

Music Player Application

صدیقه سادات غنی

شماره دانشجویی 9773132

استاد راهنما : دکتر نادران

پروژه کارشناسی

Table of Contents

Music Player Application	1
مقدمه	2
هدف پروژه	2
ابزارهای استفاده شده	2
ویژگی های اپلیکیشن	3
بخش آهنگ	3
بخش آفلاین	4
بخش آنلاین	4
توضیحات مخصوص پروژه	4
های پروژه Model	4
Network Files	11
Song Model Provider	13
User Interface صفحات	13
مجوزها	48
منابع	48

مقدمه

اپلیکیشن موزیک پلیر یک اپلیکیشن موبایل با زبان فلاتر است که قابلیت پخش آهنگ ها را به صورت آنلاین از وبسایت شخصی RadioJavan و هم به صورت آفلاین از موزیک های موجود درون دستگاه فراهم میکند.

این اپلیکیشن دارای ویژگی های زیر است :

- امکان پخش آهنگ ها به صورت آنلاین و آفلاین
- قابلیت ساخت و اضافه کردن آهنگ به پلی لیست در بخش آفلاین
- نمایش لیستی از خوانندگان و جدیدترین آهنگ ها و آلبوم ها در بخش آنلاین
- نمایش پلی لیست های مخصوص بخش آنلاین
- نمایش دسته بندی های موزیک در بخش آنلاین

در این گزارش، جزئیات پروژه به همراه روش ها و ابزارهای استفاده شده در پیاده سازی این اپلیکیشن بررسی شده است.

هدف پروژه

هدف اصلی پروژه، ایجاد یک اپلیکیشن پلیر کارآمد برای پخش آهنگ ها با قابلیت آنلاین و آفلاین بودن آن ها است.

ابزارهای استفاده شده

- زبان برنامه نویسی Flutter
- زبان برنامه نویسی Dart
- ابزار Flutter SDK امکان ایجاد رابط کاربری زیبا با استفاده از ویجت های قابل چیدمان را فراهم میکند.
- کتابخانه cupertino_icons برای آیکون های سیستمی
- کتابخانه floating_bottom_navigation_bar برای نوار پایینی برنامه مانند رابط های کاربری برنامه های موبایل
- کتابخانه dio، که یک کتابخانه HTTP است که برای امکان ارسال درخواست های dart مبتنی بر شبکه به سرور استفاده میشود.
- کتابخانه json_annotation برای تبدیل سریع مدل های JSON به Dart و بالعکس

- کتابخانه build_runner امکان اجرای خودکار کد هنگام کامپایل برنامه را فراهم میکند.
- کتابخانه retrofit_generator برای تبدیل اینترفیس های dart به retrofit
- کتابخانه json_serializable برای تبدیل کلاس های JSON به dart و بلعکس.
- کتابخانه cached_network_image امکان دریافت و ذخیره تصاویر در حافظه پنهان را میسر میکند.
- کتابخانه flutter_image_slideshow امکان فراهم کردن یک اسلاید شوی تصویری را ممکن میسازد.
- کتابخانه miniplayer برای پخش ویدیو و صدا در برنامه
- کتابخانه just_audio برای پخش فایل های صوتی با کیفیت بالاتر
- کتابخانه on_audio_query امکان بازگذاری اطلاعات مربوط به فایل های صوتی روی دستگاه را فراهم میکند.
- کتابخانه path برای دسترسی به مسیر فایل ها و دایرکتوری های دستگاه
- کتابخانه flutter_file_manager امکان دسترسی به فایل های دستگاه
- کتابخانه permission_handler تمکان درخواست مجوزهای مورد نیاز برای برنامه را فراهم میکند.
- کتابخانه just_audio_background امکان پخش فایل های صوتی را در پس زمینه فراهم میکند.
- کتابخانه provider امکان ایجاد مدل های مشترک در سراسر برنامه و مدیریت وابستگی های بین آن ها را نشان میدهد.
- کتابخانه Dio یک کتابخانه http برای ارسال درخواست های شبکه به سرور و دریافت پاسخ آن به Restful API میباشد.

ویژگی های اپلیکیشن

پخش آهنگ

این اپلیکیشن دارای دو حالت پخش آنلاین و آفلاین است. با انتخاب حالت مورد نظر، کاربران میتوانند آهنگ های مورد نظر خود را از اینترنت یا ذخیره شده در دستگاه خودشان پخش کنند.

بخش آفلاین

کاربران میتوانند در بخش آفلاین، پلی لیست های خود را بسازند و آهنگ های دلخواه خود را به آن اضافه کنند. همچنین امکان حذف و ویرایش پلی لیست ها و آهنگ های آن در بخش آفلاین نیز وجود دارد.

بخش آنلاین

در این بخش کاربران میتوانند لیستی از خوانندگان و جدیدترین آهنگ ها و آلبوم ها را مشاهده کنند. همچنین آهنگ های موجود در این بخش بر اساس دسته بندی های مختلف قابل مشاهده هستند.

توضیحات مخصوص پروژه

فایل های مخصوص پروژه در مسیر فولدر “\Music Player\Music-Player\” قرار دارد.

در فولدر فعلی سه فایل مخصوص کدهای پروژه (Code)، صفحات، تصاویر و شمای UI استفاده شده (UI) و گزارش پروژه (Report) قرار دارد.

کدهای اصلی پروژه در مسیر Music Player\Music-Player\Code\m_player\lib قرار دارند.

با اجرای فایل main.dart در برنامه Adnroid Studio میتوانید اجرای پروژه را مشاهده نمایید.

Model های پروژه

کلاس های مورد استفاده برای پروژه موزیک پلیئر شامل کلاس آهنگ، خواننده، آلبوم، دسته بندی، پلی لیست و آخرین آهنگ های اضافه شده است.

Music Model

در فایل Music_Model.dart آجبت های کلاس آهنگ شامل موارد زیر است:

- String id
- String cat_id
- String mp3_title
- String mp3_url
- String mp3_thumbnail_b
- String mp3_thumbnail_s
- String mp3_duration
- String mp3_artist
- String cid
- String category_name
- String category_image
- String category_image_thumb

در همین فایل، از تابعی برای تبدیل آبجکت های این فایل به فرمت json وجود دارد که به شرح زیر است :

```
factory Music_Model.fromJson(Map<String, dynamic> json) => _$Music_ModelFromJson(json);
Map<String, dynamic> toJson() => _$Music_ModelToJson(this);
}
```

برای استفاده از این تابع از کتابخانه json_annotation باید استفاده کرد که پیش از استفاده باید در فایل مورد نظر آن را import کرد.

Import “json_annotation/json_annotation.dart”;

Artist Model

در فایل Artist_Model.dart آبجکت های کلاس خواننده شامل موارد زیر است:

- String id
- String artist_name
- String artist_image
- String artist_image_thumb

در همین فایل مانند کلاس قبلی، از تابعی برای تبدیل آبجکت های این فایل به فرمت json استفاده شده است که به شرح زیر است:

```
factory Artist_Model.fromJson(Map<String, dynamic> json) => _$Artist_ModelFromJson(json);
Map<String, dynamic> toJson() => _$Artist_ModelToJson(this);
```

برای استفاده از این تابع از کتابخانه json_annotation باید استفاده کرد که پیش از استفاده باید در فایل مورد نظر آن را import کرد.

Import “json_annotation/json_annotation.dart”;

Category Model

در فایل Category_Model.dart آبجکت های کلاس دسته بندی شامل موارد زیر است:

- String cid
- String category_name
- String category_image
- String category_image_thumb

در همین فایل همانند کلاس های قبلی، از تابعی برای تبدیل آبجکت های این فایل به فرمت json استفاده شده است که به صورت زیر است:

```
factory Category_Model.fromJson(Map<String, dynamic> json) => _$Category_ModelFromJson(json);
Map<String, dynamic> toJson() => _$Category_ModelToJson(this);
```

برای استفاده از این تابع از کتابخانه json_annotation باید استفاده کرد که پیش از استفاده باید در فایل مورد نظر آن را import کرد.

Import “json_annotation/json_annotation.dart”;

Album Model

در فایل Album.Model.dart آبجکت های کلاس آلبوم به صورت زیر هستند:

- String aid
- String album_name
- String album_image
- String album_image_thumb

در همین فایل از تابعی برای تبدیل آبجکت های این فایل به فرمت json استفاده شده است که به صورت زیر است:

```
factory Album_Model.fromJson(Map<String, dynamic> json) => _$Album_ModelFromJson(json);  
Map<String, dynamic> toJson() => _$Album_ModelToJson(this);
```

برای استفاده از این تابع از کتابخانه json_annotation باید استفاده کرد که پیش از استفاده باید در فایل مورد نظر آن را import کرد.

Import “json_annotation/json_annotation.dart”;

Playlist Model

در فایل Playlist_Model.dart آبجکت های کلاس پلی لیست به صورت زیر هستند:

String pid

String playlist_name

String playlist_image

String playlist_image_thumb

در همین فایل از تابعی برای تبدیل آبجکت های این فایل به فرمت json استفاده شده است که به صورت زیر است:

```
factory Playlist_Model.fromJson(Map<String, dynamic> json) => _$Playlist_ModelFromJson(json);  
Map<String, dynamic> toJson() => _$Playlist_ModelToJson(this);
```

برای استفاده از این تابع از کتابخانه json_annotation باید استفاده کرد که پیش از استفاده باید در فایل مورد نظر آن را import کرد.

Import “json_annotation/json_annotation.dart”;

Latest Music Model

در فایل Latest_Music_Model.dart تنها یک آبجکت وجود دارد:

- List<Music_Model> musics

این آبجکت لیستی از آهنگ هایی که به تازگی به وب سایت اضافه شده اند را شامل میشود. همانند کلاس های قبل، این آبجکت را باید به فرمت json نیز تبدیل کنیم :

```
factory Latest_Music_Model.fromJson(Map<String, dynamic> json) => _$Latest_Music_ModelFromJson(json);  
Map<String, dynamic> toJson() => _$Latest_Music_ModelToJson(this);  
}
```

که برای استفاده از این تابع از کتابخانه json_annotation باید استفاده کرد که پیش از استفاده باید در فایل مورد نظر آن را import کرد.

Import “json_annotation/json_annotation.dart”;

در این فایل از دو دستور اضافه تری استفاده شده است:

```

5
6 part 'Latest_Music_Model.g.dart';
7
8 @JsonSerializable()
9 class Latest_Music_Model{
10
11   @JsonKey(name: "ONLINE_MP3")
12   List<Music_Model>? musics;
13
14   Latest_Music_Model(this.musics);
15
16   factory Latest_Music_Model.fromJson(Map<String, dynamic> json) => _Latest_Music_ModelFromJson(json);
17   Map<String, dynamic> toJson() => _Latest_Music_ModelToJson(this);
18 }

```

در خط 8، `@JsonSerializable` یک annotation در فلاتر است که به کلاس ها کمک میکند تا به صورت خودکار به JSON تبدیل شوند.

در خط 11، نیز یک annotation در فلاتر است که به ما امکان میدهد نام یک فیلد را در فرمت JSON به نام دلخواهی که میخواهیم تغییر دهیم.

تمام کلاس های گفته شده در فولدر خود شامل فایل دیگر هستند که اسم فایل همانند خود است ولی به `.g.dart` ختم میشوند.

فایل با پسوند `.g.dart` که با دستور زیر در ترمینال برنامه ساخته میشود، یک فایل تولیدی است که برای `serialazation` و `deserialization` در فلاتر استفاده میشود.

In terminal : `flutter pub run build_runner build`

این فایل توسط `build_runner` ساخته میشود و شامل کدهایی است که برای تبدیل خودکار اشیاء به فرمت های مختلفی مانند `Json`، `XML` و ... به کار میروند.

این دو فایل گفته شده، به عنوان مثال `Music_Model.dart` و `Music_Model.g.dart` با دستور کد زیر به همدیگر کانکت شده اند:

part 'Music_Model.g.dart'; -> in Music_Model.dart file

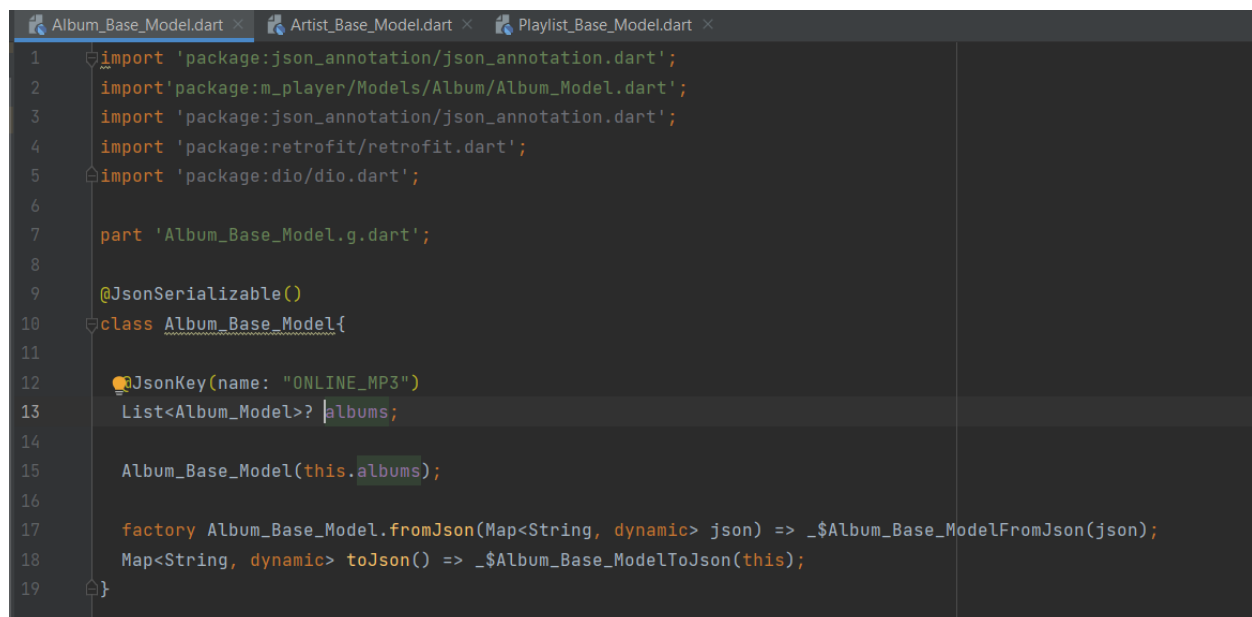
part of 'Music_Model.dart'; -> in Music_Model.g.dart file

بقیه کلاس های گفته شده (آلبوم، دسته بندی، خواننده، آخرین موزیک و پلی لیست) همانند توضیحات بالا میباشند و شامل موارد گفته شده هستند.

Album Base

در فایل Album_Base_Model.dart، تنها یک آبجکت وجود دارد که نشان دهنده لیستی از آلبوم ها است.

```
List<Album_Model> albums;
```



```
1 import 'package:json_annotation/json_annotation.dart';
2 import 'package:m_player/Models/Album/Album_Model.dart';
3 import 'package:json_annotation/json_annotation.dart';
4 import 'package:retrofit/retrofit.dart';
5 import 'package:dio/dio.dart';
6
7 part 'Album_Base_Model.g.dart';
8
9 @JsonSerializable()
10 class Album_Base_Model{
11
12   @JsonKey(name: "ONLINE_MP3")
13   List<Album_Model>? albums;
14
15   Album_Base_Model(this.albums);
16
17   factory Album_Base_Model.fromJson(Map<String, dynamic> json) => _$Album_Base_ModelFromJson(json);
18   Map<String, dynamic> toJson() => _$Album_Base_ModelToJson(this);
19 }
```

همانند کلاس آخرین موزیک، در اینجا هم از JsonSerializable برای تبدیل خودکار کلاس به فرمت JSON استفاده میشود.

Artist & Playlist & Category Base Model

سه کلاس خواننده، پلی لیست و دسته بندی نیز همانند دو کلاس قبلی گفته شده از JsonSerializable برای تبدیل خودکار کلاس به فرمت JSON استفاده میکنند.

هر کدام از این کلاس ها لیستی از آبجکت های کلاس اصی خود به عنوان آبجکت در بر دارند.

هر کدام از این مدل ها نیز با دستور `run build_runner build` یک فایل تولیدی برای هر کدام از فایل ها برای `serialazation` و `deserialization` میسازد.

Network Files

در فایل `Rest_Client.dart` از کتابخانه های `http` برای برقراری اتصال اینترنت به وب سایت مورد نظر و `dio` کتابخانه `http` برای ارسال درخواست های شبکه به سرور و دریافت پاسخ آن به `Restful API` استفاده میشوند.

```
Album_Base_Model.dart x Rest_Client.dart x Song_Model_Provider.dart x Artist_Base_Model.dart x Playlist_Base_Model.dart x
8
9 part 'Rest_Client.g.dart';
10
11 @RestApi(baseUrl: 'http://mobilemasters.ir/apps/radiojavan/')
12 abstract class Rest_Client{
13
14   factory Rest_Client(Dio dio, {String baseUrl}) = _Rest_Client;
15
16   @GET('api.php?latest')
17   Future<Latest_Music_Model> getLatestMusics();
18
19   @GET('api.php?album_list')
20   Future<Album_Base_Model> getAlbums();
21
22   @GET('api.php?playlist')
23   Future<Playlist_Base_Model> getPlayLists();
24
25   @GET('api.php?recent_artist_list')
26   Future<Artist_Base_Model> getRecentArtists();
27
28   @GET('api.php?cat_list')
29   Future<Category_Base_Model> getCategories();
30
31   @GET('api.php')
32   Future<Latest_Music_Model> getMusicsByCategory(@Query("cat_id") String id);
33
34   @GET('api.php')
35   Future<Latest_Music_Model> getMusicsByAlbum(@Query("aid") String id);
36
37   @GET('api.php')
38   Future<Latest_Music_Model> getMusicByPlayLists(@Query("pid") String id);
39
```

همانطور که در عکس بالا مشخص میباشد، در خط 11، آدرس وب سایت مورد نظر را به عنوان آدرس پایه URL قرار میدهیم که آدرس زیر میباشد:

<http://mobilemasters.ir/apps/radiojavan/>

این کد یک factory constructor در Dart است که یک کلاس با نام Rest_Client را برمیگرداند. در این تابع، دو پارامترهای ورودی dio و baseUrl وجود دارند. Dio پارامتری است که به عنوان مدیریت شبکه در این کلاس استفاده میشود و پارامتر baseUrl یک رشته است که حاوی آدرس پایه برای ارتباط با سرور استفاده میشود.

در خط هایی که کلمه کلیدی @GET وجود دارد، از آن برای گرفتن آدرس مخصوص URL ایی که میخواهیم از آن ها برای نمایش آهنگ و ... استفاده کنیم، استفاده میشود.

به عنوان مثال در دو خط 16 و 17، که برای گرفتن آخرین آهنگ های اضافه شده است، این کد یک تابع به نام getLatestMusics را تعریف میکند که با استفاده از آن، درخواست GET را به Latest_Music_Model به یک آدرس خاص ('api.php?latest') ارسال میکند و منتظر دریافت یک شی از نوع عنوان پاسخ از سرور می ماند.

در خط 19، آدرس 'api.php?album_list' برای گرفتن لیست تمام آلبوم ها میباشد.

در خط 22، آدرس 'api.php?playlist' برای گرفتن لیست تمام پلی لیست های آنلاین میباشد.

در خط 25، آدرس 'api.php?recent_artist_list' برای گرفتن لیست تمام خواننده های اخیر میباشد.

در خط 28، آدرس 'api.php?cat_list' برای گرفتن لیست تمام دسته بندی های موجود میباشد.

در سه تابع بعدی در خط های 31، 34 و 37 تنها از ادرس 'api.php' استفاده شده است. زیرا در این سه تابع از دستور Query برای گرفتن خروجی به خصوصی استفاده شده است. به عنوان مثال در خط 32، طبق cat_id (دسته بندی) که به عنوان ورودی به تابع getMusibByCategory داده میشود، موزیک های همان دسته بندی مورد نظر را به نمایش نشان میدهد.

Song Model Provider

در فایل Song_Model_Provider.dart که به صورت زیر است :

```
Song_Model_Provider.dart x
1  import 'package:flutter/material.dart';
2
3  class Song_Model_Provider with ChangeNotifier{
4
5      int _id = 0;
6      int get id => _id;
7
8      void setId(int id){
9          _id = id;
10         notifyListeners();
11     }
12
13 }
```

این کد یک کلاس به نام Song_Model_Provider را تعریف میکند که به عنوان یک provider در فلاتر مورد استفاده قرار میگیرد. Provider ها در فلاتر برای به اشتراک گذاشتن داده ها بین چندین ویجت مورد استفاده قرار میگیرند.

در خط 10، تابع notifyListeners() برای به روز رسانی ویجت های متصل به یک provider استفاده میشود. با فراخوانی این تابع، همه ویجت هایی که به این provider متصل هستند، اعلام میکنند که اطلاعات موجود در provider تغییر کرده و باید بازنویسی شوند. به این ترتیب، بازنویسی و رفرش صفحه با اطلاعات جدید صورت میگیرد و این عمل به روزرسانی داده ها را در این صفحه منجر میشود.

صفحات User Interface

صفحه اصلی main

برای اجرای پروژه از فایل آغازین یعنی main.dart استفاده میکنیم.

در هنگام اجرای کد، صفحه اصلی از فایل main.dart وارد فایل دیگری به نام Splash_Screen.dart میشود.

به علت این که صفحه اصلی برنامه دارای دیتاهایی است که از وب سرویس گرفته میشوند است، باید تا مدت زمان بارگیری شدن داده ها صبر کرده و سپس به صفحه جدیدی که در فایل Dashboard_Screen.dart است برود. نمونه کد به صورت زیر است:

```
main.dart x Splash_Screen.dart x Dashboard_Screen.dart x
1 import 'package:flutter/material.dart';
2 import 'package:m_player/UI/Screens/Dashboard/Dashboard_Screen.dart';
3
4 class SplashScreen extends StatelessWidget {
5   const SplashScreen({Key? key}) : super(key: key);
6
7   @override
8   Widget build(BuildContext context) {
9
10    Future.delayed(Duration(seconds: 3)).then((value){
11      Navigator.pushAndRemoveUntil(
12        context,
13        MaterialPageRoute(builder: (context) => DashboardScreen()),
14        (route) => false
15      );
16    });
17
18    return Scaffold();
19  }
20 }
21
```

صفحه اصلی Dashboard

صفحه اصلی ایی که در برنامه نمایان میشود، فایل Dashboard_Screen.dart میباشد. این صفحه شامل یک BottomNavigationBar میباشد که به 4 بخش Home, Category, Playlist, Device تقسیم بندی شده است. که هر کدام از این بخش ها در فایل جداگانه ایی قرار دارند.

در ابتدای فایل لیستی از جنس Widget ساخته ایم که شامل صفحه هایی است که از طریق BottomNavigationBar قرار است به آن ها دسترسی داشته باشیم که به صورت زیر است :

```
int _currentIndex = 0;
List<Widget> bodyScreens = [
  HomeScreen(),
  CategoryScreen(),
  Playlists(),
  Device_Screen_New(),
  // DeviceScreen()
];
```

طریقه استفاده از BottomNavigationBar به صورت زیر است:

```
44 bottomNavigationBar: FloatingNavbar(
45   onTap: (value){
46     setState(() {
47       _currentIndex = value;
48     });
49   },
50   currentIndex: _currentIndex,
51   backgroundColor: myColors.white,
52   selectedItemColor: myColors.green,
53   selectedBackgroundColor: myColors.white,
54   unselectedItemColor: Colors.black,
55   items: [
56     FloatingNavbarItem(
57       icon: Icons.home,
58       title: "Home"
59     ), // FloatingNavbarItem
60
61     FloatingNavbarItem(
62       icon: Icons.category,
63       title: "Category"
64     ), // FloatingNavbarItem
```


از خط 50 تا 54 میتوان تغییرات مربوط که انتخاب صفحات و تغییرات آیکون ها را اعمال کرد. برای هر صفحه ایی که میخواهیم به آن دسترسی داشته باشیم، باید آیتمی از جنس FloatingNavbarItem (در خط های 56، 61، 66 و 71) بسازیم و آیکون و تیتر مورد نظر مربوط که آن صفحات را بنویسیم.

صفحه Home

این صفحه در فایل Home_Screen.dart قرار دارد که شامل بخش های زیر است:

- نمایش اسلاید شوی پلی لیست ها
- نمایش لیست جدیدترین آهنگ ها
- نمایش لیست آلبوم ها
- نمایش لیست خوانندگان

برای نمایش هر یک از لیست های بالا به متغیر های زیر نیاز است :

```
27
28     final dio = Dio();
29     late Rest_Client rest_client;
30     late Future<Latest_Music_Model> getLatestMusics;
31     late Future<Album_Base_Model> getAlbums;
32     late Future<Playlist_Base_Model> getPlaylists;
33     late Future<Artist_Base_Model> getResentArtist;
34
35     bool isPlay = false;
36
37     late Music_Model currentMusic;
38
39     bool currantStatePlay = false;
40
41     final player = AudioPlayer();
42
43     Duration _duration = const Duration();
44     Duration _position = const Duration();
45
```

دو خط اول 28 و 29، متغیرهای dio و rest_client برای اتصال به وب سرویس و دریافت لیست ها ساخته شده اند. به دلیل گرفتن لیست هایی مانند آخرین موزیک، آلبوم، پلی لیست و خوانندگان، از وب سرویس و اتصال به اینترنت، نوع لیست آن ها باید از جنس Future باشند و جنس لیست آن ها از Base_Model آبجکت مورد نظر باشد.

متغیر های تعریف شده بعدی برای استفاده از به هنگام اجرای آهنگ، دسترسی به اطلاعات آهنگ تازه اجرا شده و player برای پخش آهنگ مورد نظر است.

دو متغیر آخر duration_ و position_ برای نمایش ابتدا و انتهای مکان صوتی آهنگ یا seek استفاده میشوند.

```
46 @override
47 void initState() {
48   // TODO: implement initState
49   super.initState();
50   rest_client = Rest_Client(dio);
51   getLatestMusics = rest_client.getLatestMusics();
52   getAlbums = rest_client.getAlbums();
53   getPlaylists = rest_client.getPlaylists();
54   getRecentArtist = rest_client.getRecentArtists();
55 }
56
57 loadMusic() async {
58   await player.setUrl(currentMusic.mp3_url!);
59 }
60
```

در نمونه کد بالا، تابع initState در هنگام اجرای برنامه و ساختن صفحه اجرا میشود. ابتدا کلاس Rest_Client ورودی ایی به نام dio گرفته و از طریق آن به وب سرویس درخواست میدهد. سپس متغیر لیست های گفته شده در قسمت قبل از طریق متغیر rest_client به توابع مورد نظر برای گرفتن داده های خود دسترسی میگیرند.

تابع بعدی loadMusic برای بارگیری لینک آهنگ مورد نظر برای اجرا استفاده میشود.

در ابتدای اجرای تابع اصلی این صفحه، برای نمایش لیستی از پلی لیست ها، آخرین آهنگ ها، آلبوم ها و خوانندگان، از 4 تا ویجت FutureBuilder طبق نوع آن لیستی مورد نظر استفاده میشود.

```
Dashboard_Screen.dart x Home_Screen.dart x
62 Widget build(BuildContext context) {
63   return Scaffold(
64     backgroundColor: myColors.white,
65     body: Stack(
66       children: [
67         // Home Screens
68         Container(
69           padding: EdgeInsets.only(bottom: 70),
70           child: SingleChildScrollView(
71             child: Column(
72               children: [
73                 FutureBuilder<Playlist_Base_Model>(...), // FutureBuilder
74                 FutureBuilder<Latest_Music_Model>(...), // FutureBuilder
75                 FutureBuilder<Album_Base_Model>(...), // FutureBuilder
76                 FutureBuilder<Artist_Base_Model>(...) // FutureBuilder
77               ],
78             ), // Column
79           ), // SingleChildScrollView
80         ], // Container
81         // when Music is Playing
82         if (isPlay == true)
83           Container(...) // Container
84       ],
85     ),
86   );
```

که توضیح هر کدام از این موارد به شرح زیر است.

Playlist Slide Show

در ابتدای صفحه Slide Show ایی وجود دارد که نشان دهنده لیست پلی لیست های وب سرویس است.



کد های این قسمت به صورت زیر است :

```
Dashboard_Screen.dart x Home_Screen.dart x
75 FutureBuilder<Playlist_Base_Model>(  
76   future: getPlaylists,  
77   builder: (context, snapshot){  
78     if (snapshot.hasData){  
79       List<Widget> imagesList = [];  
80       for(int i=0; i<snapshot.data!.playlists!.length; i++){  
81         imagesList.add(CachedNetworkImage(  
82           imageUrl: '${snapshot.data!.playlists![i].playlist_image}',  
83           width: double.infinity,  
84           height: 200,  
85           errorWidget: (context, url, error) => Center(child: Icon(Icons.error, color: myColors.darkGreen)),  
86           placeholder: (context, url) => Center(child: CircularProgressIndicator()),  
87           fit: BoxFit.cover,  
88         )); // CachedNetworkImage  
89       }  
90       return Container(  
91         height: 200,  
92         child: ImageSlideshow(  
93           width: double.infinity,  
94           height: 200,  
95           initialPage: 0,  
96           indicatorColor: myColors.yellow,  
97           indicatorBackgroundColor: myColors.white,  
98           indicatorRadius: 3,  
99           autoPlayInterval: 3000, // 3s  
100          isLoop: true,  
101          onPageChanged: (value) {  
102            print('Page changed: $value');  
103          },  
104          children: imagesList,  
105        ), // ImageSlideshow
```

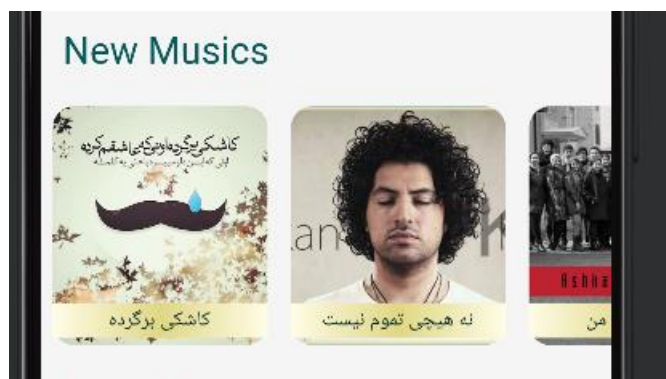
برای نشان دادن لیست هایی با داده هایی که از وب سرویس گرفته میشوند احتیاج به ویجت <Future> میباشد که کلاس درون آن باید از جنس داده هایی باشد که در ابتدای کد متغیر آن ها را ساختیم (Future<Playlist_Base_Model> getPlayllists).

در صورت نبودن دیتا یا مشکل اتصال به اینترنت علامت لودینگ نمایش داده میشود، ولی در غیر این صورت به تعداد پلی لیست های موجود در وب سرویس، از طریق CachedNetworkImage تصاویر آنها نمایش داده میشوند.

سپس به مدت 3 ثانیه، تصویر عوض شده و تصویر پلی لیست دیگری نمایش داده میشود.

Latest Music List

بعد از slide show پلی لیست ها، لیستی از آخرین آهنگ های آپلود شده در سایت میباشند:



برای گرفتن لیست آهنگ هایی که در وب سرویس وجود دارند، لیستی از نوع Future و جنس Latest_Music_Model در ابتدای برنامه میسازیم. سپس در تابعی که به هنگام اجرای برنامه اجرا شده، به آن مقدار دهی میدهم که از وب سرویس آهنگ های مورد نظر را دریافت کند. سپس در ListView ایی از نوع همان لیستی که تعریف کردیم، Future، از ویجتی به نام FutureBuildre استفاده میکنیم. به عنوان دو ورودی اصلی future و builder، که ورودی اول برای لیست آهنگ های گرفته شده از وب میباشد و ورودی دوم برای نمایش لیست میباشد.

کد های این بخش به صورت زیر است:

```

Dashboard_Screen.dart x Home_Screen.dart x
121 ●
122 ●
123 ●
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161

FutureBuilder<Latest_Music_Model>({
  future: getLatestMusics,
  builder: (context, snapshot){
    if (snapshot.hasData){
      return Container(
        //...
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            // New Music Text
            Container(...), // Container

            Container(
              height: 180,
              child: ListView.builder(
                shrinkWrap: true,
                itemCount: snapshot.data!.musics!.length,
                scrollDirection: Axis.horizontal,
                itemBuilder: (context, index){
                  return InkWell(
                    child: CachedNetworkImage(
                      width: 164,
                      height: 164,
                      imageUrl: "${snapshot.data!.musics![index].mp3_thumbnail_b}",
                      imageBuilder: (context, imageProvider) => Container(
                        margin: EdgeInsets.all(8),
                        decoration: BoxDecoration(
                          borderRadius: BorderRadius.circular(15.0),
                          image: DecorationImage(
                            image: imageProvider,
                            fit: BoxFit.cover,

```

```

Dashboard_Screen.dart x Home_Screen.dart x
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191

fit: BoxFit.cover,
), // DecorationImage
), // BoxDecoration
child: Stack(
  children: [
    Positioned(
      child: Container(
        decoration: BoxDecoration(
          gradient: LinearGradient(
            colors: [
              Colors.yellow.shade50,
              myColors.yellow,
              Colors.yellow.shade50
            ],
            stops: [0, 0.5, 1]
          ), // LinearGradient
        ), // BoxDecoration
        child: Center(
          child: Text(
            "${snapshot.data!.musics![index].mp3_title}",
            style: TextStyle(
              fontSize: 14,
              color: myColors.darkGreen
            ), // TextStyle
          ), // Text
        ), // Center
      ), // Container
    ),
  ],
  bottom: 5,
  right: 0,
  left: 0,
) // Positioned
],

```

در صورت نبود آهنگ ها و یا مشکل در برقراری اینترنت، به کاربر اطلاع داده میشود.

```
194 placeholder: (context, url) => CircularProgressIndicator(),
195 errorWidget: (context, url, error) => Icon(Icons.error),
196 ), // CachedNetworkImage
197 onTap: (){
198   setState() {
199     isPlay = true;
200     currentStatePlay = true;
201     currentMusic = snapshot.data!.musics![index];
202   });
203 },
204 ); // InkWell
205 },
206 ), // ListView.builder
207 ) // Container
208 ],
209 ), // Column
210 ); // Container
211 }
212 else if (snapshot.hasError){
213   return Center(
214     child: Text("Error accured. Please check your connection."),
215   ); // Center
216 }
217 else{
218   return Center(
219     child: CircularProgressIndicator(),
220   ); // Center
221 }
```

در ابتدای برنامه متغیرهایی تعریف شدند به نام های

- bool isPlay = false;
- late Music_Model currentMusic;
- bool currentStatePlay = false;

همانطور که در چند صفحه قبل گفته شده، متغیر isPlay برای زمانی است که حالت اجرای آهنگ را نشان میدهد و هنگامی که false است، آهنگ در حال اجرا نیست و صفحه Home نمایش داده میشود.

برای انجام چنین کاری، کد مربوطه به صورت زیر است:

```

60
61 @override
62 Widget build(BuildContext context) {
63   return Scaffold(
64     backgroundColor: myColors.white,
65
66     body: Stack(
67       children: [
68         // Home Screens
69         Container(...), // Container
70
71         // when Music is Playing
72         if (isPlay == true)
73           Container(...) // Container
74       ],
75     ), // Stack
76   ); // Scaffold

```

متغیر دوم، `currentMusic`، برای زمانی است که وقتی موزیک مورد نظر اجرا میشود، اطلاعات آن در این متغیر ذخیره شود و در هنگام نمایش موزیک از آن استفاده شود.

متغیر سوم، `currentStatePlay`، زمانی استفاده میشود که موزیک مورد نظر اجرا شود و صفحه اصلی تغییر کند.

هنگامی که آیتمی از لیست آخرین آهنگ انتخاب کنیم، مقادیر این متغیرها عوض میشود.

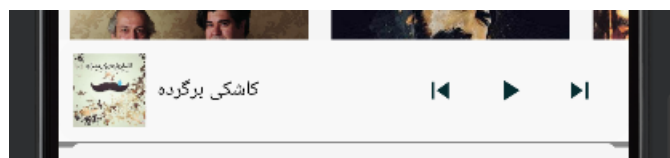
```

198 onTap: (){
199   setState() {
200     isPlay = true;
201     currentStatePlay = true;
202     currentMusic = snapshot.data!.musics![index];
203   });
204 },

```


و هنگامی که آهنگ اجرا شود، صفحه تغییر کرده و صفحه مخصوص با مشخصات آهنگ پلای شده به نمایش گذاشته میشود.

که ابتدا آهنگ به صورت miniPlayer نمایش داده میشود:



و با کلیک بر روی آهنگ تصویر به صورت بزرگ نمایش داده میشود:



برای اجرای miniPlayer ابتدا باید برای ویجت مورد نظر طولی در نظر بگیریم که در اینجا کمترین طول برای پلیر 70 میباشد.

اگر طول miniPlayer 70 بود، پلیر به صورت miniPlayer در پایین صفحه نمایش داده میشود؛ در غیر این صورت با کلیک بر روی پلیر صفحه فول اسکرین شده و آهنگ دوباره پلای میشود.

برای انجام چنین عملیاتی که به صورت زیر عمل میکند:

```
406 // when Music is Playing
407 if (isPlay == true)
408   Container(
409     margin: EdgeInsets.only(bottom: 70),
410     child: Miniplayer(
411       maxHeight: MediaQuery.of(context).size.height,
412       minHeight: 70,
413       builder: (height, percentage){
414         loadMusic();
415         if (height == 70){
416           return Container(
417             decoration: BoxDecoration(
418               color: myColors.white.withOpacity(0.5),
419               borderRadius: BorderRadius.circular(15)
420             ), // BoxDecoration
421             child: Container(
422               margin: EdgeInsets.only(left: 10),
423               child: Row(
424                 mainAxisAlignment: MainAxisAlignment.spaceBetween,
425                 children: [
426                   Row(
427                     children: [
428                       CachedNetworkImage(
429                         imageUrl: '${currentMusic.mp3_thumbnail_s}',
430                         height: 50,
431                         width: 50,
432                       ), // CachedNetworkImage
433                       SizedBox(width: 8,),
434                       Text(
435
```

```
433 width: 50,
434   ), // CachedNetworkImage
435   SizedBox(width: 8,),
436   Text(
437     '${currentMusic.mp3_title}',
438     style: TextStyle(
439       color: Colors.black
440     ), // TextStyle
441   ), // Text
442 ],
443 ), // Row
444
445 Row(
446   children: [
447     //...
448     IconButton(
449       onPressed: () async {
450         setState(() {
451           currentStatePlay = !currentStatePlay;
452         });
453         if (currentStatePlay == false){
454           await player.play();
455         }
456         else{
457           await player.pause();
458         }
459       },
460       icon: Icon(currentStatePlay == false ? Icons.pause : Icons.play_arrow, color: myColors.darkGreen,),
461     ), // IconButton
462     //...
463   ],
464   SizedBox(width: 10,)
465 ]
466
```

هنگامی که بر روی تصویر آهنگ کلیک زده میشود تا صفحه فول اسکرین شود، حالت پلی موزیک دوباره تکرار میشود.

در صفحه فول اسکرین آهنگ، تصویر موزیک به صورت تار به عنوان تصویر پس زمینه قرار میگیرد.

کدهای این قسمت از برنامه به صورت زیر است :

```
Dashboard_Screen.dart x Home_Screen.dart
408 if (isPlay == true)
409   Container(
410     margin: EdgeInsets.only(bottom: 70),
411     child: Miniplayer(
412
413       maxHeight: MediaQuery.of(context).size.height,
414       minHeight: 70,
415       builder: (height, percentage){
416         loadMusic();
417         if(height == 70){...}
418       } else{
419         return Container(
420           decoration: BoxDecoration(
421             color: myColors.white.withOpacity(0.5),
422             borderRadius: BorderRadius.circular(15)
423           ), // BoxDecoration
424           //...
425           child: Container(
426             child: CachedNetworkImage(
427               imageUrl: "${currentMusic.mp3_thumbnail_b}",
428               imageBuilder: (context, imageProvider) => Container(
429                 margin: EdgeInsets.all(8),
430                 decoration: BoxDecoration(
431                   borderRadius: BorderRadius.circular(15.0),
432                   image: DecorationImage(
433                     image: imageProvider,
434                     fit: BoxFit.cover,
435                   ), // DecorationImage
436                 ), // BoxDecoration
437               child: ClipRRect(
438                 child: BackdropFilter(
```

```
Dashboard_Screen.dart x Home_Screen.dart
531 child: BackdropFilter(
532   filter: ImageFilter.blur(sigmaY: 10, sigmaX: 10),
533   child: Container(
534     alignment: Alignment.center,
535     color: Colors.grey.withOpacity(0.1),
536     child: Container(
537       child: Column(
538         mainAxisAlignment: MainAxisAlignment.center,
539         crossAxisAlignment: CrossAxisAlignment.center,
540         children: [
541           CachedNetworkImage(
542             width: 264,
543             height: 264,
544             imageUrl: "${currentMusic.mp3_thumbnail_b}",
545             imageBuilder: (context, imageProvider) => Container(
546               margin: EdgeInsets.all(8),
547               decoration: BoxDecoration(
548                 borderRadius: BorderRadius.circular(15.0),
549                 image: DecorationImage(
550                   image: imageProvider,
551                   fit: BoxFit.cover,
552                 ), // DecorationImage
553               ), // BoxDecoration
554             ), // Container
555           placeholder: (context, url) => CircularProgressIndicator(),
556           errorWidget: (context, url, error) => Icon(Icons.error),
557         ], // CachedNetworkImage
558       ), // Column
559     ), // Container
560     SizedBox(height: 16),
561   ), // BackdropFilter
```

```

Dashboard_Screen.dart x Home_Screen.dart x
561
562       Text(
563         '${currentMusic.mp3_title}',
564         style: TextStyle(
565           color: Colors.black,
566           fontWeight: FontWeight.bold,
567           fontSize: 18
568         ), // TextStyle
569       ), // Text
570
571       Text(
572         '${currentMusic.mp3_artist}',
573         style: TextStyle(
574           color: Colors.black,
575           fontSize: 16
576         ), // TextStyle
577       ), // Text
578
579       SizedBox(height: 16,),
580
581       Row(
582         mainAxisAlignment: MainAxisAlignment.center,
583         crossAxisAlignment: CrossAxisAlignment.center,
584         children: [
585           IconButton(
586             onPressed: (){
587               if(player.hasPrevious){
588                 player.seekToPrevious();
589                 print("skip previous if condition");
590               }
591               print("skip previous");
592             },

```

```

Dashboard_Screen.dart x Home_Screen.dart x
594       ), // IconButton
595       SizedBox(width: 30,),
596       IconButton(
597         onPressed: () async {
598           setState() {
599             currantStatePlay = !currantStatePlay;
600           });
601           if(currantStatePlay == false){
602             await player.play();
603           }
604           else{
605             await player.pause();
606           }
607         },
608         icon: Icon(currantStatePlay == true ? Icons.play_arrow : Icons.pause, color: myColors.darkGreen, size: 48,),
609       ), // IconButton
610       SizedBox(width: 30,),
611       IconButton(
612         // onPressed: () async {
613         //   await player.seekToNext();
614         // },
615         onPressed: (){
616           if(player.hasNext){
617             player.seekToNext();
618             print("skip previous if condition");
619           }
620         },
621         icon: Icon(Icons.skip_next, color: myColors.darkGreen, size: 48,),
622       ), // IconButton
623     ],
624   ), // Row

```

در این قسمت از برنامه برای رفتن به آهنگ بعد و یا برگشت به آهنگ قبل، اگر آهنگ مربوطه در انتهای لیست نیست نباشد و یا در ابتدای لیست نباشد، میتوان عملیات های seekNext و seekPrevious را انجام داد.

برای توقف و اجرای آهنگ، هنگامی که دکمه توقف زده میشود حالت اجرا، currentStatePlay، عوض شده و هم آهنگ متوقف میشود و هم علامت پلی موزیک به آیکون توقف عوض میشود.

Album List

در بخش بعدی که برای نمایش آلبوم ها میباشد، همانند لیست آخرین موزیک ها میباشد با این تفاوت که در هنگام انتخاب آلبوم مورد نظر و کلیک بر روی آن به جای پلی آهنگ های آن وارد صفحه جدیدی برای نشان دادن لیست آهنگ های خود میشود.



برای رفتن به صفحه بعدی برای نمایش لیست آهنگ های آلبوم از کد زیر استفاده میکنیم:

```
return GestureDetector(  
  onTap: () {  
    Navigator.push( //category: snapshot.data!.category![index],  
      context,  
      MaterialPageRoute(builder: (context) => Album_Musics_Screen(album: snapshot.data!.albums![index]))  
    );  
  },  
);
```

برای انتقال به صفحه بعدی، کلاس Album_Music_Screen نیاز به یک ورودی میباشد که این ورودی اطلاعات آلبوم مورد نظر میباشد که طبق آن داده های مورد نیاز را بگیریم. طراحی صفحه بعد، لیست آهنگ های آلبوم، به صورت زیر است:



سرتیتر برنامه اسم آلبوم میباشد و در ادامه لیست آهنگ ها به نمایش گذاشته میشود. در این قسمت از برنامه متغیر های استفاده شده همانند قسمت قبل است ولی از توابع جدیدی استفاده میکند.

```

76 playSong(String? uri){
77     try {
78         _audioPlayer.setAudioSource(
79             AudioSource.uri(Uri.parse(uri!))
80         );
81         _audioPlayer.play();
82     } on Exception {
83         log("Error parsing song");
84     }
85 }
86
87 ConcatenatingAudioSource createPlaylist(List<Music_Model> songs){
88     List<AudioSource> sources = [];
89     for (var song in songs){
90         sources.add(AudioSource.uri(Uri.parse(song.mp3_url!)));
91     }
92     return ConcatenatingAudioSource(children: sources);
93 }
94
95 void _updateCurrentPlaySongDetails(int index){
96     setState(() {
97         if(songs.isNotEmpty){
98             currentTitle = songs[index].mp3_title!;
99             currentIndex = index;
100         }
101     });
102 }
103
104 loadMusic() async {
105     await _audioPlayer.setUrl(currentMusic.mp3_url!);
106 }

```

تابع اول، playSong، برای اجرای آهنگ مورد نظر میباشد.

تابع دوم، createPlaylist، لیستی از آهنگ های آلبوم میسازد تا بتوان از آن برای قسمت هایی همچون seekPrevious و seekNext استفاده کرد.

تابع سوم، _updateCurrentPlaysSongDetails، زمانی استفاده میشود که وقتی که به آهنگ بعدی و یا آهنگ قبلی برویم، اطلاعات صفحه نمایش از حمله عکس کاور آهنگ، اسم موزیک و خواننده و زمان آهنگ را عوض کند.

در قسمت بعدی کد که شامل نمایش لیست صفحه میشود، همانند قبل میباشد ولی برخلاف قسمت قبل، اجرای آهنگ در صفحه جدید انجام میشود و در هنگام رفتن به صفحه بعد آهنگ مورد نظر اجرا میشود و اطلاعات آهنگ به صفحه جدید منتقل میشود.

```

157 onTap: (){
158   setState() {
159     isPlaying = true;
160     currantStatePlay = true;
161     currentMusic = snapshot.data!.musics![index];
162   });
163   Navigator.push(
164     context,
165     MaterialPageRoute(builder: (context) =>
166       PlayAlbumMusics(
167         music_model: snapshot.data!.musics![index],
168         audioPlayer: _audioPlayer,
169         list: snapshot.data!.musics!,
170         //FutureList: getMusics,
171       )) // PlayAlbumMusics, MaterialPageRoute
172   );
173   print("Future : ");
174   print(rest_client.getMusicsByAlbum(widget.album.aid!));
175   print("Album : ");
176   print(index);
177 },

```

در ابتدا حالت پخش آهنگ و حالت عوض شده، اطلاعات موزیک فعلی در متغیر `currentMusic` ذخیره شده و سپس این اطلاعات به همراه کنترلر آهنگ پلی شده به صفحه بعد منتقل میشود.

تصویر صفحه بعد به صورت زیر است:



سه دکمه آخر صفحه از سمت چپ به شرح زیر هستند:

- playlist button برای برگشت به پلی لیست قبلی میباشد.
- Repeat button برای اجرای دوباره آهنگ در هنگام پایان آن
- Shuffle button برای اجرای غیر پشت سرهم آهنگ ها

برای اجرای همزمان آهنگ موقع کلیک روی مورد نظر و seekBar برای کنترل موقعیت موزیک از تابع زیر استفاده میکنیم و آن را در تابع initState() استفاده میکنیم:

```
60 widget.audioPlayer.setAudioSource(  
61   AudioSource.uri(  
62     Uri.parse(widget.music_model.mp3_url!),  
63     tag: MediaItem(  
64       id: '${widget.music_model.id}',  
65       displayDescription: '${widget.music_model.mp3_description}',  
66       album: '${widget.music_model.category_name}',  
67       title: '${widget.music_model.mp3_title}',  
68       artUri: Uri.parse('https://example.com/albumart.jpg'),  
69     ),  
70   ),  
71 );  
72 widget.audioPlayer.setAudioSource(createPlaylist(widget.list));  
73 widget.audioPlayer.play();  
74 _isPlaying = true;  
75 _isShuffle = false;  
76 _isRepeat = false;  
77 }  
78 on Exception{  
79   log("Can not parse song.");  
80 }  
81 widget.audioPlayer.durationStream.listen((d) {  
82   setState(() {  
83     _duration = d!;  
84   });  
85 });  
86 widget.audioPlayer.positionStream.listen((p) {  
87   setState(() {  
88     _position = p!;  
89   });  
90 });
```

برای استفاده از seekBar به صورت زیر عمل میکنیم:

تعریف 2 متغیر در کلاس :

```
Duration _duration = const Duration();  
Duration _position = const Duration();
```

اعمال آن ها به audioPlayer برای داشتن اطلاعات زمان موزیک و کنترل آن:

```
widget.audioPlayer.durationStream.listen((d) {
  setState(() {
    _duration = d!;
  });
});
widget.audioPlayer.positionStream.listen((p) {
  setState(() {
    _position = p!;
  });
});
```

استفاده از تابع زیر برای تبدیل زمان به ثانیه:

```
void changeToSeconds(int sec){
  Duration duration = Duration(seconds: sec);
  widget.audioPlayer.seek(duration);
}
```

طریقه استفاده نهایی از این توابع:

```
Row(
  children: [
    Text(_position.toString().split(".")[0]),
    Expanded(
      child: Slider(
        value: _position.inSeconds.toDouble(),
        max: _duration.inSeconds.toDouble(),
        min: Duration(milliseconds: 0).inSeconds.toDouble(),
        activeColor: myColors.yellow,
        inactiveColor: myColors.yellow.withOpacity(0.5),
        onChanged: (value){
          setState(() {
            changeToSeconds(value.toInt());
            value = value;
          });
        },
      ),
    Text(_duration.toString().split(".")[0]),
  ],
),
```

برای وقتی که بخواهیم بعد از اتمام آهنگ، دوباره پلی شود باید حالت loop once را فعال کنیم که به صورت زیر است:


```

      child: Icon(
        Icons.shuffle,
        color: _isShuffel? myColors.yellow : myColors.darkGreen
      ),
    ),
  ),
),
),

```

Artist List

نمایش لیست خوانندگان همانند نمایش لیست آلبوم ها می باشد با این تفاوت که دیتا داده شده به وِجِت Future باید از جنس کلاس Artist_Base_Model باشد و لیستی که با آن می دهیم نیز دیتاهای مربوط به خوانندگان را برگرداند.

صفحه Category

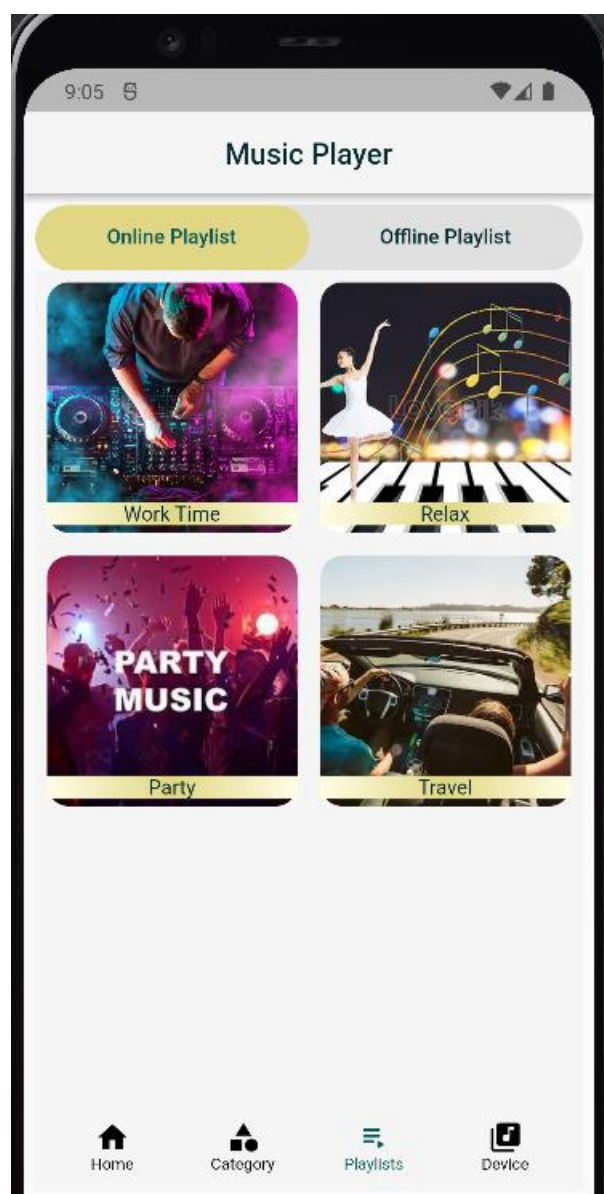
در این صفحه همانند صفحه قبل در قسمت لیست خواننده و آلبوم میبایشد با این تفاوت که جیدمان ویجت های این صفحه با آن ها تفاوت دارد و به صورت زیر است:



این صفحه همانند قسمت لیست آلبوم ها میباشد و با کلیک کردن بر روی دسته بندی مورد نظر، لیستی از آهنگ های آن دسته بندی در صفحه جدید مشخص میشود و با انتخاب آهنگ مورد نظر، موزیک در صفحه جدیدی همانند قبل شروع به پخش شدن میکند.

صفحه Playlist

صفحه اصلی پلی لیست از دو قسمت پلی لیست آنلاین و آفلاین تشکیل شده است.



برای نشان دادن دو قسمت لیست همزمان از ویجت `TabBarView` استفاده میکنیم.

```
36 Container(  
37   height: 45,  
38   decoration: BoxDecoration(  
39     color: Colors.grey[300],  
40     borderRadius: BorderRadius.circular(  
41       25.0,  
42     ), // BorderRadius.circular  
43   ), // BoxDecoration  
44   child: TabBar(  
45     controller: _tabController,  
46     indicator: BoxDecoration(  
47       borderRadius: BorderRadius.circular(25.0,),  
48       color: myColors.yellow,  
49     ), // BoxDecoration  
50     labelColor: myColors.green,  
51     unselectedLabelColor: myColors.darkGreen,  
52     tabs: [  
53       Tab(...), // Online // Tab  
54       Tab(...), // Offline // Tab  
55     ],  
56   ), // TabBar  
57   ), // Container  
58   // tab bar view here  
59   Expanded(  
60     child: TabBarView(  
61       controller: _tabController,  
62       children: [  
63         OnlinePlaylist(),  
64         OfflinePlaylist()  
65       ],  
66     ),  
67   ),  
68 ],  
69 ),  
70
```

```

13 class _PlaylistsState extends State<Playlists>
14     with SingleTickerProviderStateMixin {
15     late TabController _tabController;
16
17     @override
18     void initState() {
19         _tabController = TabController(length: 2, vsync: this);
20         super.initState();
21     }
22
23     @override
24     void dispose() {
25         super.dispose();
26         _tabController.dispose();
27     }
28
29     @override

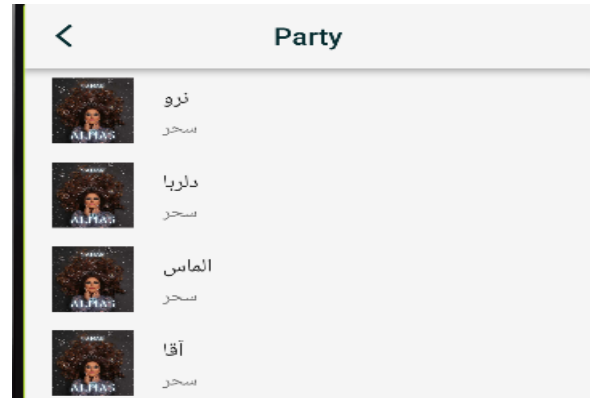
```

در ابتدا کنترلی تعریف شده که باید مقدار طول TabBar ایی که می‌خواهیم استفاده کنیم را مشخص نماییم. سپس نام و یا آیکون های مورد نظر را در قسمت tabs در ویجت TabBar وارد میکنیم. در آخر در ویجت TabBarView به تعداد tabهایی که تعریف کردیم، صفحات مورد نظر را برای آن مشخص میکنیم.

Online Playlist

نمایش پلی لیست آنلین، همانند نمایش لیست های دسته بندی میباشد با این تفاوت لیست گرفته شده از وب سرویس از جنس Playlist_Base_Model میباشد.

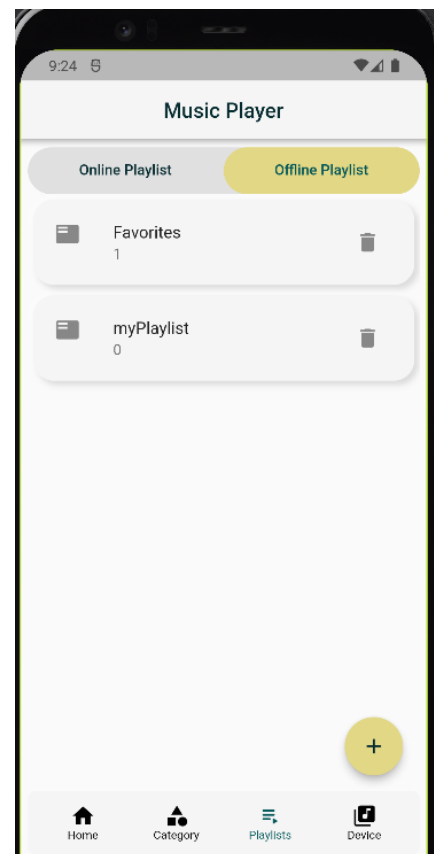
لیست گرفته شده از وب سرویس برای نشان دادن آهنگ های پلی لیست به صورت زیر است:



در هنگام اجرای آهنگ هم همانند قسمت دسته بندی به جای گرفتن id دسته بندی، از id پلی لیست های آنلاین استفاده میکند.

Offline Playlist

صفحه پلی لیست افلاین به صورت زیر است:

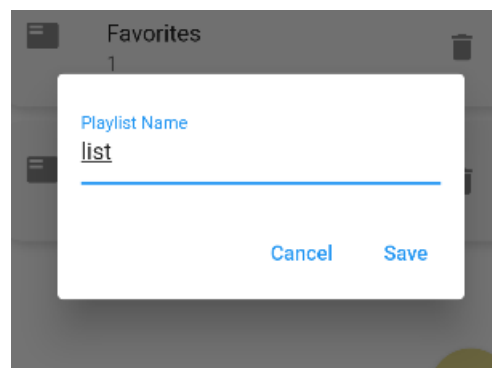


نمایش لیست پلی لیست ها همانند قبل میباشد با است تفاوت که برای گرفتن لیست آن ها از دستگاه از کلاس `OnAudioQuery` استفاده میکنیم؛ و هنگام استفاده از این کلاس به صورت زیر پیش میرویم:

```
future: _audioQuery.queryPlaylists(  
  sortType: null,  
  orderType: OrderType.ASC_OR_SMALLER,  
  uriType: UriType.EXTERNAL,  
  ignoreCase: true  
)
```

طبق دیتاهای داده شده به تابع `queryPlaylists` (برای گرفتن لیست پلی لیست های دستگاه)، نمایش آن ها به ترتیب کاهشی میباشد. آهنگ های گرفته شده از حافظه خارجی میباشد.

با زدن بر روی آیکون اضافه کردن، `AlertDialog`ایی به وجود میآورد که اسم لیستی که میخواهیم را مینویسیم که به صورت زیر است:



نمونه کد استفاده شده برای انجام این کار به صورت زیر است:

```
onPressed: () async {  
  await showDialog<String>(  
    context: context,  
    builder: (BuildContext context) {  
      return AlertDialog(  
        contentPadding: const EdgeInsets.all(16.0),  
        content: new Row(  
          children: <Widget>[  
            new Expanded(  

```

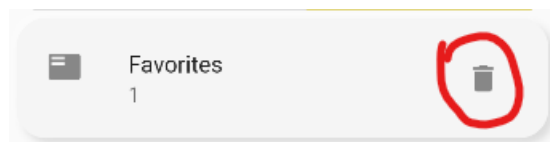
```

        child: new TextField(
          controller: playlistName,
          autofocus: true,
          decoration: new InputDecoration(
            labelText: 'Playlist Name'),
        ),
      ],
    ),
    actions: <Widget>[
      new TextButton(
        child: const Text('Cancel'),
        onPressed: () {
          Navigator.pop(context);
        },
      ),
      new TextButton(
        child: const Text('Save'),
        onPressed: () async {
          setState(() async {
            String name = playlistName.text.toString();
            print(name);
            await OnAudioQuery.platform.createPlaylist(name);
            playlistName.clear();
            Navigator.of(context).pop();
          });
        },
      ),
    ],
  );
},
);
},
};
```

برای ذخیره پلی لیست های ایجاد شده از کتابخانه OnAudioQuery استفاده میکنیم.

```
await OnAudioQuery.platform.createPlaylist(name);
```

سپس برای نمایش پلی لیست های آفلاین از `Future<List<PlaylistModel>>` و `PlaylistModel` استفاده میکنیم که کلاس `PlaylistModel`، کلاسی است برای پلی لیست ها که در کتابخانه `OnAudioQuery` تعریف شده است.

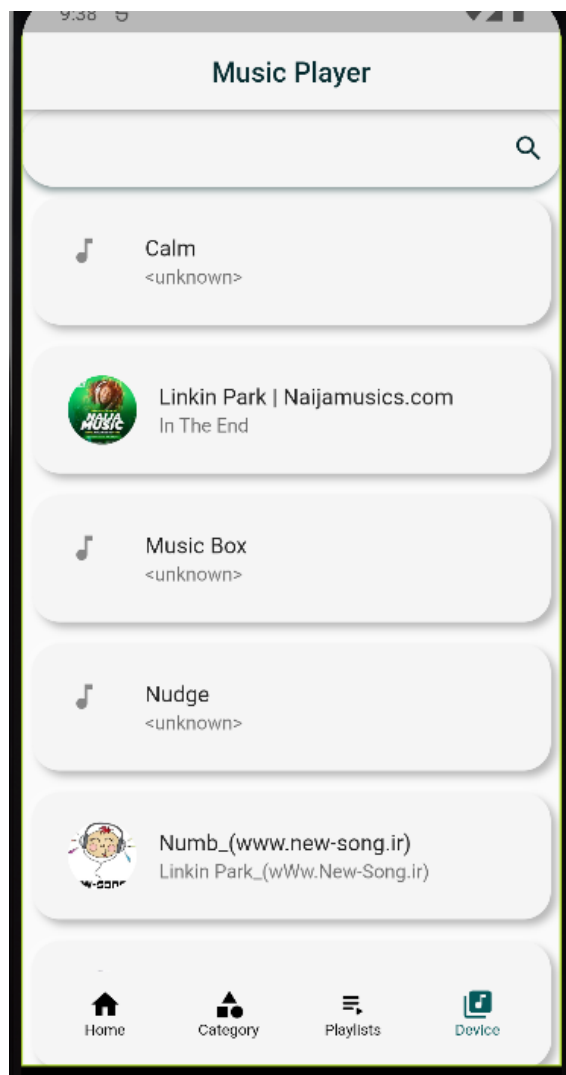


پلی لیست مورد نظر را میتوان حذف کرد که در این صورت آهنگ های درون پلی لیست هم از بین میروند.

نمایش آهنگ های این نوع پلی لیست آفلاین و اجرای آهنگ ها همانند پلی لیست آنلاین میباشد. با این تفاوت که چون برای گرفتن موزیک های دستگاه از کتابخانه OnAudioQuery استفاده میشود، اجرا و نمایش و مدیریت آهنگ ها هم از این کتابخانه استفاده میکنند.

صفحه Device Musics

لیست مخصوص نمایش آهنگ های دستگاه به شکل زیر است :



برای نمایش آهنگ های دستگاه از کتابخانه های `just_audio`، `just_audio_background`، `on_audio_query` استفاده میکنیم.

متغیرهای استفاده شده این قسمت همانند بخش آنلاین میباشد.

در تابع `initState`، که در هنگام اجرای برنامه همزمان اجرا میشود، باید درخواستی از کاربر گرفته شود تا بتوان به موزیک های بخش دستگاه دسترسی داشته باشیم.

```
void requestStoragePermission() async {
  // Permission.storage.request();
  // only if the platform is not web. (web has no permission)
  if(!kIsWeb){
    bool permissionStatus = await _audioQuery.permissionsStatus();
    if(!permissionStatus)
      await _audioQuery.permissionsRequest();
  }
  // Permission.storage.request();

  // ensured build methods is called
  setState(() {

  });
}
```

در شرط اول اگر دستگاهی که برای اجرای برنامه از آن استفاده میکنیم، وب نباشد، درخواست مجوز برای دسترسی به فایل های آهنگ و تصاویر مربوط با آن ها به کاربر داده میشود.

برای استفاده از اطلاعات آهنگ های دستگاه از کلاس آماده `SongModel` استفاده میشود. برای آپلود اطلاعات آهنگ برای اجرا هم از همین کلاس به صورت زیر میتوان استفاده کرد:

```
try {
  _player.setAudioSource(
    AudioSource.uri(
      Uri.parse(songModel.uri!),
      tag: MediaItem(
        id: '${songModel.id}',
        album: '${songModel.album}',
        title: '${songModel.displayNameWOExt}',
        artUri: Uri.parse('https://example.com/albumart.jpg'),
      ),
    ),
  );
}
```

```

    ),
  ),
);
_player.play();
_isPlayerViewVisible = true;
_isShuffel = false;
}
on Exception{
  log("Can not parse song.");
}

```

نمایش لیست و اجرای آهنگ تقریباً همانند قسمت اول، نمایش صفحه Home و اجرای آخرین آهنگ ها، میباشد.

برای لیست آهنگ های دستگاه از ویجت Future استفاده میکنیم ولی نوع کلاسی که به آن میدهم از نوع SongModel میباشد.

برخلاف قسمت های آنلاین، در قسمت future این قسمت که لیست آهنگ ها را میدهد باید به صورت زیر انجام شود:

```

future: _audioQuery.querySongs (
  sortType: null,
  orderType: OrderType.ASC_OR_SMALLER,
  uriType: UriType.EXTERNAL,
  ignoreCase: true
),

```

در اینجا از طریق متغیری که از کلاس OnAudioQuery ساخته شده است، لیست آهنگ ها را با تابع querySongs میگیریم.

برای نمایش تک آهنگ در لیست بر خلاف قسمت آنلاین، به صورت زیر باید باشد:

```

child: ListTile(
  leading: QueryArtworkWidget(
    id: item.data![index].id,
    type: ArtworkType.AUDIO,
    nullArtworkWidget: Icon(Icons.music_note),
  ),
  title: Text(item.data![index].title),
  subtitle: Text("${item.data![index].artist}"),
  // trailing: const Icon(Icons.more_vert),
  onTap: () async {
    _changePlayerViewVisibility();
  }
),

```

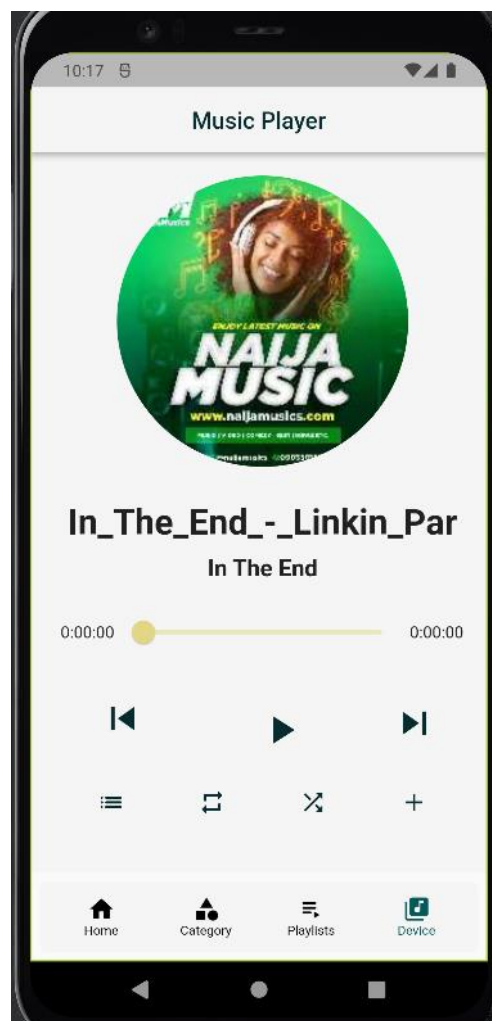
```

        await _player.setAudioSource(createPlaylist(item.data!), initialIndex:
index);
        await _player.play();
        toast(context, item.data![index].title.toString());
    },
),

```

تصویر آهنگ در صورت وجود داشتن، آیکون موزیک به جای آن قرار میگیرد. در هنگام انتخاب و اجرای آهنگ، حالت های آهنگ، `isPlaying`, `currentStatePlay` عوض شده و به مقدار `true` تغییر داده میشوند. در همون حال از طریق `audioSource` پلی لیستی اضافه میشود و آهنگ انتخابی را در آن قرار میدهم تا بتوان برای قسمت `seekNext`, `seekPrevious` استفاده کنیم.

به هنگام اجرای آهنگ صفحه عوض شده و به صورت زیر میباشد:



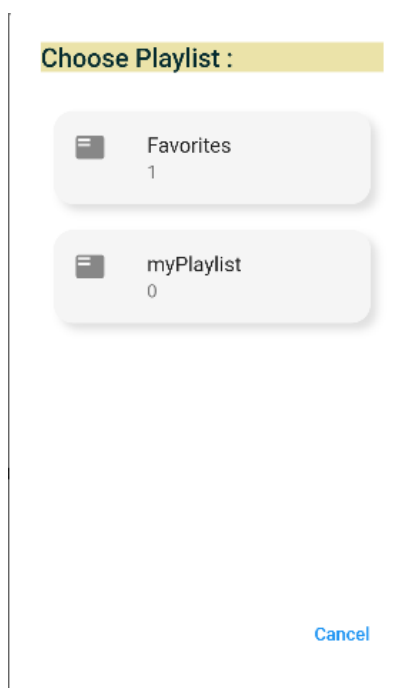
برای گرفتن تصویر آهنگ پلی شده، ویجت زیر را در ستون قسمت بندی صفحه قرار میدهیم:

```
child: QueryArtworkWidget(  
  id: songs[currentIndex].id,  
  type: ArtworkType.AUDIO,  
  artworkBorder: BorderRadius.circular(200.0),  
) ,
```

طبق کد زیر در صورتی که خواننده مشخص نباشد که چه کسی است، کلمه unknown به جای او قرار میگیرد:

```
songs[currentIndex].artist.toString() == "<unknown>" ?  
"Unknown Artist" : songs[currentIndex].artist.toString(),
```

قسمت های تکرار آهنگ و اجرای درهم آهنگ همانند قبل میباشد.
ولی اکنون جدیدی که در این قسمت میباشد، برای اضافه کردن آهنگ پلی شده به پلی لیستی است که از قبل ساخته ایم.



لیست پلی لیست هایی که ساختیم همراه با تعداد آهنگ هایی که دارد برای ما به نمایش گذاشته میشود و میتوان از بین آنها پلی لیست مورد نظر خود را انتخاب کنیم.
با انتخاب پلی لیست از طریق دستور زیر میتوان آهنگ خود را اضافه کرد:

```
await OnAudioQuery.platform.addToPlaylist(playlistId, audioId);
```

مجوزها

در فایل AndroidManifest.xml که در مسیر زیر وجود دارد:

Music Player\Music-Player\Code\m_player\android\app\src\main

باید مجوزهای لازم برای گرفتن و اجرا و در کل مدیریت آهنگ ها را از پیش تعیین کنیم:

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_MEDIA_IMAGES"/>
<uses-permission android:name="android.permission.READ_MEDIA_VIDEO"/>
<uses-permission android:name="android.permission.READ_MEDIA_AUDIO"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
```

در صورت انجام همه کارها ولی موقع اجرای آهنگ به اروری برخورد کردید میتوانید دستور زیر را در همین فایل تغییر دهید:

```
android:name="androidx.multidex.MultiDexApplication"
```

منابع

- https://www.youtube.com/watch?v=UdJ9Ifjptcg&list=PLhGeZ1SOK_3ARkBKRnVjDDZmV2MDi_0N5&index=2
- <https://www.youtube.com/watch?v=aZiuSBpnU5g>
- https://www.youtube.com/watch?v=otIRH2HIFoY&list=PL0kMjh_00eNdleGLZtd9lIXMioypkqCij

- <https://www.youtube.com/watch?v=fWLIbxCNFKo>
- <https://www.youtube.com/watch?v=PnjeNjCq0k4>
- <https://www.youtube.com/watch?v=T4yeX0uGpow>
- https://www.youtube.com/results?search_query=online+music+player+with+flutter
- <https://www.youtube.com/watch?v=QKqLv7uqCE>
- https://www.youtube.com/watch?v=iu7sIJh6LZc&list=PLhGeZ1SOK_3ARkBKRnVjDDZmv2MDi_0N5&index=6
- <http://mobilemasters.ir/apps/radiojavan/index.php>
- <https://stackoverflow.com/questions/50713888/how-to-show-image-from-network-in-flutter-boxdecoration>
- <https://pub.dev/packages/retrofit>
- https://pub.dev/packages/flutter_image_slideshow/install
- https://pub.dev/packages/flutter_text_field_fab/example
- https://pub.dev/packages/searchable_listview
- <https://flutterawesome.com/flutter-plugin-used-to-query-audios-songs-infos-from-device-storage/>
- https://pub.dev/packages/on_audio_query/versions/2.1.1
- <https://api.flutter.dev/flutter/widgets/GridView-class.html>
- <https://android-learn.ir/>

پایان

