                 index,
205                 Todos(
206                   title: _titleController.text,
207                   text: _textController.text,
208                   date: now.toString(),
209                   isComplete: false,
210                 ), // Todos
211               );
212               Navigator.of(context).pop();
213               _titleController.clear();
214               _textController.clear();
215               setState(() {});
216             },
217             icon: const Icon(Icons.check),
218           ), // IconButton
219         ], // <Widget>[]
220       ); // AlertDialog
221     },
222   );
223 }

```

اینم مثل متد قبلیه و تنها فرقی اینکه با اون index میگیرم اطلاعات اون تودو رو تو textfield نمایش میدیم



. Hive در دیتابیس ValueListenableBuilder

برای اینکه بیایم و یه لیستی بسازیم از دیتا های دیتابیس مون ، که لیست بیاد و دائما به Box ما گوش بده و اگر تغییری توی box ما رخ داد بیاد و لیست مارو هم آپدیت کنه میتونیم از ویجت ValueListenableBuilder برای این کار استفاده کنیم.

```
main.dart

1 body: ValueListenableBuilder(
2   valueListenable: box.listenable(),
3   builder: (context, Box<Todos> box, _) {
4     if (box.values.isEmpty) {
5       return const Center(child: Text("You have no todos"));
6     } else {
7       return ListView.separated(
8         itemCount: box.values.length,
9         itemBuilder: (context, index) {
10           var result = box.getAt(index);
11
12           return Card(
13             child: ListTile(
14               title: Text(result!.title!),
15               subtitle: Text(result.text!),
16               onTap: () {
17                 _dialogEditTodo(context);
18               },
19               trailing: InkWell(
20                 child: const Icon(
21                   Icons.remove_circle,
22                   color: Colors.red,
23                 ),
24                 onTap: () {
25                   box.deleteAt(index);
26                 },
27               ),
28             ),
29           );
30         },
31         separatorBuilder: (context, i) {
32           return const SizedBox(height: 8);
33         },
34       );
35     }
36   },
```

همین طور که در کد بالا میبینید ما در Body از ویجت ValueListenableBuilder استفاده کردیم، که تو پراپرتی داره اولی valueListenable که میاد از ما یه مقداری میگیره و اگه اون مقدار تغییری کرد میاد و لیست مارو آپدیت میکنه.

پراپرتی دوم builder هستش که همون لیست مارو میسازه.

. در قسمت valueListenable اومدیم و box خودمون رو بهش دادیم و یادتون نره حتما از متد listenable استفاده کنید تا اگه تغییری در دیتابیس به وجود اومد به پراپرتی valueListenable خبر کنه از تغییر.

- . در لاین ۳ اومدیم و از متد builder برای ساخت لیست مون استفاده کردیم.
- . در لاین های ۴ تا ۶ اومدیم یه شرطی رو تعریف کردیم که اگه داخل box من هیچ مقداری نبود بیاد و وسط صفحه متنی رو نشون بده ، ولی اگه مقداری بود بیاد و لیست مون رو بسازه.
- . در لاین ۷ اومدیم و یه ListView رو برای ساخت لیست به متد Builder دادیم.
- . در لاین ۸ اومدیم و طول لیستی میخوایم بسازه رو بهش دادیم که برابر با طول دیتا های box مون هستش.
- . از لاین ۹ هم میایم و اون ردیف هایی که باید بسازه رو بهش دادیم.
- . در لاین ۱۰ اومدیم و یه متغیر ساختیم که مقدارش برابر با اون دیتایی که داخل دیتابیس هست برابره ، در واقع میایم و از متد getAt برای دریافت دیتا های فیلدی که با index ما برابر هست رو درمیاریم و داخل متغیری که ساختیم میریزیم.
- . در لاین های ۱۲ تا ۳۰ اومدیم و یه Card داریم و درونش یه ListTile دادیم درون ListTile اومدیم و title رو برابر با عنوان دیتا خودمون قرار دادیم و در subtitle هم متن اون دیتارو نمایش دادیم.
- و در قسمت trailing اومدیم و یه دکمه برای حذف دیتا از box گذاشتیم ، همون طور که میبینید عملیات حذف دیتا با متد deleteAt انجام شده.
- . در لاین های ۳۱ تا ۳۳ اومدیم و یه فاصله ای بین ردیف هامون ایجاد کردیم.

. باز نویسی متد dialogEditTodo

در پروژه Todo برای ویرایش دیتا ها اول میومدیم index رو میفرستادیم و سپس در متد ویرایش با استفاده از index به box میرفتیم و اون دیتا رو پیدا میکردیم و سپس عملیات ویرایش رو انجام میدادیم.

اما ما میتونیم به جای اینکه index دیتا رو بفرستیم بیایم و خود دیتا رو بفرستیم .

```
150 _dialogEditTodo(BuildContext context, Todos todos) async {
151   _titleController.text = todos.title.toString();
152   _textController.text = todos.text.toString();
153
154   return showDialog(
155     context: context,
156     builder: (context) {
157       return AlertDialog(
158         title: const Text('Edit Todos'),
159 >       content: SizedBox( // SizedBox ...
175         actions: <Widget>[
176 >       IconButton( // IconButton ...
184         IconButton(
185           onPressed: () {
186             final now = DateTime.now();
187             todos.title = _titleController.text;
188             todos.text = _textController.text;
189             todos.date = now.toString();
190             todos.isComplete = false;
191             todos.save();
192
193             Navigator.of(context).pop();
194             _titleController.clear();
195             _textController.clear();
196             setState(() {});
197           },
198 ✓       icon: const Icon(Icons.check),
199       ), // IconButton
200     ], // <Widget>[]
201   ); // AlertDialog
202 },
203 );
```

(برای اینکه کد ها زاد نشه من یه سری از کد هارو خفی کردم ، نگران نباشید این کد هارو در درس قبلی توضیح دادم)، توجه کنید اگر از ValueListenableBuilder استفاده میکنید نیازی به setState آخر نیست.

. همون طور که در لاین ۱۵۰ میبینید اومدیم و بجای index یه Todos گرفتیم که یه اسمم داره.

. در لاین ۱۵۱ و ۱۵۲ اومدیم و متن های textField هارو برابر با عنوان و متن اون Todo که فرستادیم کردیم.

. در لاین های ۱۸۶ تا ۱۹۷ اومدیم و عملیات Save رو به این شکل جدید انجام دادیم.

. در لاین ۱۸۶ اومدیم و زمان تغییر رو توی یه متغیر ریختیم.

. در لاین های ۱۸۷ تا ۱۹۰ اومدیم و دیتا های جدید رو به دیتا مون دادیمو

. در لاین ۱۹۱ اومدیم و با متد save تغییراتی که در دیتا به وجود آوردیم رو ذخیره کردیم.

و در آخرم هم به صفحه home برگشتم و متن های textField هارو حذف کردیم و یه setState.



فصل هفتم : بررسی مباحث پیشرفته و کاربردی دیتابیس Hive



```

1
2 import 'package:hive/hive.dart';
3
4 part 'todos.g.dart';
5
6 @HiveType(typeId: 1)
7 class Todos extends HiveObject {
8   @HiveField(0, defaultValue: 'title')
9   String title;
10  @HiveField(1, defaultValue: '')
11  String text;
12  @HiveField(2, defaultValue: '2023/**/**')
13  String date;
14  @HiveField(3, defaultValue: false)
15  bool isComplete;
16
17  Todos({
18    required this.title,
19    required this.text,
20    required this.date,
21    required this.isComplete,
22  });
23 }
24

```

اگر در پروژه گذشته یادتون باشه ما برای اینکه دیتا ها null نباشن میومدیم و از یه '' خالی استفاده میکردیم.

ولی اگه دیتامون از جایی بیاد که null باشه دیتابیس ارور میده حتما.

برای اینکه دیتا هایی که میفرستیم اگه null بود اروری نده میتونیم از defaultValue استفاده کنیم.

در کد بالا مثالی کامل رو مشاهده میکنید.

```

1  @HiveType(typeId: 2)
2  enum Priority {
3    @HiveField(0)
4    low,
5
6    @HiveField(1)
7    normal,
8
9    @HiveField(2)
10   high,
11  }

```

برای ذخیره سازی enum ها تو دیتابیس همیشه enum رو به عنوان فیلد به دیتابیس داد، باید در کلاسی جداگانه ازش استفاده کرد.

در کد بالا مثالی ساده از Enum ها رو مشاهده میکنید.

در enum ها هم میتونید از defaultValue استفاده کنید.

defaultValue در enum ها با true انجام میشود.

```

1  @HiveType(typeId: 2)
2  enum Priority {
3    @HiveField(0)
4    low,
5
6    @HiveField(1)
7    normal,
8
9    @HiveField(2, defaultValue: true)
10   high,
11  }

```

Relationships .

دیتابیس Hive میتونه روابط بین کلاس هارو هم ذخیره کنه.

پس هیچ نگرانی نداشته باشید.

برای مثال من یه کلاسی دارم به اسم Person که لیست افرادم رو توش ذخیره میکنم و در این لیست یه لیستی دیگه از Person به اسم friends برای ذخیره سازی دوستان هر کدوم از Person به کار میره.

در کد پایین مثالی ساده از این روابط رو مشاهده میکنید.



```
1 class Person extends HiveObject {
2   String name;
3
4   int age;
5
6   List<Person> friends;
7
8   Person(this.name, this.age);
9 }
10
```


Snippets for VSCode .

در ویژوال استودیو کد میانبر های مختلفی داریم که کار مارو خیلی ساده تر و سریع تر میکنه. این میانبر ها انواع مختلفی دارن بعضی برای خود ویژوال استودیو کد هستن مثل میانبر `Ctrl + Shift + P` و بعضی از میانبر ها برای پکیج هایی هست که به پروژه اضافه میکنیم.

این نوع میانبر ها چی هستن؟ برای مثال پکیج خود flutter که با زدن میانبر `stl` میایم و یه کلاس `Stateless` میسازیم یا با میانبر `stf` میومدیم و یه کلاس `Stateful` میساختیم.

پکیج Hive هم این نوع میانبر هارو برای راحتی کار و سریع شدن کار برای ما گذاشته.

برای مثال با زدن کلمه `ht` و بعد `Enter` میاد برای ما یه `HiveType` خودکار میسازه.

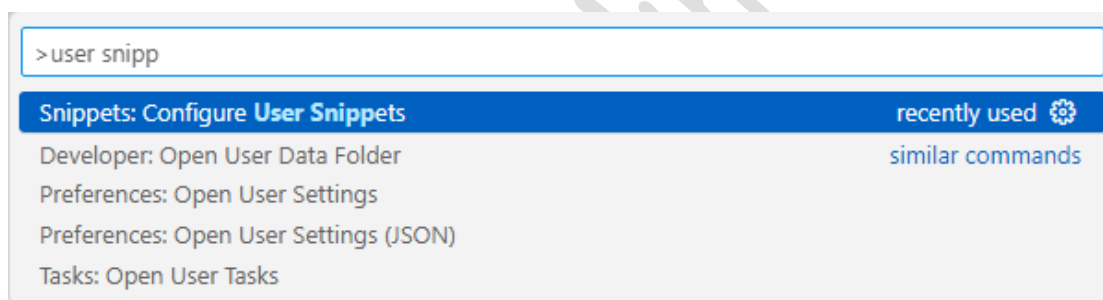
اما چطور ازش استفاده کنیم؟

متأسفانه پکیج Hive به طور پیشفرض این قابلیت رو نداره.

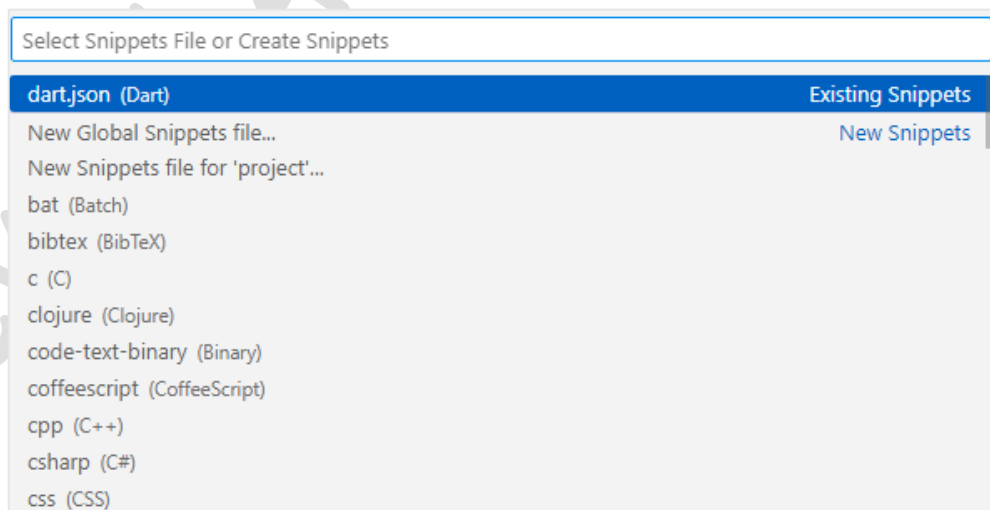
پس اولین کاری باید انجام بدیم این هست که این قابلیت رو به ویژوال استودیو کد اضافه کنیم.

اول از همه کلید های ترکیبی `Ctrl + Shift + P` رو بزنید. تا پنجره پایین براتون باز بشه.

بعد از باز شدن پنجره عبارت `user snipp` رو تایپ کنید.



روی اولین گزینه کلیک کنید تا پنجره زیر براتون باز بشه.



ما با میانبر های زبان های دیگه کاری نداریم روی زبان dart بزنید تا فایل `dart.json` رو برای ما باز کنه. به طور پیشفرض داخل فایل کد خاصی وجود نداره پس همشو پاک کنید و کد های زیر رو اونجا بزارید.

```
main.dart  todos.dart  dart.json  X
C:\Users\Amir\AppData\Roaming\Code\User\snippets> dart.json
1 {
2   "HiveType class extends HiveObject": {
3     "prefix": "hive",
4     "body": [
5       "import 'package:hive/hive.dart';",
6       "",
7       "part '${TM_FILENAME_BASE}.dart';",
8       "",
9       "@HiveType()",
10      "class ${TM_FILENAME_BASE/(.*)/${1:/pascalcase}/} extends HiveObject {",
11        "\t@HiveField(0)",
12        "\tString id;",
13        "\t${0:// Hive fields go here}",
14      "}"
15    ],
16    "description": "Creates an HiveType class extending HiveObject structure based on the filename."
17  },
18  "HiveType class": {
19    "prefix": "hive",
20    "body": [
21      "import 'package:hive/hive.dart';",
22      "",
23      "part '${TM_FILENAME_BASE}.dart';",
24      "",
25      "@HiveType()",
26      "class ${TM_FILENAME_BASE/(.*)/${1:/pascalcase}/} {",
27        "\t@HiveField(0)",
28        "\tString id;",
29        "\t${0:// Hive fields go here}",
30      "}"
31    ],
32    "description": "Creates an HiveType class structure based on the filename."
33  },
}
```

حدود ۱۰۰ خط کد هستش که در تصویر بالا فقط تا ۳۳ خط رو گذاشتیم
میتونید این کد رو از لینک زیر کپی کنید.

<https://docs.hivedb.dev/#!/more/vscode-snippets?id=usage>

کد رو کپی کنید و در فایل dart.json پیست کنید و فایل رو ذخیره کنید.
خب حالا میتونیم از این قابلیت استفاده کنیم.

Command	Action
hive	Creates the basic structure of a Hive model and extends HiveObject
hivec	.Creates the basic structure of a Hive model
hf	Creates a annotation. HiveField()
ht	Creates a annotation. HiveType()
hfs	Creates a String Hive field.
hfi	Creates an int Hive field.
hfb	Creates a bool Hive field.
hfl	.Creates a List Hive field
hfd	.Creates a double Hive field

. پیدا کردن دیتا با استفاده از کلمه

در فصل های گذشته ما تونستیم دیتا های box مون رو با استفاده از index و یا فرستادن خود دیتا به متد آشنا شدید.

گاهی وقت ها نه index داریم و نه خود دیتا رو .

برای اینکه به اون دیتا دسترسی داشته باشیم یه راه حل سومی هم وجود داره.

شما میتونید با استفاده از یکی از فیلد های اون دیتا برای مثال از فیلد title در دیتای Todos میتونیم استفاده کنیم.

یعنی اگه من عنوان یکی از todo هامون بدونم میتونم به اون دسترسی داشته باشم .

اما چطوری؟

```
main.dart

1 @override
2 Widget build(BuildContext context) {
3   final box = Hive.box<Todos>(boxName);
4   final Todos todos =
5     box.values.firstWhere((element) => element.name == 'Gym');
6
7   return Scaffold();
8   ;
```

در کد بالا مثالی کامل از انجام این کارو مشاهده میکنید.

. در لاین ۳ اومدیم یه متغیر ساختیم که توش یه نمونه از box مون ریختیم.

. در لاین ۴ اومدیم و عملیات جستجو رو انجام دادیم.

اول یه متغیر از نوع Todos به نام todos ساختیم.

بعد اومدیم و box مون رو صدا زدیم و با عبارت value به دیتا های box مون دسترسی پیدا کردیم.

و بعد با متد firstWhere تونستیم در box عملیات جستجو رو انجام بدیم.

در قسمت اول یعنی (element) اومدیم و تمام دیتا های باکس رو آوردیم.

و گفتیم دیتایی در متغیر todos من بیاد که عنوانش برابر با Gym باشه .

این طوری متد firstWhere میاد و در box ما میگردد و اون دیتایی که عنوانش برابر با Gym باشه رو در متغیر todos میریزه.

اگر پیدا نکرد مقدار null برگشت داده میشه.

. جستجو در دیتابیس و دیدن نتایج آن در ListView

خب بیشتر اپ هایی که ازشون استفاده میکنیم قسمت جستجو وجود داره. ما در دیتابیس Hive هم میتونیم عملیات جستجو رو انجام بدیم و نتایج جستجو رو به کاربر نشون بدیم.

```
search_screen.dart

1 class _SearchScreenState extends State<SearchScreen> {
2   final _searchController = TextEditingController();
3   List<dynamic> _searchResults = [];
4
5   @override
6   void dispose() {
7     _searchController.dispose();
8     super.dispose();
9   }
10
11   void _performSearch() async {
12     final results = await searchInBox(_searchController.text);
13     setState(() {
14       _searchResults = results;
15     });
16   }
17
18   @override
19   Widget build(BuildContext context) {
```

. دی لاین های ۲ و ۳ اومدیم یه کنترلر برای textField که کاربر میخواهد توش تایپ کنه تعریف کردیم. و در خط پایین هم یه لیستی ایجاد کردیم از نوع داینامیک که نتایج لیست ما توش نمایش داده میشه.

. در متد dispose اومدیم و کنترلر مون هم dispose کردیم.

. در لاین های ۱۱ تا ۱۶ اومدیم و یه متدی برای سرچ کردن ساختیم.

که میاد با استفاده از یه متد دیگه عملیات سرچ رو انجام میده.

متد بعدی رو در صفحه بعد نشون میدیم.

اما متد _performSearch چیکار میکنه؟

میاد یه final به اسم result میسازه و توش مقداری که متد searchInBox میده رو ریخته. که همون نتایج ما میشن.

حتما از await استفاده کنید چون عملیات جستجو طول میکشه.

در آخرم اون لیست بالا رو برابر با results ما میزازه و از setState هم استفاده کند تا مقداری که استفاده شدن آپدیت بشن.

```

1  Future<List<Todos>> searchInBox(String searchString) async {
2      final box = await Hive.openBox<Todos>(boxName);
3      var results = <Todos>[];
4
5      results.addAll(box.values
6          .where((item) =>
7              item.title!.contains(searchString))
8          .toList());
9
10     return results;
11 }

```

به این صورت.

متد ما که از جنس لیستی از تودو ها هست از ما یه string میگیره که همون متن جستجو هستش.

. در لاین ۲ که اومدیم و box مون رو ساختیم و توی یه متغیری ریختیم.

. در لاین ۳ اومدیم و یه لیستی از جنس تو دو ها ساختیم که قراره دیتا هامون رو توش بریزیم.

. در لاین های ۵ تا ۸ اومدیم و گفتیم به لیست ما یعنی results بیا هر چی میدم رو بهش اضافه کن addAll توی متد addAll هر چی بدیم از جنس تو دو میاد به لیست اضافه میکنه.

که ما اومدیم هر چی دیتا داخل box داریم رو به متد دادیم box.value پس همه دیتا هامون به لیست اضافه میشه و ما اینو نمیخوایم ، چرا؟ چون همه دیتا هارو لازم نداریم فقط دیتا هایی که توشون از مقدار متغیر searchString استفاده شده به لیست من اضافه بشه.

. در لاین ۶ اومدیم همین کارو کردیم ، اومدیم از عبارت where برای جستجو در دیتا ها استفاده کردیم.

گفتیم دیتا هایی رو بیار که توش کلمه searchString استفاده شده باشه contains .

و در آخر همه دیتا هارو به لیست تبدیل کردیم و ریختیم توی لیستمون.

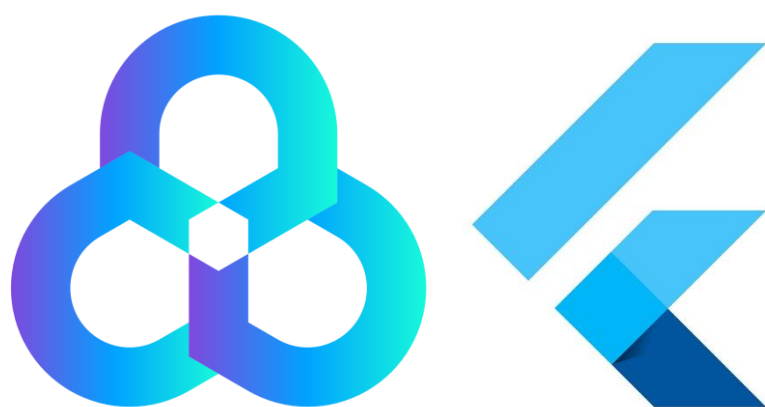
در آخرم هم لیست رو return کردیم.

خب حالا جستجو رو انجام دادیم و نتایج رو در لیستی که در لاین ۳ در صفحه ۶۷ ریختیم.

حالا که لیست آمادست میتونیم و اون رو نمایش بدیم.

کافیه متد performSearch_ در متد onChange فراخوانی بکند.

و لیست رو توی لیست ویو نشون بدید. همین (:



فصل هشتم : بررسی تمام متدهای مهم و کاربردی پکیج Hive

isEmpty & isNotEmpty .

بعضی مواقع ما نیاز داریم که بدونیم آیا در box مون مقداری داریم یا نه. عبارت های isEmpty این کاو برای ما انجام میده.

```
main.dart

1 var box = await Hive.openBox(boxName);
2
3 print('isEmpty is : ${box.isEmpty} -- isNotEmpty is : ${box.isNotEmpty}');
```

متد isEmpty به ما bool بر میگردونه. متد isNotEmpty دقیقا برعکس این متد هستش یعنی میگه آیا داخل box چیزی نیست؟

```
isEmpty is : false -- isNotEmpty is : true
```

خروجی کد به رو در عکس بالا میتونید ببینید.

hashCode .

این عبارت به ما یه کد بر میگردونه که در box های مختلف این کد فرق میکنه این کد کامل یونیک هست و تکراری نمیشه.

```
main.dart

1 var box = await Hive.openBox(boxName);
2
3 print('hashCode is : ${box.hashCode}');
```

خروجی به صورت زیر میشود.

```
hashCode is : 1047865851
```

شما میتونید با این کد چک کنید ببینید آیا دیتابیس تغییری کرده هست یا نه چون کوچک ترین تغییر در حد یک حد باعث میشود کل کد تغییر کند.

Values .

همون طور که قبلا هم با این عبارت کار کردید میدونید از عبارت برای دسترسی به دیتا های یک box استفاده میشود.

```
main.dart
1 var box = await Hive.openBox(boxName);
2
3 print('Value is : ${box.values}');
```

به این صورت نتیجه رو در تصویر زیر میتونید مشاهده کنید.

```
Value is : (ga7089036@gmail.com, Amir, 123521565)
```

isOpen .

این عبارت چه کاربردی دارد؟

میاد چک میکنه که آیا box با باز هست یا بسته (توجه کنید که در صورت بسته بودن box هیچ اطلاعاتی همیشه توش ذخیره کرد و همیشه اطلاعاتی رو ازش خوند)
مقدار بازگشتی این عبارت از جنس bool هست.

```
main.dart
1 var box = await Hive.openBox(boxName);
2
3 print('is Open ${box.isOpen}');
```

خروجی به شکل زیر خواهد بود:

```
is Open true
```


Keys .

کاربرد این عبارت دقیقا مثل عبارت Values هست و تنها فرق شون اینکه عبارت Values میاد و مقدار هارو بهمون میده ولی Keys میاد و کلید هارو بهمون میده.

```
main.dart

1 var box = await Hive.openBox(boxName);
2
3 print('Keys is : ${box.keys}');
```

خروجی به شکل زیر خواهد بود:

```
Keys is : (email, firstName, password)
```

Lazy .

این عبارت به ما میگوید که آیا جنس box ما از نوع lazy هست یا خیر.

```
main.dart

1 var box = await Hive.openBox(boxName);
2
3 print('lazy is : ${box.lazy}');
```

خروجی به صورت زیر خواهد بود:

```
lazy is : false
```

توجه کنید اگر نوع box از نوع lazy باشه این مقدار true میشود.

Length .

در طول آموزش خیلی با این عبارت کار کردیم که میومد و طول دیتا هایی که در box ذخیره شدن رو بهمون میداد.

```
main.dart

1  var box = await Hive.openBox(boxName);
2
3  print('Length is : ${box.length}');
```

خروجی کار به صورت زیر خواهد بود:

Length is : 3

name .

این عبارت میاد و نام box رو بهمون میده.

```
main.dart

1  var box = await Hive.openBox(boxName);
2
3  print('Name is : ${box.name}');
```

خروجی کال به شکل زیر خواهد بود :

Name is : user_box

همون طور که میبینید دقیقا همون نامی که استفاده کردیم رو نشون میده.

. toMap

این عبارت میاد و مقادیر box مارو به map تبدیل میکنه و میتونیم از مقداری خودمون به صورت map شده استفاده کنیم.

```
main.dart

1  var box = await Hive.openBox(boxName);
2
3  print(box.toMap());
```

. containsKey

این متد میاد و از ما یه کلید میگیره و میره توی box و میگرده از اون کلید رو توی boxمون پیدا کرد مقدار true رو بهمون برمیگردونه ولی اگه این کلید رو توی box نتونست پیدا کنه مقدار false رو برمیگردونه.

```
main.dart

1  var box = await Hive.openBox(boxName);
2
3  print(box.containsKey('email'));
```

. compact

این متد میاد و باکس مارو فشرده میکنه و دیتا هایی که لازم نیست رو حذف میکنه ، بهتره بعد از حذف کردن دیتا ها این متد رو صدا بزنید.

```
main.dart

1  var box = await Hive.openBox(boxName);
2
3  //delete
4
5  box.compact();
```

Clear .

این میاد تمام دیتا های box رو حذف میکنه.

```
main.dart

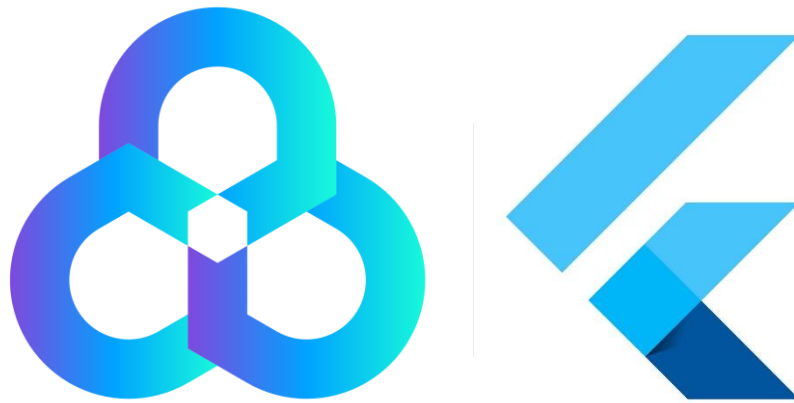
1  var box = await Hive.openBox(boxName);
2
3  box.clear();
```

deleteFromDisk .

این متد میاد و کل box رو حذف میکنه و دیگه قابل استفاده نیست.

```
main.dart

1  var box = await Hive.openBox(boxName);
2
3  box.deleteFromDisk();
```



فصل نهم : ذخیره فایل و تصاویر به دو روش در دیتابیس Hive

. ذخیره تصاویر به صورت بایت در دیتابیس

اولین روش برای ذخیره سازی تصاویر توی دیتابیس اینکه تصویر رو از کاربر بگیریم و سپس عکس رو به بایت (Byte) تبدیل کنیم و سپس بایت هارو توی دیتابیس ذخیره کنیم. و برای نشون دادن عکس به کاربر میایم و بایت هارو از دیتابیس دریافت میکنیم و سپس دوباره به عکس تبدیل میکنیم و به کاربر نمایش میدیم.

```
main.dart

1 class _MyHomePageState extends State<MyHomePage> {
2   ImagePicker picker = ImagePicker();
3   XFile? file;
4   Uint8List? imageByte;
5
6   @override
7   Widget build(BuildContext context) {
8     return Scaffold(
9       body: Column(
10        children: [
11          ElevatedButton(
12            onPressed: () async{
13              XFile? image = await picker.pickImage(source: ImageSource.gallery);
14
15              if(image != null) {
16                file = image;
17              } else{
18                print('Error');
19                return;
20              }
21
22              imageByte = await file?.readAsBytes();
23              print(imageByte);
24
25              setState(() {});
26            },
27            child: const Text('Select Image');
28          ),
29          imageByte != null ? Image.memory(imageByte!) : const Text('Null');
30        ],
31      ),
32    );
33  }
34 }
```

در لاین های ۲ تا ۴ اومدیم یه نمونه از کلاس imagePicker ساختیم.

و دو تا متغیر که نوع شون رو در کد میتونید مشاهده کنید.

در لاین ۱۳ اومدیم و یه متغیر از نوع XFile ساختیم و توش اومدیم با استفاده از کلاس imagePicker یه عکس از گالری کاربر دریافت کردیم.

در لاین های ۱۵ تا ۲۰ اومدیم و شرط گذاشتیم و گفتیم که آیا عکسی از گالری دریافت کردیم null هست یا نه واقعا عکس رو کاربر داده.

اگر که Null بود بیا و return کن و دیگه ادامه نده.

بعد در لاین ۲۲ اومدیم و گفتیم که این عکس رو با استفاده از متد readAsBytes به بایت تبدیل کن.

الان بایت ها در متغیر imageByte هستش.

در لاین ۲۵ هم setState رو فراخونی کردیم تا تغییرات اعمال بشه.

در لاین ۲۹ هم اومدیم و چک کردیم که آیا imageByte من null هست اگر نیست بیا و یه ویجت Image رو نمایش بده.

با استفاده از Image.memory میتونیم بایت هارو تبدیل به عکس کنیم.

شما میتوانید متغیر imageByte رو توی دیتابیس ذخیره کنید و هر جا لازم داشتید ازش استفاده کنید.

این روش کمی غیر استاندارد هست و حتی خود Hive هم گفته این روش برای ذخیره چند مگابایت عکس خیلی خوبه ولی اگه حجم عکس ها بالا باشه قطعا به مشکل میخورید.

برای حل این مشکل میتونید از روش بعدی استفاده کنید.

راستی یادتون نره برای دریافت تصویر از گالری کاربر حتما باید پکیج ImagePicker رو به پروژه تون اضافه کنید.

. ذخیره تصاویر به صورت آدرس در دیتابیس

در روش قبلی اومدیم و عکس هارو به بایت تبدیل کردیم و سپس بایت هارو توی دیتابیس ذخیره کردیم که برای تصاویری با حجم بالا این کار به ارور برخورد میکرد.

در این روش ما میام و تصویری رو از کاربر دریافت میکنیم و سپس تصویر رو در پوشه مخصوص اپ خودمون کپی میکنیم و در نظر داشته باشید که عکسی که توی این پوشه خود اپ قرار بگیره گالری نمیتونه به اون دسترسی داشته باشه و این یعنی عکسایی که توی پروژه استفاده میکنیم توی گالری قابل مشاهده نیست و این خیلی خوبه.

و سپس آدرس عکس رو توی دیتابیس ذخیره میکنم و هر موقع به اون عکس نیازی داشتیم به راحتی میتونیم ازش استفاده کنیم.

فرض میکنیم ما به اپی داریم که میخوایم دارو هایی رو توش ذخیره کنیم و در کنار مشخصات دارو عکس دارو رو هم باید ذخیره .

خب بیاین با هم این پروژه کوچیک رو انجام بدیم



```
1 import 'package:hive_flutter/adapters.dart';
2
3 part 'medicine_list.g.dart';
4
5 @HiveType(typeId: 0)
6 class Medicines extends HiveObject {
7   @HiveField(0)
8   String? name;
9   @HiveField(1)
10  String? description;
11  @HiveField(2)
12  String? image;
13
14  Medicines({
15    required this.name,
16    required this.description,
17    required this.image,
18  });
19 }
20
```

این از دیتابیس مون خودتون کلاس رو بسازید من مستقیم میرم سر کد ها بخاطر اینکه این چیزا رو قبلا یاد گرفته بودیم.


```
main.dart

1 class AddScreen extends StatefulWidget {
2   const AddScreen({super.key});
3
4   @override
5   State<AddScreen> createState() => _AddScreenState();
6 }
7
8 File? image;
9 String? imagePath;
10 File? resultPath;
11 String? format;
12
13 class _AddScreenState extends State<AddScreen> {
14   List<String> tagsList = [];
15   final _tagsController = TextEditingController();
16   final _nameController = TextEditingController();
17   final _descriptionController = TextEditingController();
18   @override
19   void dispose() {
20     _tagsController.dispose();
21     _nameController.dispose();
22     _descriptionController.dispose();
23     super.dispose();
24   }
```

این از اولین کد های کلاسمون.

همه رو با هم کار کردیم پس توضیح اضافه ای نمیدم که وقتتون تلف نشه.

حالا یه باتن بسازید و کد های صفحه بعد رو توش متد onPressed بنویسید.

```
main.dart

1      onTap: () async {
2          final box = await Hive.box<Medicines>(boxName);
3          if (_nameController.text.isEmpty) {
4              showSnackBar(context, 'نام دارو را وارد کنید', Colors.redAccent);
5              return;
6          } else {
7              bool medicineExists = await box.values
8                  .any((medicine) => medicine.name == _nameController.text);
9
10             if (medicineExists) {
11                 showSnackBar(context, 'نام دارو تکراری هست', Colors.redAccent);
12                 return;
13             }
14         }
15         await copyImage();
16         await box.add(
17             Medicines(
18                 name: _nameController.text,
19                 description: _descriptionController.text,
20                 image: imagePath ?? '',
21             ),
22         );
23         image = null;
24         imagePath = null;
25         resultPath = null;
26         format = null;
27         setState(() {});
28         Navigator.of(context).pushReplacement(
29             MaterialPageRoute(
30                 builder: (context) => HomeScreen(isStarted: false),
31             ),
32         );
33     },
```

در لاین ۲ اومدیم و box مون رو ساختیم.

در لاین های ۳ تا ۱۴ اومدیم چک کردیم که آیا نام دارو وارد شده اگه نامش وارد شده آیا این دارو قبلا تو دیتابیس بوده یا نه.

در صورتی که نام رو وارد نکرده باشه یا قبلا اضافه شده بوده میای و یه پیغامی نشون میدیم.

در لاین ۱۵ اومدیم از متد copyImage استفاده کردیم و عکس رو توی اون پوشه کپی کردیم در ادامه متد رو هم توضیح میدم.

در لاین های ۱۶ تا ۲۲ اومدیم و دارو جدید رو اضافه کردیم به دیتابیس همون طور که میبینید در لاین ۲۰ اومدیم و آدرس تصویر رو دادیم و گفتیم اگر null بود هیچی نذار.

بعد در آخر متغیر هارو null میکنم که اگر خواستیم دوباره تصویری رو اضافه کنیم به مشکلی نخوردیم. در آخرم اومدیم و به صفحه اصلی انتقال دادیم کاربر رو.

```

1 Future<void> copyImage() async {
2   if (image != null) {
3     try {
4       Directory appDocDir = await getApplicationDocumentsDirectory();
5       String appDocPath = appDocDir.path;
6
7       String fileName = DateTime.now().millisecondsSinceEpoch.toString();
8       String destPath = '$appDocPath/$fileName.$format';
9
10      await image!.copy(destPath);
11      setState(() {
12        imagePath = destPath;
13        resultPath = destPath as File?;
14      });
15    } catch (e) {}
16  }
17 }

```

خب در کد بالا هم عملیات کپی کردن عکس رو انجام دادیم

در لاین ۴ اومدیم و آدرس پوشه داکيومنت اپ رو با استفاده از پکیج path_provider در آورديم و ريختيم توی يه متغیري.

در لاین ۷ هم اومدیم اسم برای عکس مون انتخاب کردیم و که همون زمان ذخیره سازی هستش.

در لاین ۸ اومدیم و آدرس کامل رو ساختيم.

و در لاین ۱۰ عملیات کپی کردن رو انجام دادیم.

و سپس دو متغیري که بالا ساخته بودیم رو مساوی با آدرس عکس میزارم

الان آدرس عکس توی همونی که توی متغیر destPath و imagePath هستش.

برای گرفتن عکس از کاربر میتونید از کد زیر استفاده کنید

```

1 Future<void> pickCameraImage() async {
2   await Permission.photos.request();
3   await Permission.storage.request();
4   await Permission.manageExternalStorage.request();
5   final picker = ImagePicker();
6   final pickedImage = await picker.pickImage(source: ImageSource.camera);
7   if (pickedImage == null) return;
8   setState(() {
9     image = File(pickedImage.path);
10    format = pickedImage.path.split('.').last.toLowerCase();
11  });
12 }

```

شما میتونید از این روش بالا برای ذخیره صدا و فایل های مختلف استفاده کنید.

. سخن پایانی

دوستان عزیز این مقاله هم به پایان خودش رسید
امیدوارم واقعا این مقاله به درد تون خورده باشه و نهایت استفاده رو ازش کرده باشید.
این مقاله کاملا رایگان هست و هیچ کس نمیتونه این مقاله رو به فروش برسونه.
پیشنهادم اینکه مقاله رو چاپ کنید تا همیشه باهاتون باشه.
سعی کنید از یاد گیری دست بر ندارید و د تو چیزی که استعدادش رو دارید تلاش کنید.
من عاشق برنامه نویسی هستم و براش تلاش میکنم و خواهم کرد.
هر سوالی که داشتید حتما بهم پیام بدید تا کمکتون کنم.

شما میتونید از راه های زیر باهام در ارتباط باشید.

- Email: gaY۰۸۹۰۳۶@gmail.com
- Telegram: https://t.me/AmirGh_۶۶۶
- Github: github.com/SeyyedAmirNimaGhaebi

۱. <https://docs.hivedb.dev/#/>
۲. <https://blog.devgenius.io/the-ultimate-guide-to-using-hive-in-flutter-۵f۵cada۸۸۸۴۱>
۳. <https://medium.com/@mustafatahirhussein/hive-database-in-flutter-an-overview-۳۰e۶d۴ad۷۴۲۴>
۴. <https://virgool.io/flutter-community/%D۸%AF%DB%۸C%D۸%AA%D۸%AY%D۸%A۸%DB%۸C%D۸%B۳-hive-%D۸%AF%D۸%B۱-%D۹%۸۱%D۹%۸۴%D۸%AY%D۸%AA%D۸%B۱-%D۸%A۸%D۹%۸۷-%D۹%۸۷%D۹%۸۵%D۸%B۱%D۸%AY%D۹%۸۷-%D۹%۸۵%D۸%AB%D۸%AY%D۹%۸۴-vynkker۲۴b۱۱>
۵. <https://flutter-learn.ir/hive/>
۶. <https://www.youtube.com/watch?v=IZSTMNufYkc>
۷. <https://www.youtube.com/watch?v=h۸YrmFWNpLA>
۸. <https://www.youtube.com/watch?v=aaR۸۷۱DK۴qc>
۹. <https://www.aparat.com/v/CqPah>
۱۰. https://www.youtube.com/watch?v=xN_OTO۵EYKY
۱۱. <https://www.youtube.com/watch?v=w۸cZKm۹s۲۲۸>
۱۲. <https://www.youtube.com/watch?v=ee۲RUDriM۵g>
۱۳. <https://www.youtube.com/watch?v=FB۹GpmL۰Qe۰>
۱۴. <https://blog.logrocket.com/handling-local-data-persistence-flutter-hive/>
۱۵. <https://viveky۲۵۹۲۵۹.medium.com/hive-for-flutter-۳-implementation-e۸۲۵a۰a۸۳۳d۳>
۱۶. <https://www.kindacode.com/article/flutter-hive-database/>