YouTube BM DERSLERİ

www.youtube.com/BMdersleri

MAKÜ
BURDUR MEHMET AKİF ERSOY ÜNİVERSİTESİ

Flutter ile Mobil Programlamaya Giriş

Flutter

13.HAFTA
HAZIR PAKET KULLANMA
CİHAZ HAFIZASINA VERİ KAYDETME
JSON PARSE İŞLEMLERİ

Hazırlayan : **Zeynep İrem KESLER 1911404048**

Tarih : 01/06/2022

Sürüm : v1

Ders Yürütücüsü : Doç. Dr. İsmail KIRBAŞ

İÇİNDEKİLER

Package ve Plugin Nedir?

Paket: En azından Dart paketi, bir pubspec dosyası içeren bir dizindir. Ek olarak, bir paket bağımlılıklar (pubspec'te listelenmiştir), Dart kitaplıkları, uygulamalar, kaynaklar, testler, resimler ve örnekler içerebilir. pub.dev sitesi, Google mühendisleri ve Flutter ve Dart topluluğunun üyeleri tarafından geliştirilen ve uygulamanızda kullanabileceğiniz birçok paketi listeler.

Plugin: Eklenti paketi, platform işlevselliğini uygulamaya sunan özel bir paket türüdür. Eklenti paketleri Android (Kotlin veya Java kullanılarak), iOS (Swift veya Objective-C kullanılarak), web, macOS, Windows, Linux veya bunların herhangi bir kombinasyonu için yazılabilir. Örneğin, bir eklenti Flutter uygulamalarına bir cihazın kamerasını kullanma yeteneği sağlayabilir.

Package ve Plugin arasındaki fark nedir?

Plugin, bir paket türüdür. Tam atama, genellikle eklenti olarak kısaltılan *eklenti paketidir* .

Video

Mevcut paketler birçok kullanım örneğini mümkün kılar; örneğin, ağ istekleri yapma (**http**), özel gezinme/rota işleme (**fluro**), cihaz API'leri ile entegrasyon (**url_launcher**) ve Firebase (**FlutterFirebattery**) gibi üçüncü taraf platform SDK'larını kullanma

Flutter SDK'yı Yükseltme

Flutter SDK'yı güncellemek için ***'flutter upgrade'*** komutunu kullanın.

```
$ flutter upgrade
```

Bu komut, Flutter SDK'nın mevcut Flutter kanalınızda bulunan en son sürümünü alır. pubspec.yaml Dosyanızı değiştirdiyseniz veya yalnızca uygulamanızın bağlı olduğu paketleri (hem paketler hem de Flutter'ın kendisi yerine) güncellemek istiyorsanız, ***'flutter pub'*** komutlardan birini kullanın.

- Dosyada listelenen tüm dependencies 'ın en son uyumlu sürümlerini güncellemek için şu komutu kullanın:

```
$ flutter pub upgrade
```

- Güncel olmayan paket dependencies 'nı belirlemek ve bunların nasıl güncelleneceği konusunda tavsiye almak için şu komutu kullanın:

```
$ flutter pub outdated
```

Paket Kullanımı

Paketler pub.dev'de yayınlanır.

Örnek: **css_colors** paketini kullanma
Paket css_colors, CSS renkleri için renk sabitlerini tanımlar.

Bu paketi kullanmak için:

1. Yeni bir proje oluşturun
2. Açın ***pubspec.yaml*** ve ***css-colors*** dependency ekleyin:

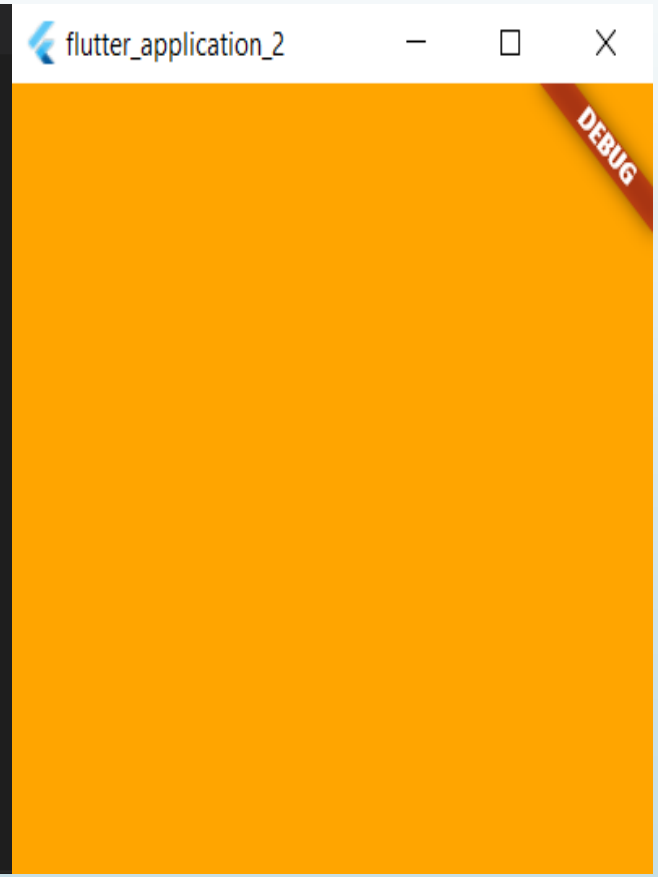
```
dependencies:
  flutter:
    sdk: flutter
  css_colors: ^1.0.0
```

Paket Kullanımı

3. Terminalde ***flutter pub get*** çalıştırın veya IntelliJ veya Android Studio'da ***Packages get*** tıklayın.

4. ***lib/main.dart*** tüm içeriğini açın ve şununla değiştirin: Örnek

```dart
import 'package:css_colors/css_colors.dart';
import 'package:flutter/material.dart';

Run | Debug | Profile
void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({key});

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      home: DemoPage(),
    ); // MaterialApp
  }
}

class DemoPage extends StatelessWidget {
  const DemoPage({key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(body: Container(color: CSSColors.orange));
  }
}
```



5. Uygulamayı çalıştırın. Uygulamanın arka planı artık turuncu olmalıdır.

Cihaz Hafızasına Veri Kaydetme

Flutter uygulamaları , pub.dev'de bulunan eklenti aracılığıyla **SQLite** veritabanlarını kullanabilir. Bu, çeşitli dogs hakkında veri eklemek, okumak, güncellemek ve kaldırmak için kullanmanın temellerini gösterir .

1. Bağımlılıkları Ekleyin

SQLite veritabanlarıyla çalışmak için sqlite ve path paketlerini içe aktarın.

Paket , bir SQLite veritabanıyla etkileşim kurmak için sınıflar ve işlevler sağlar.

Paket , veritabanını diskte depolamak için konumu tanımlamak için işlevler sağlar.

```
dependencies:
  flutter:
    sdk: flutter
  sqflite:
  path:
```

Cihaz Hafızasına Veri Kaydetme

Paketleri, üzerinde çalışacağınız dosyaya aktardığınızdan emin olun.

```
import 'dart:async';

import 'package:flutter/widgets.dart';
import 'package:path/path.dart';
import 'package:sqflite/sqflite.dart';
```

Cihaz Hafızasına Veri Kaydetme

2. Dog Veri Modelini Tanımlayın:

Bu örnek için, üç veri parçası içeren bir Dog sınıfı tanımlayın: Her dog'un benzersiz id, name, ve age 'i vardır.

```
class Dog {
  final int id;
  final String name;
  final int age;

  const Dog({
    required this.id,
    required this.name,
    required this.age,
  });
}
```

Cihaz Hafızasına Veri Kaydetme

3. Veritabanını Açın:

Veritabanına veri okumadan ve yazmadan önce, veritabanına bir bağlantı açın. Bu iki adımı içerir:

- Veritabanı dosyasının yolunu tanımlayın. (***getDatabasesPath()***)
- Veritabanını açın. (***openDatabase()***)

```
// Avoid errors caused by flutter upgrade.
// Importing 'package:flutter/widgets.dart' is required.
WidgetsFlutterBinding.ensureInitialized();
// Open the database and store the reference.
final database = openDatabase(
  // Set the path to the database. Note: Using the `join` function from the
  // `path` package is best practice to ensure the path is correctly
  // constructed for each platform.
  join(await getDatabasesPath(), 'doggie_database.db'),
);
```

Cihaz Hafızasına Veri Kaydetme

4. Dogs Tabloyu Oluşturun:

Ardından, çeşitli dog'lar hakkında bilgi depolamak için bir tablo oluşturun. Her biri Dog, bir id, name ve age içerir. Bu nedenle, bunlar dogs tabloda üç sütun olarak temsil edilmektedir.

1. The **id** is a Dart **int**, and is stored as an **INTEGER** SQLite Datatype. It is also good practice to use an **id** as the primary key for the table to improve query and update times.
2. The **name** is a Dart **String**, and is stored as a **TEXT** SQLite Datatype.
3. The **age** is also a Dart **int**, and is stored as an **INTEGER** Datatype.

Not: Bir SQLite veritabanında depolanabilen mevcut Veri Tipleri hakkında daha fazla bilgi için SQLite tıklayın.

```
final database = openDatabase(
  // Set the path to the database. Note: Using the `join` function from the
  // `path` package is best practice to ensure the path is correctly
  // constructed for each platform.
  join(await getDatabasesPath(), 'doggie_database.db'),
  // When the database is first created, create a table to store dogs.
  onCreate: (db, version) {
    // Run the CREATE TABLE statement on the database.
    return db.execute(
      'CREATE TABLE dogs(id INTEGER PRIMARY KEY, name TEXT, age INTEGER)',
    );
  },
  // Set the version. This executes the onCreate function and provides a
  // path to perform database upgrades and downgrades.
  version: 1,
);
```

Cihaz Hafızasına Veri Kaydetme

5. Veritabanına Bir Dog Ekleyin:

Artık çeşitli dog'lar hakkında bilgi depolamak için uygun bir tablo içeren bir veritabanımız vardır.

1. Convert the Dog into a Map
2. Use the insert() method to store the Map in the dogs table.

```
class Dog {
  final int id;
  final String name;
  final int age;

  const Dog({
    required this.id,
    required this.name,
    required this.age,
  });

  // Convert a Dog into a Map. The keys must correspond to the names of the
  // columns in the database.
  Map<String, dynamic> toMap() {
    return {
      'id': id,
      'name': name,
      'age': age,
    };
  }

  // Implement toString to make it easier to see information about
  // each dog when using the print statement.
  @override
  String toString() {
    return 'Dog{id: $id, name: $name, age: $age}';
  }
}
```

```
// Define a function that inserts dogs into the database
Future<void> insertDog(Dog dog) async {
  // Get a reference to the database.
  final db = await database;

  // Insert the Dog into the correct table. You might also specify the
  // `conflictAlgorithm` to use in case the same dog is inserted twice.
  //
  // In this case, replace any previous data.
  await db.insert(
    'dogs',
    dog.toMap(),
    conflictAlgorithm: ConflictAlgorithm.replace,
  );
}
```

```
// Create a Dog and add it to the dogs table
var fido = const Dog(
  id: 0,
  name: 'Fido',
  age: 35,
);

await insertDog(fido);
```

Cihaz Hafızasına Veri Kaydetme

6. Dog'ların Listesini Alın:

Artık bir Dog veritabanında saklandığına göre, belirli bir dog veya tüm dog'ların bir listesi için veritabanını sorgulayın.

Bu iki adımı içerir:

1. Run a query against the dogs table. This returns a List<Map>.
2. Convert the List<Map> into a List<Dog>.

```
// A method that retrieves all the dogs from the dogs table.
Future<List<Dog>> dogs() async {
  // Get a reference to the database.
  final db = await database;

  // Query the table for all The Dogs.
  final List<Map<String, dynamic>> maps = await db.query('dogs');

  // Convert the List<Map<String, dynamic> into a List<Dog>.
  return List.generate(maps.length, (i) {
    return Dog(
      id: maps[i]['id'],
      name: maps[i]['name'],
      age: maps[i]['age'],
    );
  });
}
```

```
// Now, use the method above to retrieve all the dogs.
print(await dogs()); // Prints a list that include Fido.
```

Cihaz Hafızasına Veri Kaydetme

7. Veritabanındaki Bir Dog'u Güncelleyin:

Veritabanına bilgi ekledikten sonra, bu bilgileri daha sonra güncellemek isteyebilirsiniz. Bunu **update()** yöntemi kullanarak yapabilirsiniz.

Bu iki adımı içerir:

1. Convert the Dog into a Map.
2. Use a where clause to ensure you update the correct Dog.

```dart
Future<void> updateDog(Dog dog) async {
  // Get a reference to the database.
  final db = await database;

  // Update the given Dog.
  await db.update(
    'dogs',
    dog.toMap(),
    // Ensure that the Dog has a matching id.
    where: 'id = ?',
    // Pass the Dog's id as a whereArg to prevent SQL injection.
    whereArgs: [dog.id],
  );
}
```

```dart
// Update Fido's age and save it to the database.
fido = Dog(
  id: fido.id,
  name: fido.name,
  age: fido.age + 7,
);
await updateDog(fido);

// Print the updated results.
print(await dogs()); // Prints Fido with age 42.
```

Cihaz Hafızasına Veri Kaydetme

8. Bir Dog'u Veritabanından Silin:

Dog'lar hakkında bilgi eklemeye ve güncellemeye ek olarak, veritabanından dog'ları da kaldırabilirsiniz. Verileri silmek için **delete()** yöntemi kullanın. Silinen kayıtları sınırlamak için bir madde (**where**) sağlamalısınız.

```
Future<void> deleteDog(int id) async {
  // Get a reference to the database.
  final db = await database;

  // Remove the Dog from the database.
  await db.delete(
    'dogs',
    // Use a `where` clause to delete a specific dog.
    where: 'id = ?',
    // Pass the Dog's id as a whereArg to prevent SQL injection.
    whereArgs: [id],
  );
}
```

JSON Parse İşlemleri

İster Android ister iOS olsun eğer bir mobil uygulama yazıyorsak elbet bir yerde JSON parse etmeye, uzak sunucular ile bilgi alışverişi yapmaya ihtiyaç duyarız.

1. http Paketi Ekleyin:

İlk önce http paketi projenize ekleyin. http paket, bir JSON uç noktasından veri alma gibi ağ isteklerini gerçekleştirmeyi kolaylaştırır.

```
dependencies:
  http: <latest_version>
```

2. Bir Ağ İsteğinde Bulunun: Bu örnek , ***http.get()*** metodu kullanarak JSONPlaceholder REST API'sinden 5000 fotoğraf nesnesinin bir listesini içeren büyük bir JSON belgesinin nasıl getirileceğini kapsar.

```
Future<http.Response> fetchPhotos(http.Client client) async {
  return client.get(Uri.parse('https://jsonplaceholder.typicode.com/photos'));
}
```

JSON Parse İşlemleri

3. JSON'u Ayrıştırın ve Bir Fotoğraf Listesine Dönüştürün:

Ardından, İnternetten veri alma rehberliğini izleyerek , bunları bir ***http.Response*** Dart nesneleri listesine dönüştürün. Bu, verilerle çalışmayı kolaylaştırır.

İlk olarak, bir fotoğraf hakkında veri içeren bir sınıf oluşturun. Bir JSON nesnesiyle bir başlangıç oluşturmayı kolaylaştırmak için bir ***fromJson()*** metodu ekleyin.

```
class Photo {
  final int albumId;
  final int id;
  final String title;
  final String url;
  final String thumbnailUrl;

  const Photo({
    required this.albumId,
    required this.id,
    required this.title,
    required this.url,
    required this.thumbnailUrl,
  });

  factory Photo.fromJson(Map<String, dynamic> json) {
    return Photo(
      albumId: json['albumId'] as int,
      id: json['id'] as int,
      title: json['title'] as String,
      url: json['url'] as String,
      thumbnailUrl: json['thumbnailUrl'] as String,
    );
  }
}
```

JSON Parse İşlemleri

Yanıtı bir fotoğraf listesine dönüştürün:

1. Create a parsePhotos() function that converts the response body into a List<Photo>.
2. Use the parsePhotos() function in the fetchPhotos() function.

JSON PARSE

```dart
// A function that converts a response body into a List<Photo>.
List<Photo> parsePhotos(String responseBody) {
  final parsed = jsonDecode(responseBody).cast<Map<String, dynamic>>();

  return parsed.map<Photo>((json) => Photo.fromJson(json)).toList();
}

Future<List<Photo>> fetchPhotos(http.Client client) async {
  final response = await client
      .get(Uri.parse('https://jsonplaceholder.typicode.com/photos'));

  // Use the compute function to run parsePhotos in a separate isolate.
  return parsePhotos(response.body);
}
```

JSON Parse İşlemleri

4. Bu Çalışmayı Ayrı Bir İzoleye Taşıyın:

compute() işlevi, arka plan yalıtımında önemli işlevleri çalıştırır ve sonucu döndürür. Bu durumda, ***parsePhotos()*** işlevi arka planda çalıştırın.

```
Future<List<Photo>> fetchPhotos(http.Client client) async {
  final response = await client
      .get(Uri.parse('https://jsonplaceholder.typicode.com/photos'));

  // Use the compute function to run parsePhotos in a separate isolate.
  return compute(parsePhotos, response.body);
}
```

JSON Parse İşlemleri

4. Bu Çalışmayı Ayrı Bir İzoleye Taşıyın:

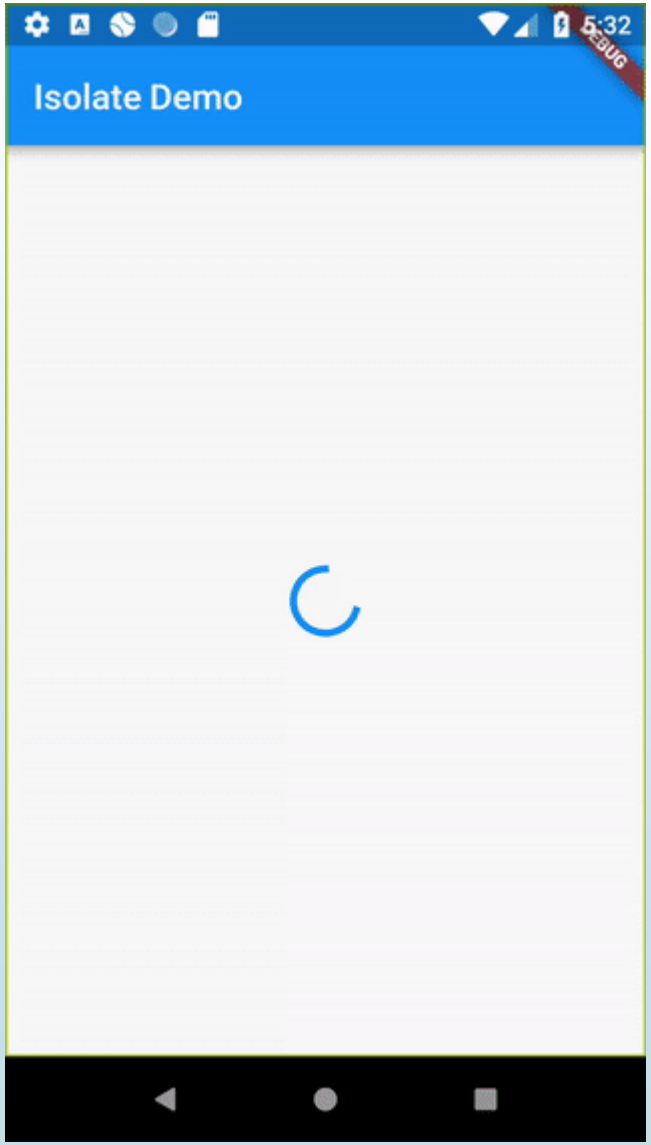
compute() işlevi, arka plan yalıtımında önemli işlevleri çalıştırır ve sonucu döndürür. Bu durumda, ***parsePhotos()*** işlevi arka planda çalıştırın.

```
Future<List<Photo>> fetchPhotos(http.Client client) async {
  final response = await client
      .get(Uri.parse('https://jsonplaceholder.typicode.com/photos'));

  // Use the compute function to run parsePhotos in a separate isolate.
  return compute(parsePhotos, response.body);
}
```

- İzolatlar mesajları ileri geri ileterek iletişim kurarlar. Bu mesajlar; **null**, **num**, **bool**, **double**, veya **string** ya da ***List<Photo>*** bu örnekteki gibi basit nesneler olabilir.

Örnek

```dart
import 'dart:async';
import 'dart:convert';

import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;

Future<List<Photo>> fetchPhotos(http.Client client) async {
  final response = await client
      .get(Uri.parse('https://jsonplaceholder.typicode.com/photos'));

  // Use the compute function to run parsePhotos in a separate isolate.
  return compute(parsePhotos, response.body);
}

// A function that converts a response body into a List<Photo>.
List<Photo> parsePhotos(String responseBody) {
  final parsed = jsonDecode(responseBody).cast<Map<String, dynamic>>();

  return parsed.map<Photo>((json) => Photo.fromJson(json)).toList();
}

class Photo {
  final int albumId;
  final int id;
  final String title;
  final String url;
  final String thumbnailUrl;

  const Photo({
    required this.albumId,
    required this.id,
    required this.title,
```



Yardımcı Kaynaklar

- Adım Adım Flutter İle Mobil Uygulamalar (Rakıcı Oğuz , 2021)
- https://flutter.dev/

İlginiz için teşekkürler…

Hazırlayan : **Zeynep İrem KESLER 1911404048**
E-posta : zeynepiremkesler@gmail.com
Tarih : 01/06/2022
Ders Yürütücüsü : Doç. Dr. İsmail KIRBAŞ
E-posta : ismkir@gmail.com