# **Music Player Application**

صدیقه سادات غنی

شماره دانشجویی 9773132

استاد راهنما : دکتر نادران

پروژه کارشناسی

Table of Contents

[**Music Player Application** 1](#_Toc134351505)

[مقدمه 2](#_Toc134351506)

[هدف پروژه 2](#_Toc134351507)

[ابزارهای استفاده شده 2](#_Toc134351508)

[ویژگی های اپلیکیشن 3](#_Toc134351509)

[پخش آهنگ 3](#_Toc134351510)

[بخش آفلاین 4](#_Toc134351511)

[بخش آنلاین 4](#_Toc134351512)

[توضیحات مخصوص پروژه 4](#_Toc134351513)

[Model های پروژه 4](#_Toc134351514)

[Network Files 11](#_Toc134351515)

[Song Model Provider 13](#_Toc134351516)

[صفحات User Interface 13](#_Toc134351517)

[مجوزها 48](#_Toc134351518)

[منابع 48](#_Toc134351519)

## 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 cupertion\_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 بلعکس
* 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\_serailizable 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

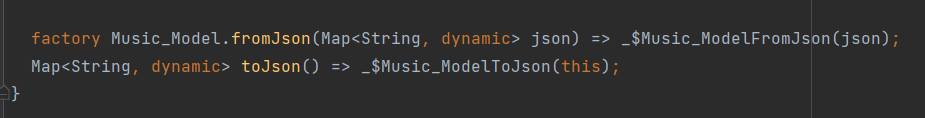
کلاس های مورد استفاده برای پروژه موزیک پلیر شامل کلاس آهنگ، خواننده، آلبوم، دسته بندی، پلی لیست و آخرین آهنگ های اضافه شده است.

#### Music Model

در فایل Music\_Model.dartآبجت 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 :



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.

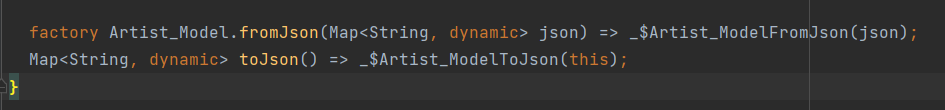
Import “json\_annotation/json\_annotation.dart”;

#### 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:



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.

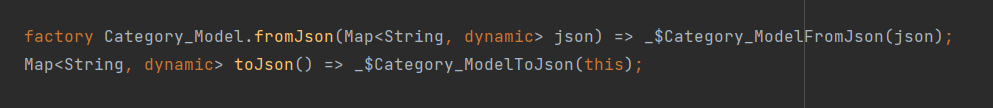
Import “json\_annotation/json\_annotation.dart”;

#### 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:



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.

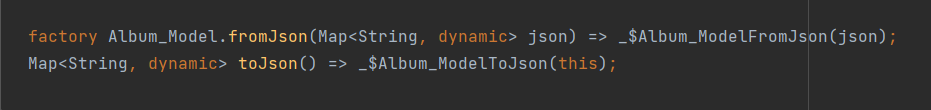
Import “json\_annotation/json\_annotation.dart”;

#### 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:



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:

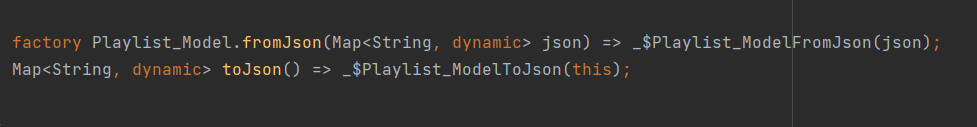
String pid

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:



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”;

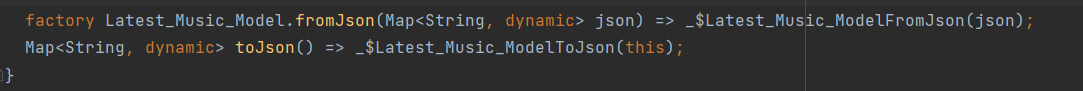
#### Latest Music Model

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.

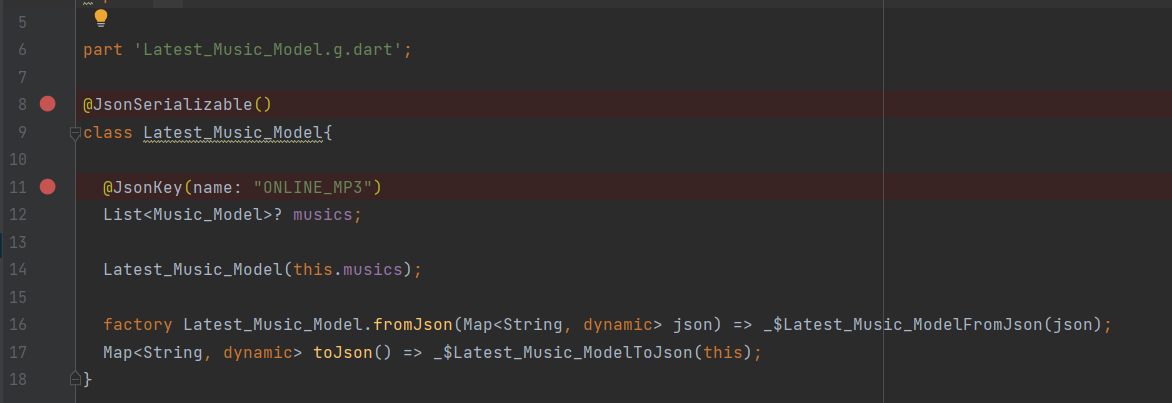
همانند کلاس های قبل، این آبجکت را باید به فرمت json نیز تبدیل کنیم :



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:



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

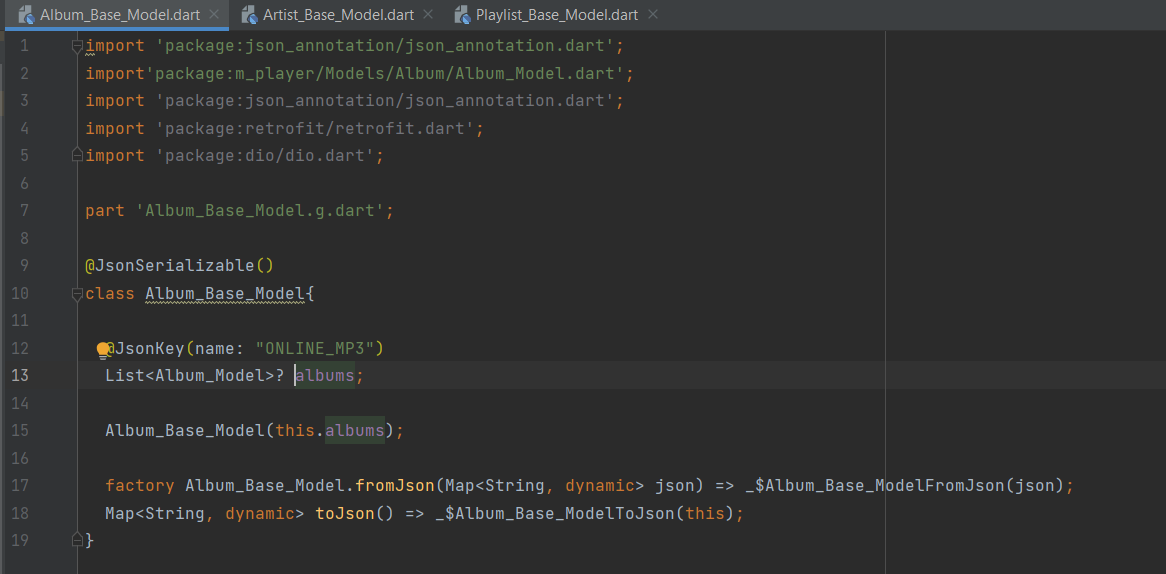
part of ‘Music\_Model.dart’; -> in Music\_Model.g.dart file

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;



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.



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 URLایی 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.

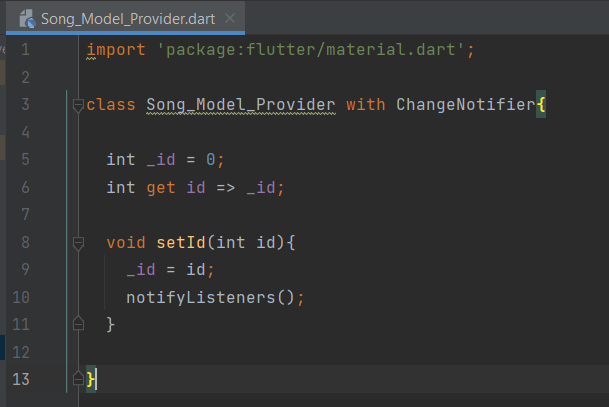
در خط 25، آدرس ‘api.php?recent\_artist\_list’ برای گرفتن لیست تمام خواننده های اخیر میباشد.

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 (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 :



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.

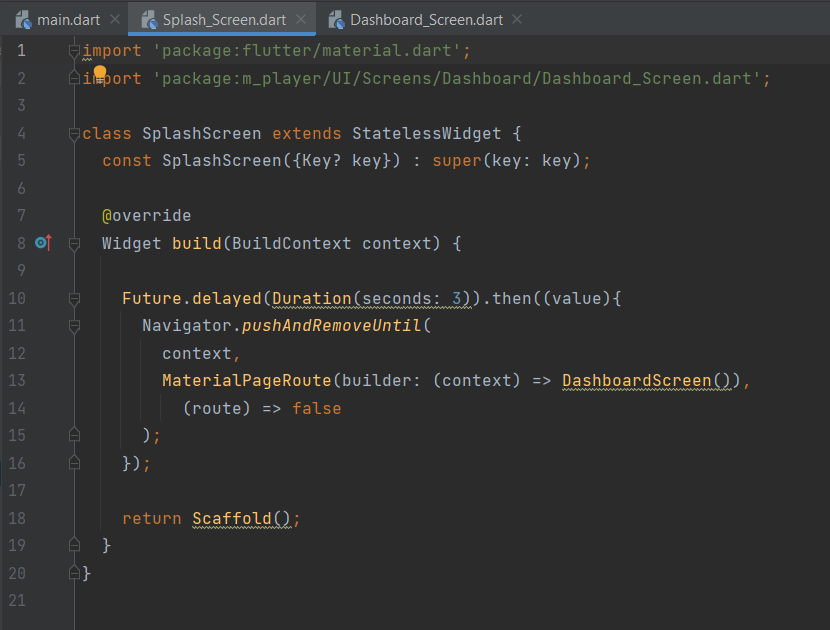
### صفحات User Interface

#### 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.

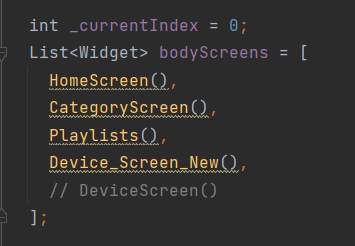
به علت این که صفحه اصلی برنامه دارای دیتاهایی است که از وب سرویس گرفته میشوند است، باید تا مدت زمان بارگیری شدن داده ها صبر کرده و سپس به صفحه جدیدی که در فایل Dashboard\_Screen.dart است برود. نمونه کد به صورت زیر است:



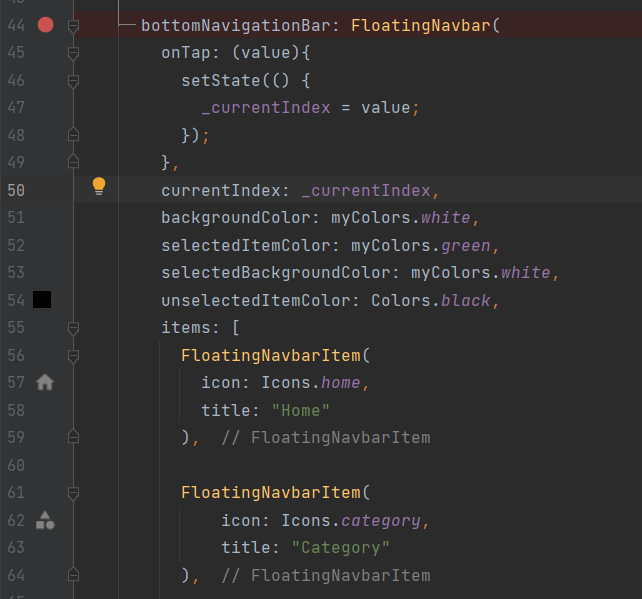
#### صفحه اصلی Dashboard

صفحه اصلی ایی که در برنامه نمایان میشود، فایل Dashboard\_Screen.dart میباشد. این صفحه شامل یک BottonNavigationBar 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 BottonNavigationBar قرار است به آن ها دسترسی داشته باشیم که به صورت زیر است :



طریقه استفاده از ButtonNavigationBar به صورت زیر است:



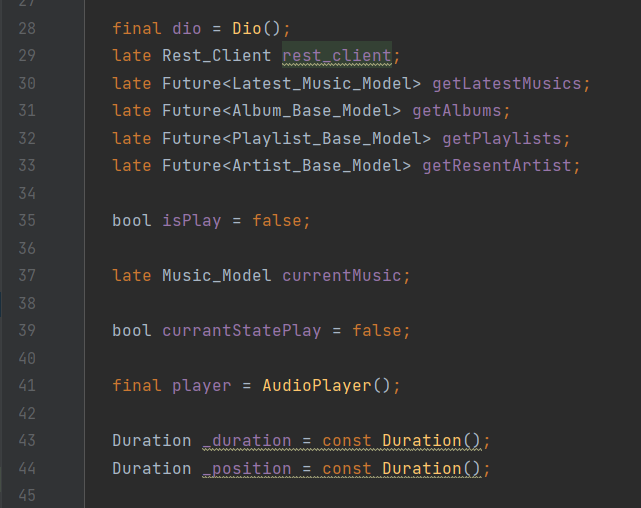
از خط 50 تا 54 میتوان تغیرات مربوط که انتخاب صفحات و تغیرات آیکون ها را اعمال کرد. برای هر صفحه ایی که میخواهیم به آن دسترسی داشته باشیم، باید آیتمی از جنس FloatingNavbarItem (در خط های 56، 61، 66 و 71) بسازیم و آیکون و تیتر مورد نظر مربوط که آن صفحات را بنویسیم.

#### 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

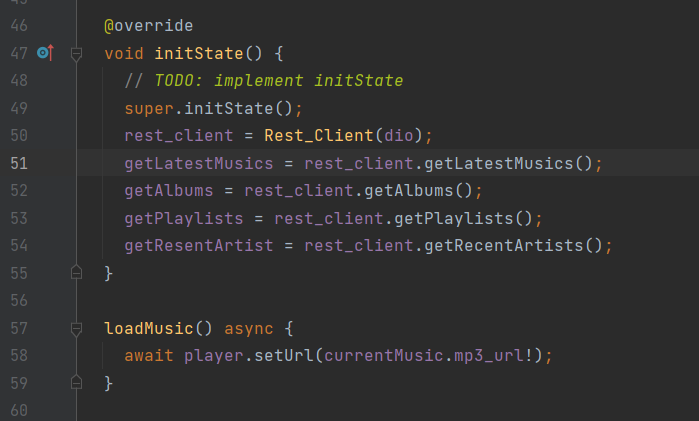
برای نمایش هر یک از لیست های بالا به متغیر های زیر نیاز است :



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.

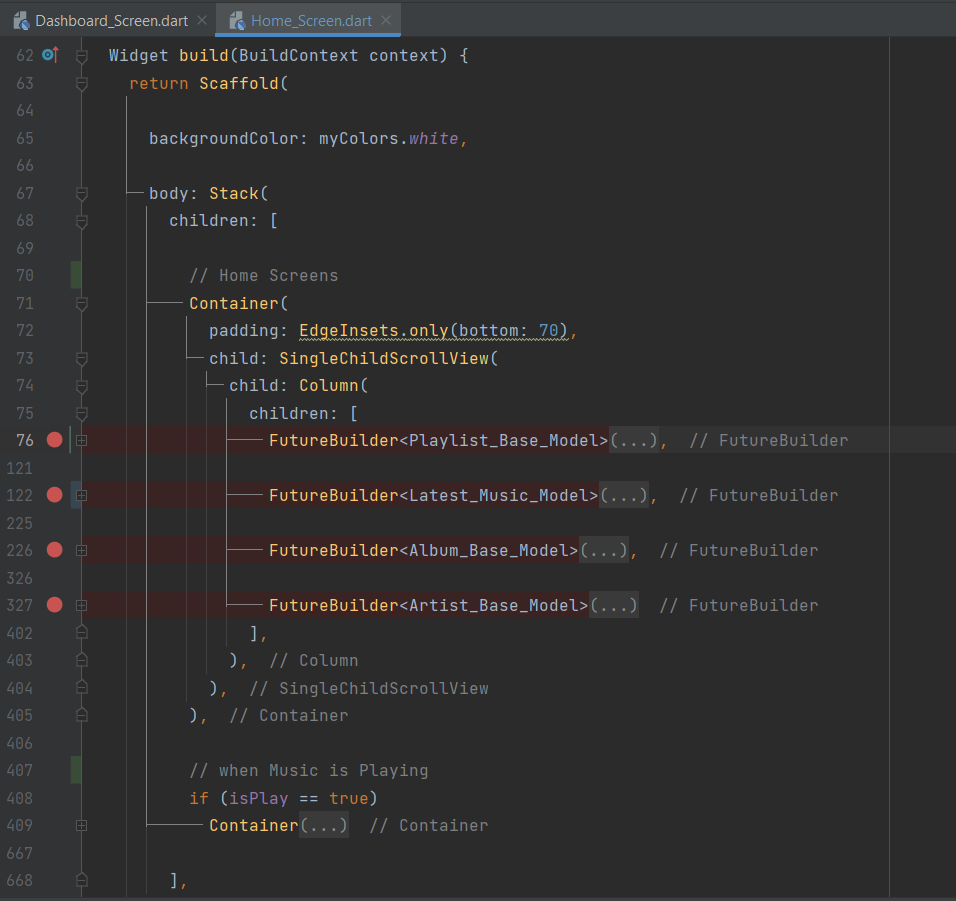
دو متغیر آخر \_duration و \_position برای نمایش ابتدا و انتهای مکان صوتی آهنگ یا seek استفاده میشوند.



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.

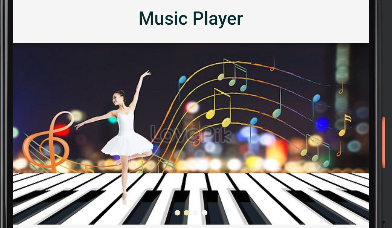
در ابتدای اجرای تابع اصلی این صفحه، برای نمایش لیستی از پلی لیست ها، آخرین آهنگ ها، آلبوم ها و خوانندگان، از 4 تا ویجت FutureBuilder طبق نوع آن لیستی مورد نظر استفاده میشود.



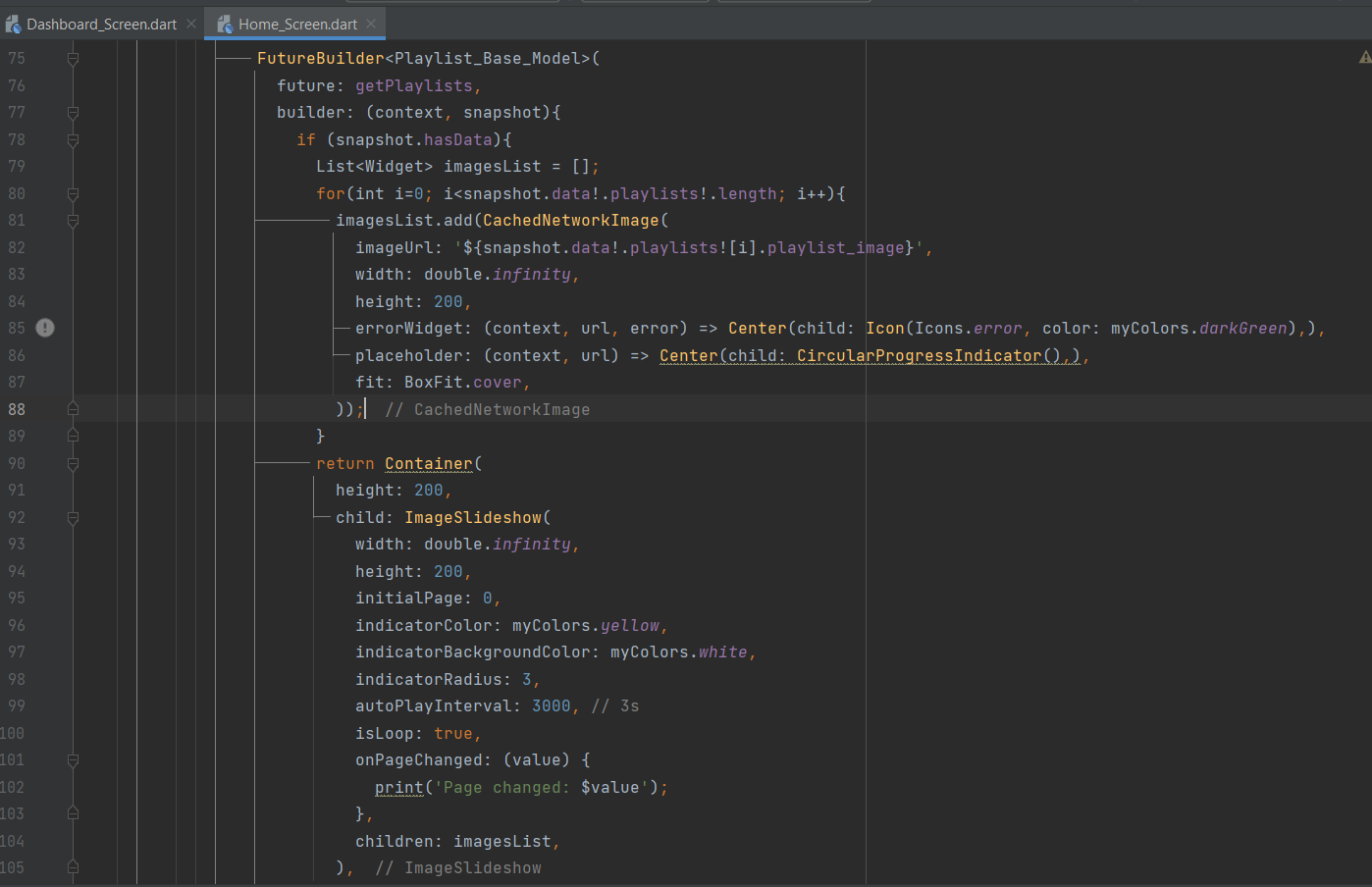
که توضیح هرکدام ازین موارد به شرح زیر است.

##### 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 :



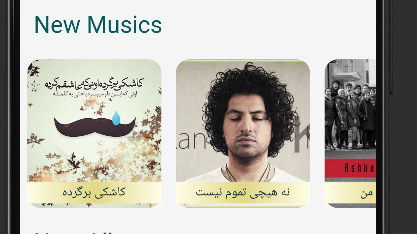
برای نشان دادن لیست هایی با داده هایی که از وب سرویس گرفته میشوند احتیاج به ویجت Future<> میباشد که کلاس درون آن باید از جنس داده هایی باشد که در ابتدای کد متیغیر آن ها را ساختیم ( Future<Playlist\_Base\_Model> getPlayllists).

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

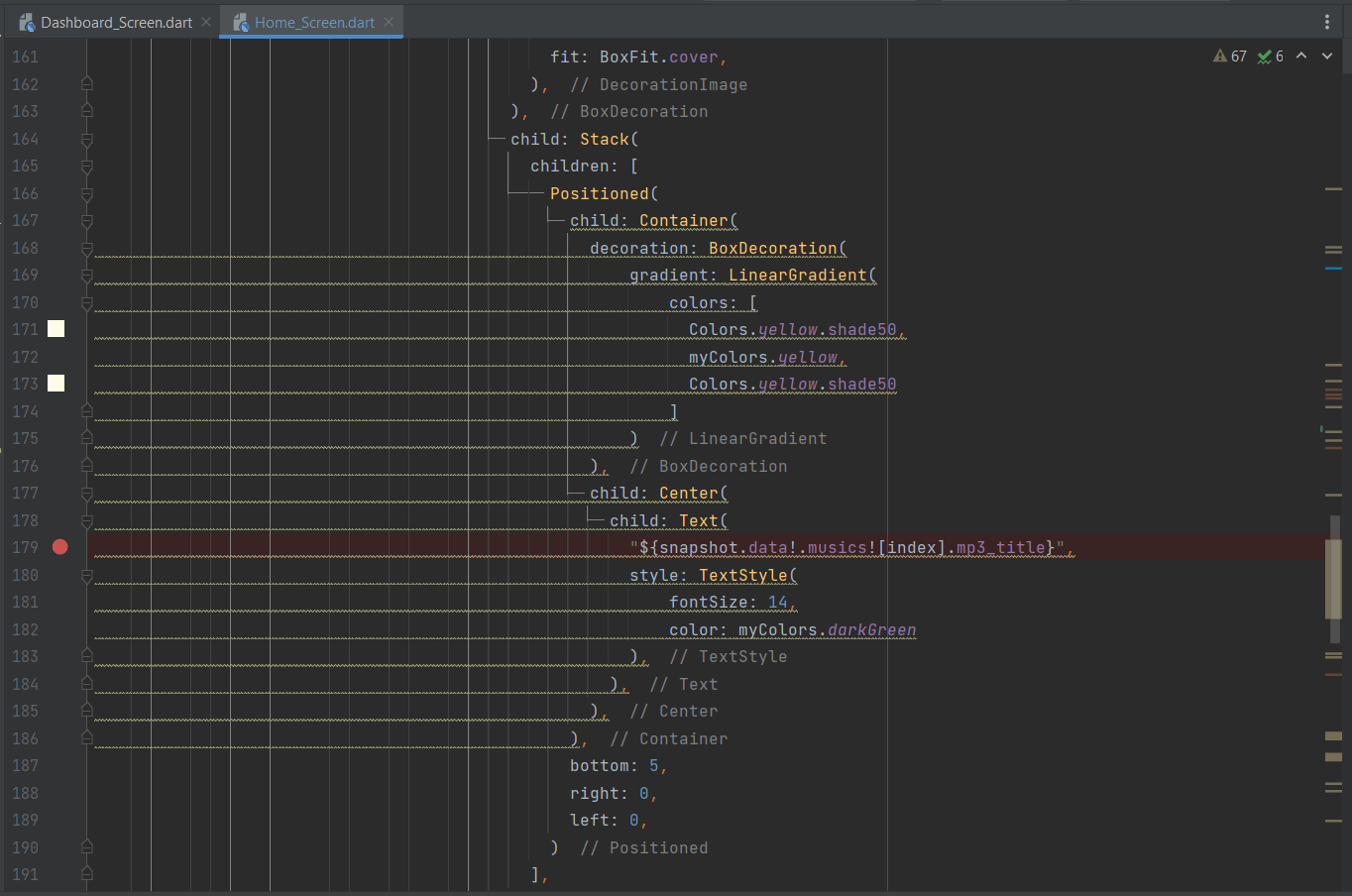
بعد از slide show پلی لیست ها، لیستی از آخرین آهنگ های آپلود شده در سایت میباشند:



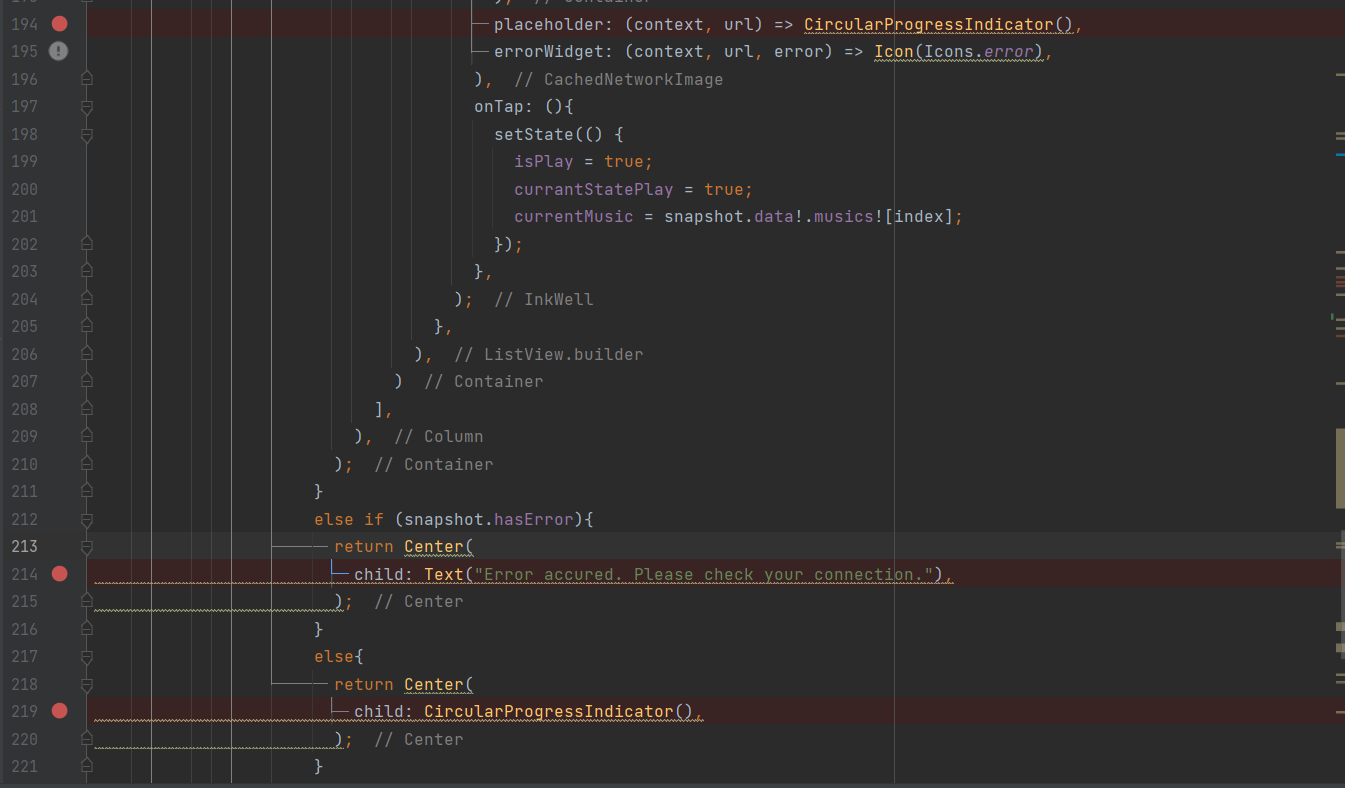
برای گرفتن لیست آهنگ هایی که در وب سرویس وجود دارند، لیستی از نوع Future و جنس Latest\_Music\_Model در ابتدای برنامه میسازیم. سپس در تابعی که به هنگام اجرای برنامه اجرا شده، به آن مقدار دهی میدهیم که از وب سرویس آهنگ های مورد نظر را دریافت کند. سپس در ListViewایی از نوع همان لیستی که تعریف کردیم، Future، 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:





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

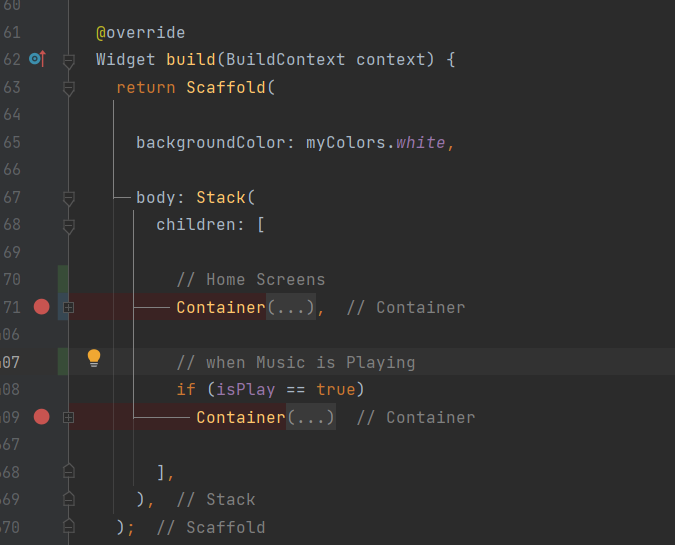


در ابتدای برنامه متغیرهایی تعریف شدند به نام های

* 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:



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

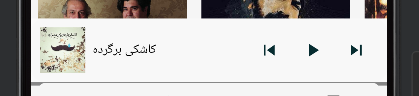
متغیر سوم، currentStatePlay، زمانی استفاده میشود که موزیک مورد نظر اجرا شود و صفحه اصلی تغییر کند.

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



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

که ابتدا آهنگ به صورت miniPlayer نمایش داده میشود:



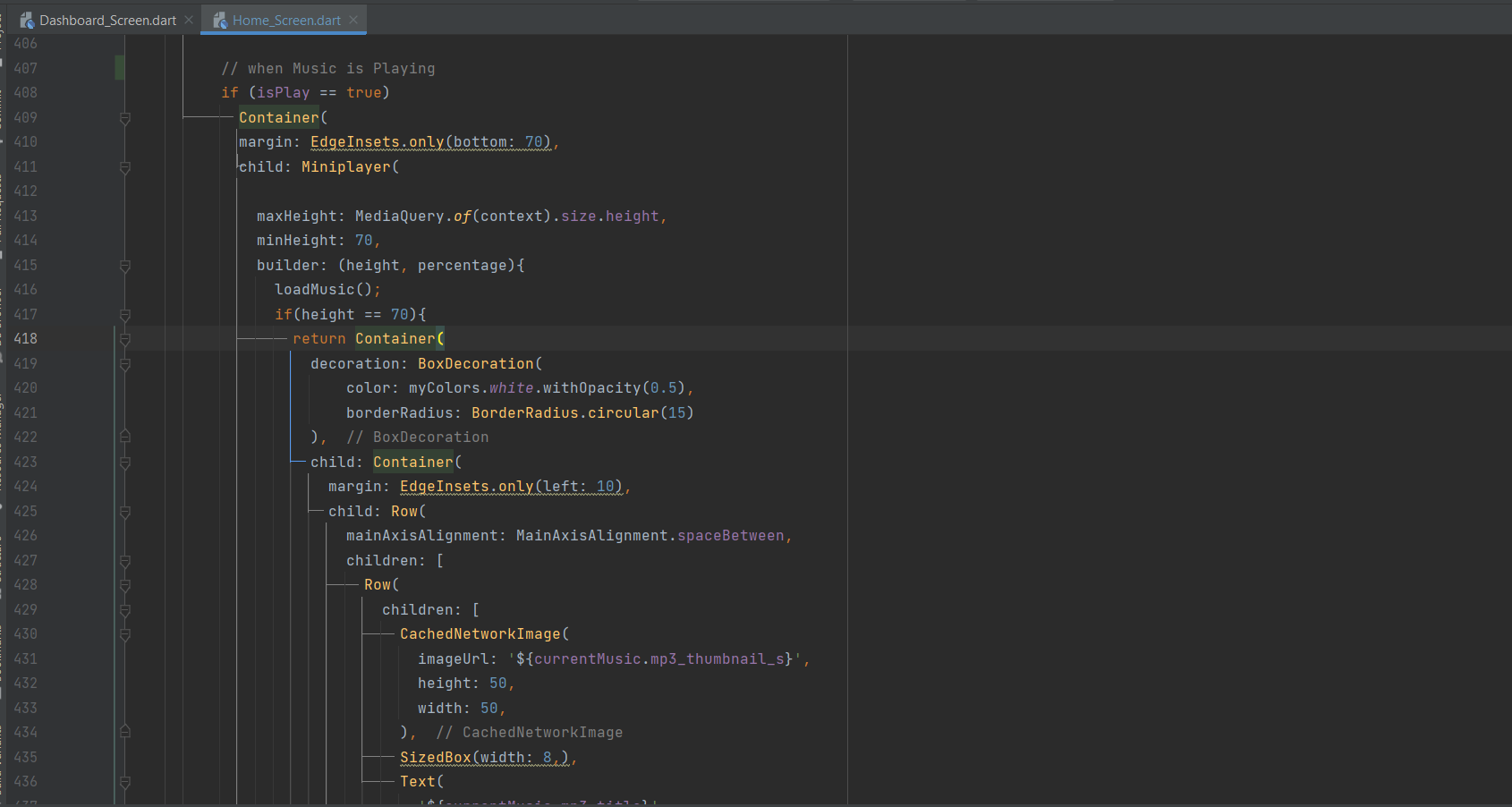
و با کلیک بر روی آهنگ تصویر به صورت بزرگ نمایش داده میشود:

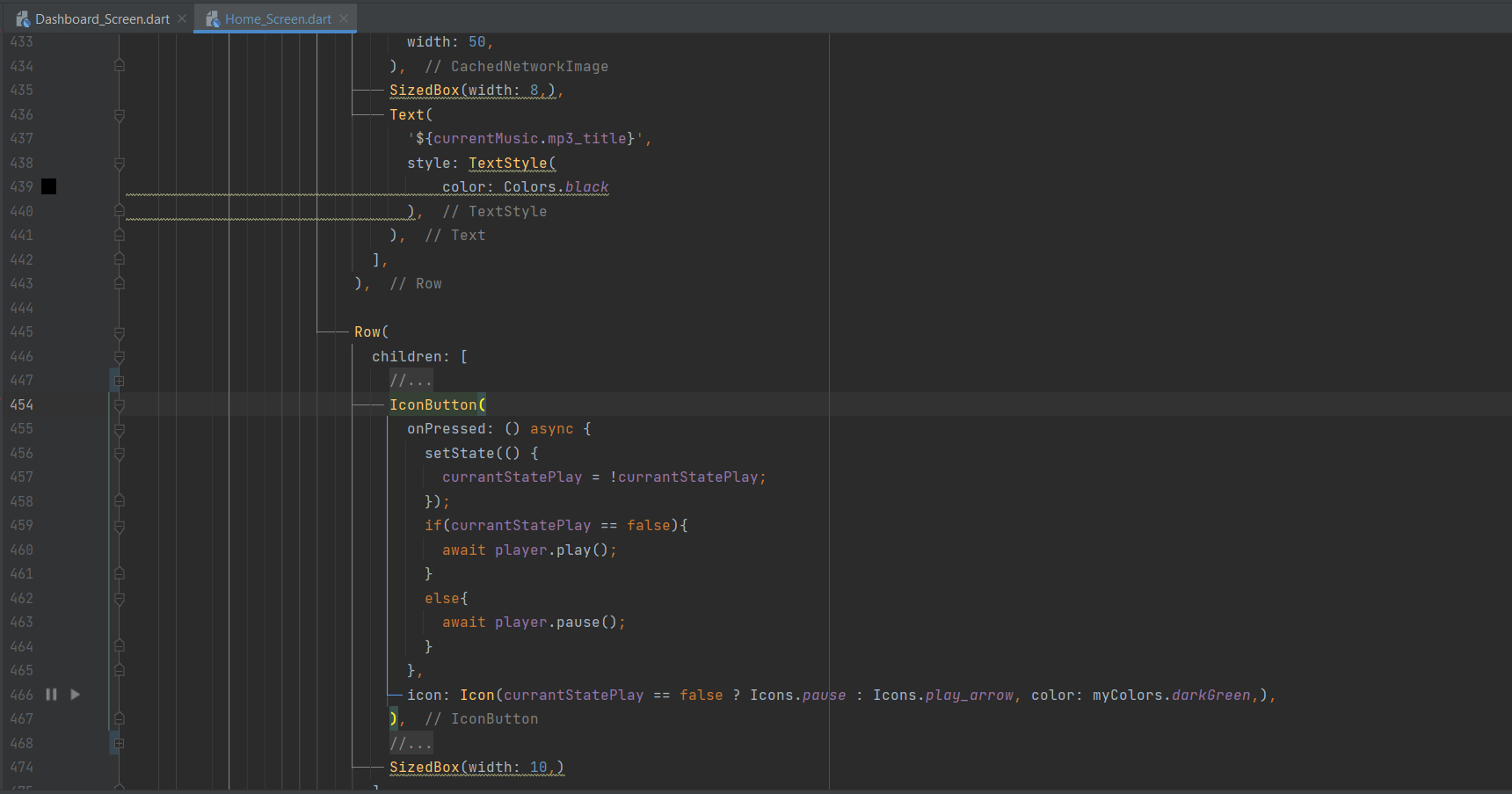


برای اجرای miniPlayer ابتدا باید برای ویجت مورد نظر طولی در نظر بگیریم که در اینجا کمترین طول برای پلیر 70 میباشد.

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

برای انجام چنین عملیاتی که به صورت زیر عمل میکند:

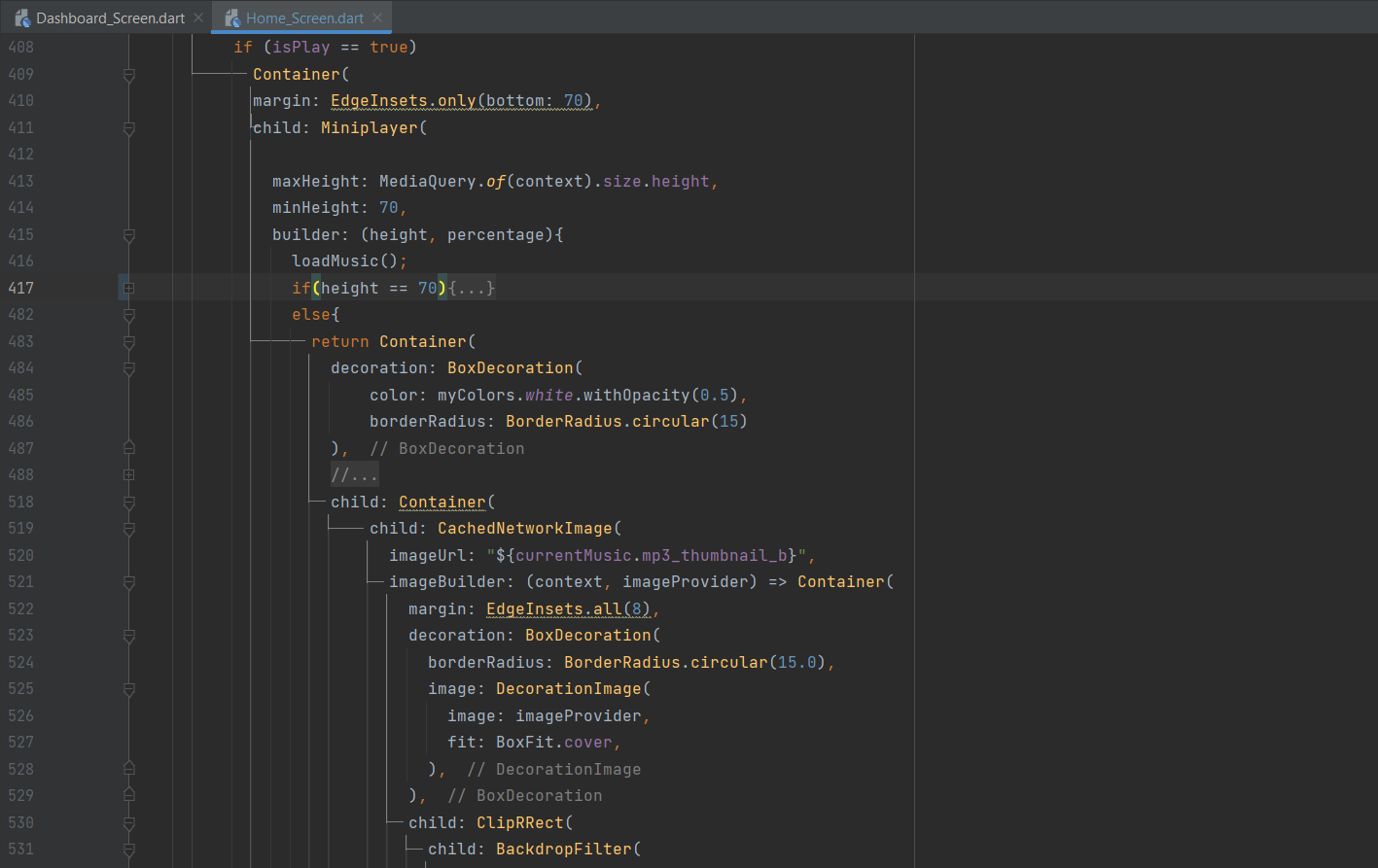


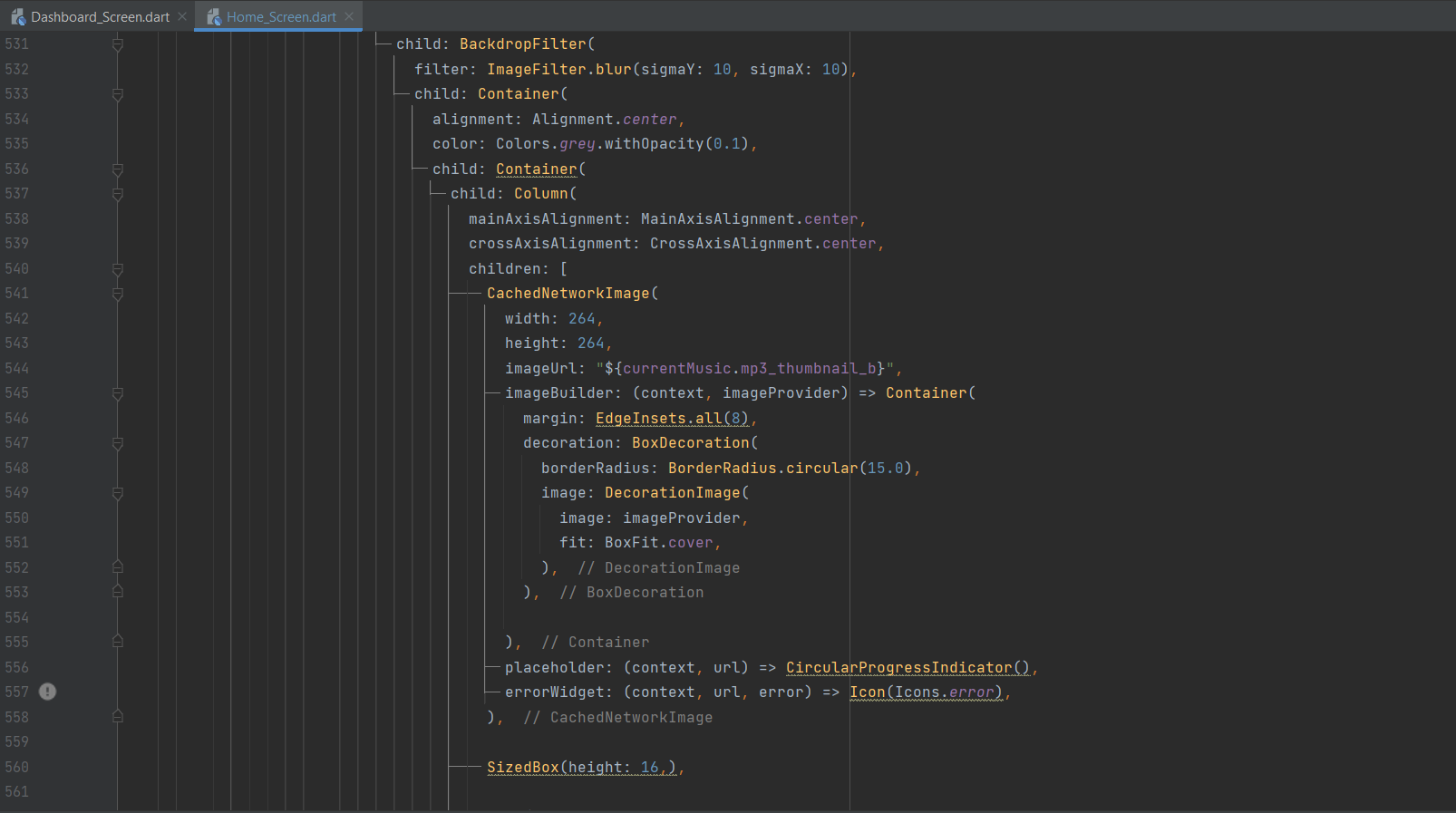


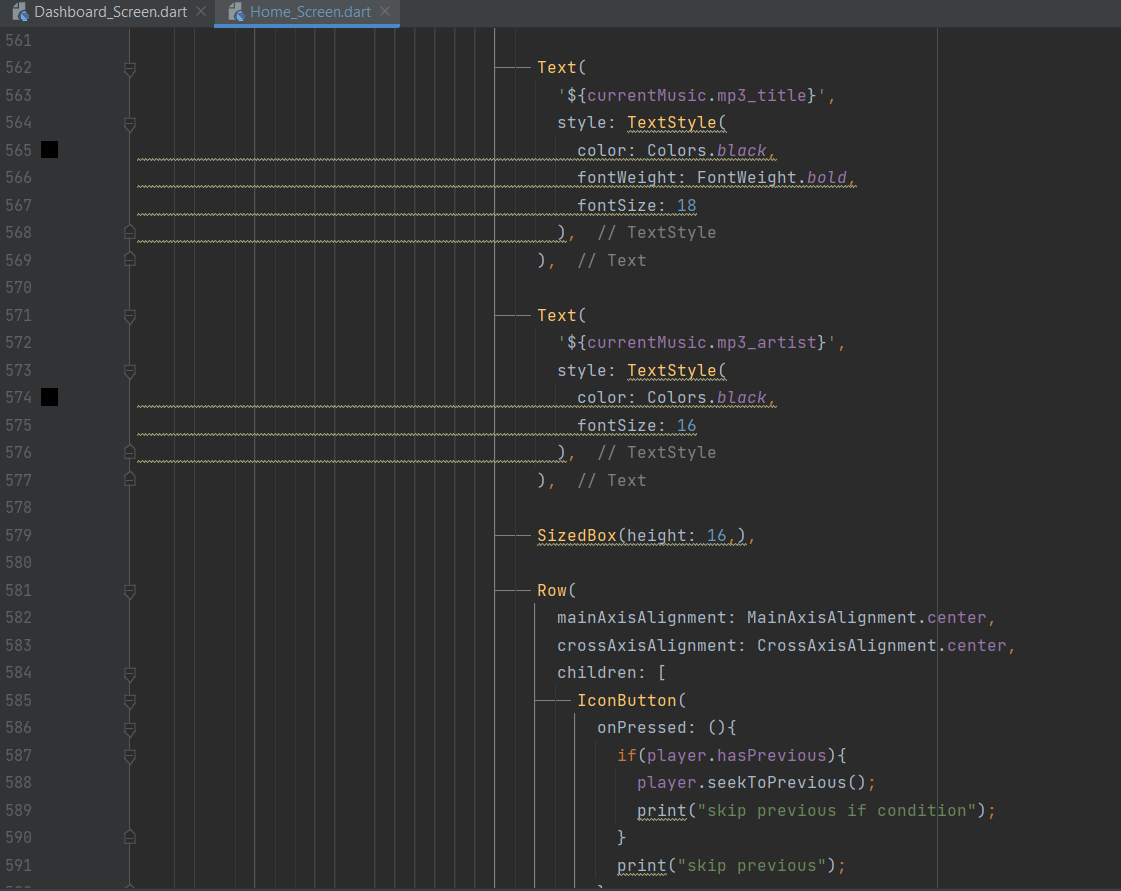
هنگامی که بر روی تصویر آهنگ کلیک زده میشود تا صفحه فول اسکرین شود، حالت پلی موزیک دوباره تکرار میشود.

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 :







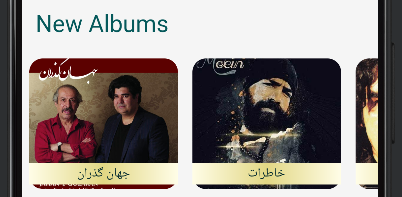


در این قسمت از برنامه برای رفتن به آهنگ بعد و یا برگشت به آهنگ قبل، اگر آهنگ مربوطه در انتهای لیست لیست نباشد و یا در ابتدای لیست نباشد، میتوان عملیات های seekNext و seekPrevious را انجام داد.

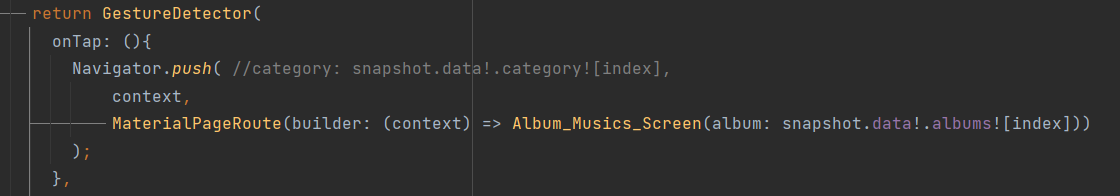
برای توقف و اجرای آهنگ، هنگامی که دکمه توقف زده میشود حالت اجرا، currentStatePlay، عوض شده و هم آهنگ متوقف میشود و هم علامت پلی موزیک به آیکون توقف عوض میشود.

##### 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:



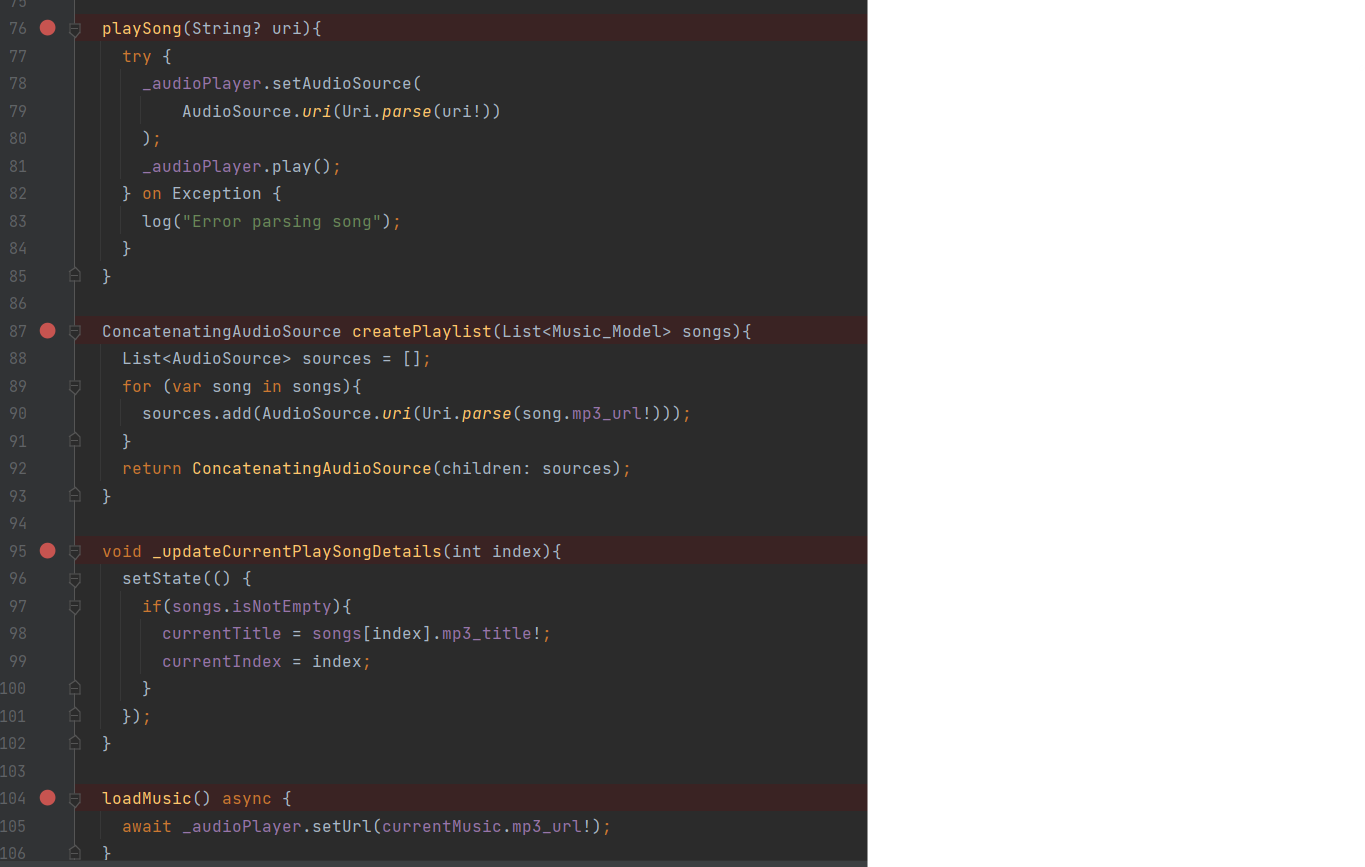
برای انتقال به صفحه بعدی، کلاس Album\_Music\_Screen نیاز به یک ورودی میباشد که این ورودی اطلاعات آلبوم مورد نظر میباشد که طبق آن داده های مورد نیاز را بگیریم.

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.



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 seekNext و seekPrevious 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.



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.

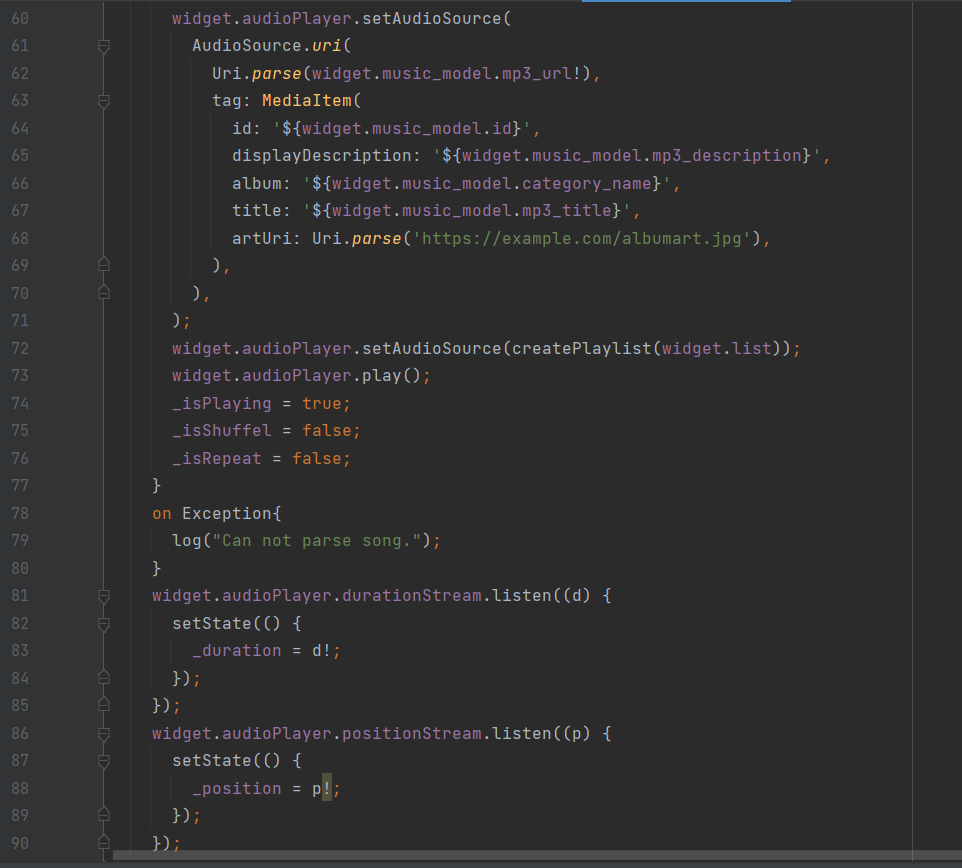
تصویر صفحه بعد به صورت زیر است:



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:



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 auidoPlayer 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(microseconds: 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:

Flexible(  
 child: InkWell(  
 onTap: (){  
 widget.audioPlayer.loopMode == LoopMode.one ?  
 widget.audioPlayer.setLoopMode(LoopMode.all) :  
 widget.audioPlayer.setLoopMode(LoopMode.one);  
 },  
 child: Container(  
 padding: EdgeInsets.all(10),  
 child: StreamBuilder<LoopMode>(  
 stream: widget.audioPlayer.loopModeStream,  
 builder: (context, snapchat){  
 final loopMode = snapchat.data;  
 if(LoopMode.one == loopMode){  
 return Icon(Icons.*repeat\_one*, color: myColors.*yellow*,);  
 }  
 return Icon(Icons.*repeat*, color: myColors.*darkGreen*,);  
 },  
 ),  
 ),  
 ),  
),

در حالتی که چیزی فعال نباشد، در هنگام اتمام آهنگ به آهنگ بعدی میرود و اگر آخر لیست باشد از اول لیست آهنگ ها دوباره اجرا میشوند.

برای استفاده از حالت پخش درهم یا play shuffle به صورت زیر عمل میکنیم:

ابتدا متغیری از جنس bool تعریف میکنیم:

bool \_isShuffel = false;

و سپس برای استفاده نهایی به صورت زیر پیش میرویم:

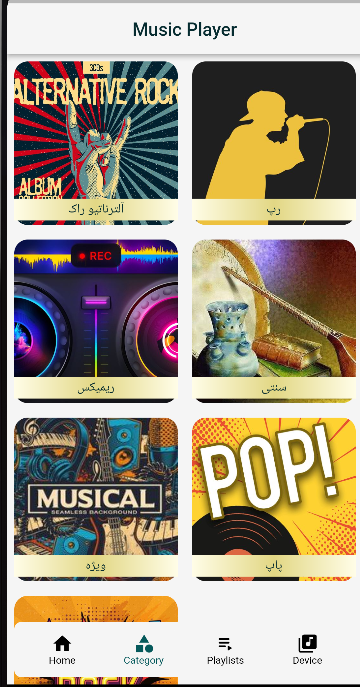
Flexible(  
 child: InkWell(  
 onTap: (){  
 setState(() {  
 if(\_isShuffel){  
 widget.audioPlayer.setShuffleModeEnabled(true);  
 }  
 else{  
 widget.audioPlayer.setShuffleModeEnabled(false);  
 }  
 \_isShuffel = !\_isShuffel;  
 });  
 },  
 child: Container(  
 padding: EdgeInsets.all(10),  
 child: Icon(  
 Icons.*shuffle*,  
 color: \_isShuffel? myColors.*yellow* : myColors.*darkGreen* ),  
 ),  
 ),  
),

##### Artist List

نمایش لیست خوانندگان همانند نمایش لیست آلبوم ها میباشد با این تفاوت که دیتا داده شده به ویجت Future باید از جنس کلاس Artist\_Base\_Model باشد و لیستی که با آن میدهیم نیز دیتاهای مربوط به خوانندگان را برگرداند.

#### صفحه Category

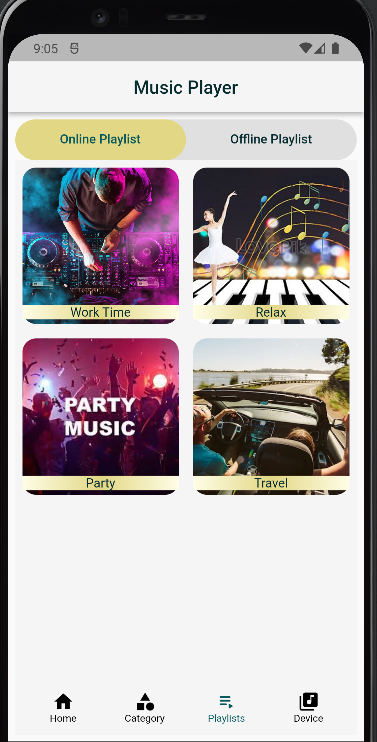
در این صفحه همانند صفحه قبل در قسمت لیست خواننده و آلبوم میباشد با این تفاوت که چیدمان ویجت های این صفحه با آن ها تفاوت دارد و به صورت زیر است:



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.



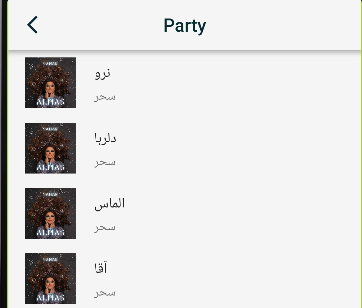


در ابتدا کنترلری تعریف شده که باید مقدار طول TabBarایی که میخاهیم استفاده کنیم را مشخص نماییم. سپس نام و یا آیکون های مورد نظر را در قسمت tabs در ویجت TabBar وارد میکنیم. در آخر در ویجت TabBarView به تعداد tabهایی که تعریف کردیم، صفحات مورد نظر را برای آن مشخص میکنیم.

##### Online Playlist

نمایش پلی لیست آنلاین، همانند نمایش لیست های دسته بندی مبیاشد 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:

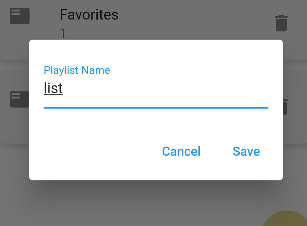


نمایش لیست پلی لیست ها همانند قبل میباشد با است تفاوت که برای گرفتن لیست آن ها از دستگاه از کلاس OnAudioQuery استفاده میکنیم؛ و هنگام استفاده از این کلاس به صورت زیر پیش میرویم:

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 میباشند.

با زدن بر روی آیکون اضافه کردن، AlertDialogایی به وجود میاورد که اسم لیستی که میخواهیم را مینویسیم که به صورت زیر است:



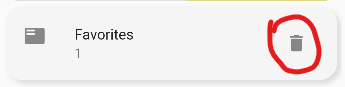
نمونه کد استفاده شده برای انجام این کار به صورت زیر است:

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 ویجت Future<List<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\_backgeround, 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 اطلاهات 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 تقریبا همانند قسمت اول، نمایش صفحه Home و اجرای آخرین آهنگ ها، میباشد.

برای لیست آهنگ های دستگاه از ویجت Futureاستفاده میکنیم ولی نوع کلاسی که به آن میدهیم 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  
),

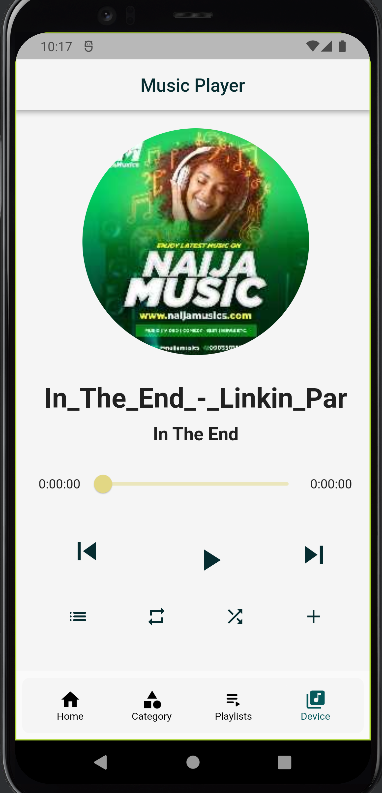
در اینجا از طریق متغیری که از کلاس OnAudioQuery ساخته شده است، لیست آهنگ ها را با تابع .querySongs میگیریم.

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 میدهیم 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 آیکون 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 AndroidMainfest.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_O0eNdleGLZtd9lIXMioypkqCiJ>
* <https://www.youtube.com/watch?v=fWLlbxCNFKo>
* <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=QKqLvy7uqCE>
* <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**