DIMA Project report: MBox

Simone Abelli, Stefano Azzone

May 13, 2022

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to provide more technical and detailed information about the mobile application developed. The Design Document is a guide for the programmer that will manage the future development of the codebase for the application in all its functions. The document will explain and motivate all the architectural choices by providing a description of the components and their interaction. We will also enforce the quality of the product through a set of design characteristics. Finally we describe the implementation, integration and test planning. The topics touched by this document are:

- high level architecture
- main components, screens and widgets
- runtime behavior
- design patterns
- more details on user interface
- requirements and their mapping on the architecture
- implementation and test planning

1.2 Scope

MBox is an application for mobile devices that allows users to access all the music present on their devices without having to worry about manually managing the metadata. The target user has basic knowledge of the functions of the mobile device (e.g. is able to interact with the filesystem). Since the application is focused on users who want to listen to music, which is usually done with a portable device, we will focus smartphones maily (the application also works on tablets). For the development of this applications we have decided to use flutter since it's a framework thought for multiplatform deployment.

1.3 Glossary

Queue: a queue is a data structure containing a list of tracks which are to be played according to a FIFO policy.

Metadata: metadata are information of a track regarding the track itself (e.g. track title, artist, album cover ...)

2 Description and requirements

2.1 Product description

MBox has all the functionalities of a music player:

- Detect music present in the Music folder
- Allow management of playlists
- Organize music by artist, album and playlist
- Allow to arrange tracks in a playback queue
- Play tracks (in random order if desired)
- Visualize metadata related to tracks

In addition to this, MBox allows a more advanced and automated metadata management, and the access to other music information:

- Automatically set missing metadata (Title, Artist, Album, Cover, Lyrics, Track number, Artist image, ...)
- Manually edit metadata
- Visualize information about artists like a brief description, albums and other songs
- Search on the internet for other songs and listen to them

2.2 Assumptions

- When downloading metadata or looking for other music, internet connection is available
- The user is able to move their music to the Music folder
- The tracks are present on Spotify to download metadata
- When the user looks for a track not present on their device, it must be present on YouTube
- The tracks are in mp3 format (otherwise metadata management is harder)
- The access to the filesystem is granted
- The device on which the application is used has some means to play music

2.3 Functional requirements

1. The application can access the filesystem and fetch songs from the music folder
2. The application can play the selected tracks
3. The application allows the user to add or remove tracks from the playback queue
4. The application allows the user to pause and resume playback
5. The application allows the user to skip tracks in the queue
6. The application should save the queue state to resume playback if the application is closed
7. The application allows the user to add or remove playlists
8. The application allows the user to add or remove tracks from playlists
9. The application organizes the tracks by artist and album
10. The application allows the user to view lyrics of currently playing track
11. The application allows the user to edit metadata of the tracks
12. The application automatically adds missing metadata using an external service (in our case Spotify); already present metadata is kept
13. The application can show information about the artists of the tracks present on the device (such as all their albums and tracks)
14. The application allows the user to search for tracks not present on their device, and play them through an external service (in our case YouTube)
15. It should be possible to use the application without an internet connection, obviously with limited functionalities (no external song search, metadata editing . . .)

2.4 Non functional requirements

1. The application should feel snappy and responsive
2. The application layout should be intuitive to use
3. The application should be reliable

3 Architectural Design

3.1 Architectural Style

The application is logically subdivided in a frontend and a backend. The frontend presents the information to the user and allows them to interact with the backend. The backend contains and manages all the data and interfaces with the external services.

3.1.1 Frontend

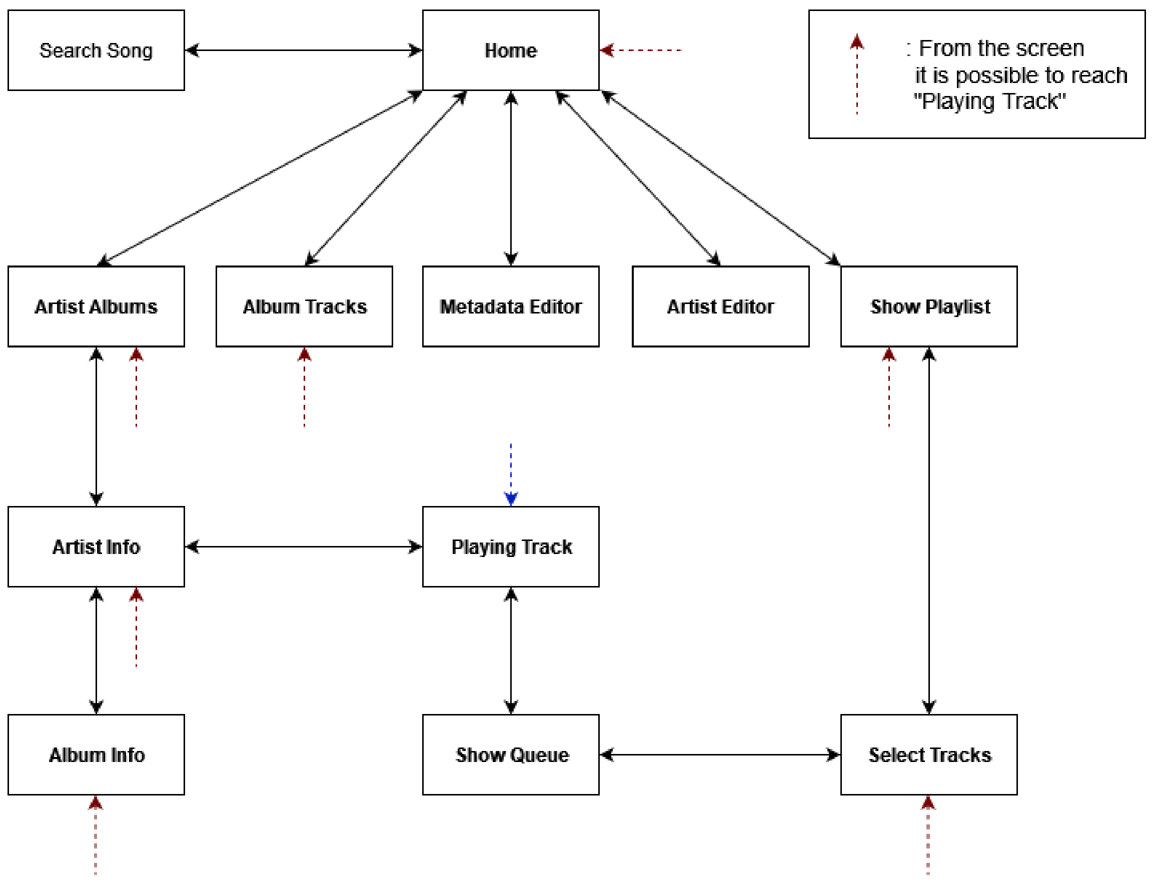

Figure 1: Frontend system architecture

The frontend is composed by the screens of the application. Once opened MBox shows the **Home** screen. The Home screen is composed by 4 tabs:

- *Tracks*: a list of the tracks in alphabetical order; by selecting one we switch to the **PlayingTrack** screen, and start playing that track.

- *Albums*: a list of the albums in alphabetical order; by selecting one we switch to the **AlbumTracks** screen, where we can select a track to play.
- *Artists*: a list of the artists in alphabetical order; by selecting one we switch to the **AlbumArtists** screen, where we can select an album to show.
- *Playlists*: a list of the playlists in alphabetical order; by selecting one we switch to the **ShowPlaylist** screen, from which we can start the playback; in the Playlists tab it is also possible to add a new playlist or remove an existing one.

By keeping a track pressed it is possible to open a drop down menu with a few entries:

- *Edit metadata*: switch to the **MetadataEditor** screen to modify the metadata of the track
- *Add to queue*
- *Add to playlist*

By keeping an artist pressed it is possible to switch to the **ArtistEditor** screen in order to change its image.

By touching the magnifying glass in the top right corner of the application, it is possible to reach the **SearchSong** screen that allows the user to look for songs that are not present on their device, and play them through an external service (in our case YouTube).

From the **PlayingTrack** screen it is possible to:

- *pause* and *resume* playback
- *skip* forward and backward in the queue
- *change playback position* of the currently playing track
- view the *album cover* and the *lyrics* of the currently playing song
- check the *artist information* by switching to the **ArtistInfo** screen: this screen shows a brief description of the artist (from Wikipedia) and a list of their albums; by selecting one of these, the user is brought to the **AlbumInfo** screen, from which they can view all the tracks and play them using YouTube
- *show the queue* by switching to the **ShowQueue** screen.

The **ShowQueue** screen shows the current track queue; the tracks are ordered according to their position in the queue, where the currently playing track has index 0, the past tracks have negative index, and the future tracks have positive index. From here it is possible to add new tracks to the queue by means of the **SelectTracks** screen. Almost every screen has a **PlayBar** that allows to pause and resume playback, and shows currently playing track info and artwork. By pressing it, the user is brought back to the **PlayingTrack** screen.

3.1.2 Backend

The backend is composed by various modules:

- **Database**
- **Track queue**
- **Metadata loader**
- **Player**
- **Worker**

The database is the core of the application and all the other backend components depend on it. Indeed the metadata loader is used by the database to collect data from the internet, the Worker is an isolate that allows the database to load in a parallel fashion data from internal storage, the track queue is initialized by the database, and the player uses the information contained in the database to play music.

Database The database is the main component of the application. This component is not an actual database for the following reason: in our application most of the time we want to fetch the data of a particular category (tracks, albums, artists, playlists). The use of a real database for this reason would be superfluous: it suffices for our purposes to use a hand crafted one.
Moreover our database has other functionalities that are tightly coupled with the domain's data and it would thus be difficult to integrate them in a traditional one, and probably inefficient.

When the appication boots, the database is initialized.
It uses the worker isolate and the metadata loader to discover which tracks are present on the device:

1. Loads the saved database file (if it exists) with all its contents: tracks, artists, albums, playlist, current track queue.
2. Checks the Music folder for songs not present in the database ile.
3. For those songs the missing metadata, available on the internet, are downloaded and written in the music file (as id3 tag).
4. The new tracks are inserted in the database (along with its metadata).
5. Update the database file.

Once the database is initialized all the other components can access the music data from it.

Another functionality of the database is to allow the user to modify the metadata of the tracks. Finally this component also grants the user the ability to refresh the database itself: all the data will be reloaded from scratch.

Track queue The track queue is the list of all the songs to be played. It is initialized and saved by the database to grant persistency across application restart. An index is assigned to each track: the track of index 0 is the one currently playing, one with negative index has already been played, and one with positive index will be played.
Through the frontend components the user can modify the track queue, by adding, removing tracks or changing the one currently playing.

Metadata loader The metadata loader is the component that handles the interaction with the internet. It is capable of:

1. Checking the internet connection.
2. Retrieving and extracting track information from the Spotify api.
3. Retrieving and extracting lyrics from the Genius api.
4. Retrieving information about artists from Wikipedia.
5. Searching for songs on YouTube.

All these functions are used by the application to perform all its tasks.

Player The player is the component that allows music playback: it allows to pause or resume a track, update the track position, go forward or backward in the track queue. It automatically switches to the next song in queue when the current one is over.
It also informs the components interested when the currently playing song changes.

Worker The worker is a component that includes an isolate and an interface to communicate with it. The isolate is tasked with the management of the access to the internal storage.
It reads and writes database information. It is used to separate the access to file from the main isolate, in order to increase performance.

3.2 Architectural Patterns

3.2.1 Facade pattern

In the system architecture we can notice that the user accesses the application services through the frontend components; the frontend is the facade through which the user accesses the internal logic of the system (the backend). In this way it hides the internal complexity of the system, providing a simple and unique interface that the user can access.

Using this pattern the user does not need to the internal structure of the software and it can use the simple screens of the application to interact with the backend. Another major benefit of this pattern is that any change in the backend of the architecture will stay undetectable from the user, provided that the facade stays the same.

For instance, if we decided to substitute the existing database with a SQL based one, the user wouldn't notice a thing.

3.2.2 Master-Worker pattern

When the application boots an isolate is created. This isolate is a worker and accepts orders from the main thread (the Master). The Worker component in fact keeps waiting for a message from the Master, and when it receives one, it is processed, and if a reply is needed, it is sent to the Master. The Worker then resumes waiting for work.

3.2.3 Singleton pattern

Some of the components are to be intended as singletons: only one instance of that particular component must exist at a time, and different calls to that component must reach that unique instance. This is achieved through the singleton pattern. In our application all the components of the backend (database, track queue, metadata loader, worker, player) are singletons, since it would be unreasonable to have more than one instance of each.

3.3 Component view

Here we display the main architecture of our application, and describe in detail all of its subcomponents.

3.3.1 Home

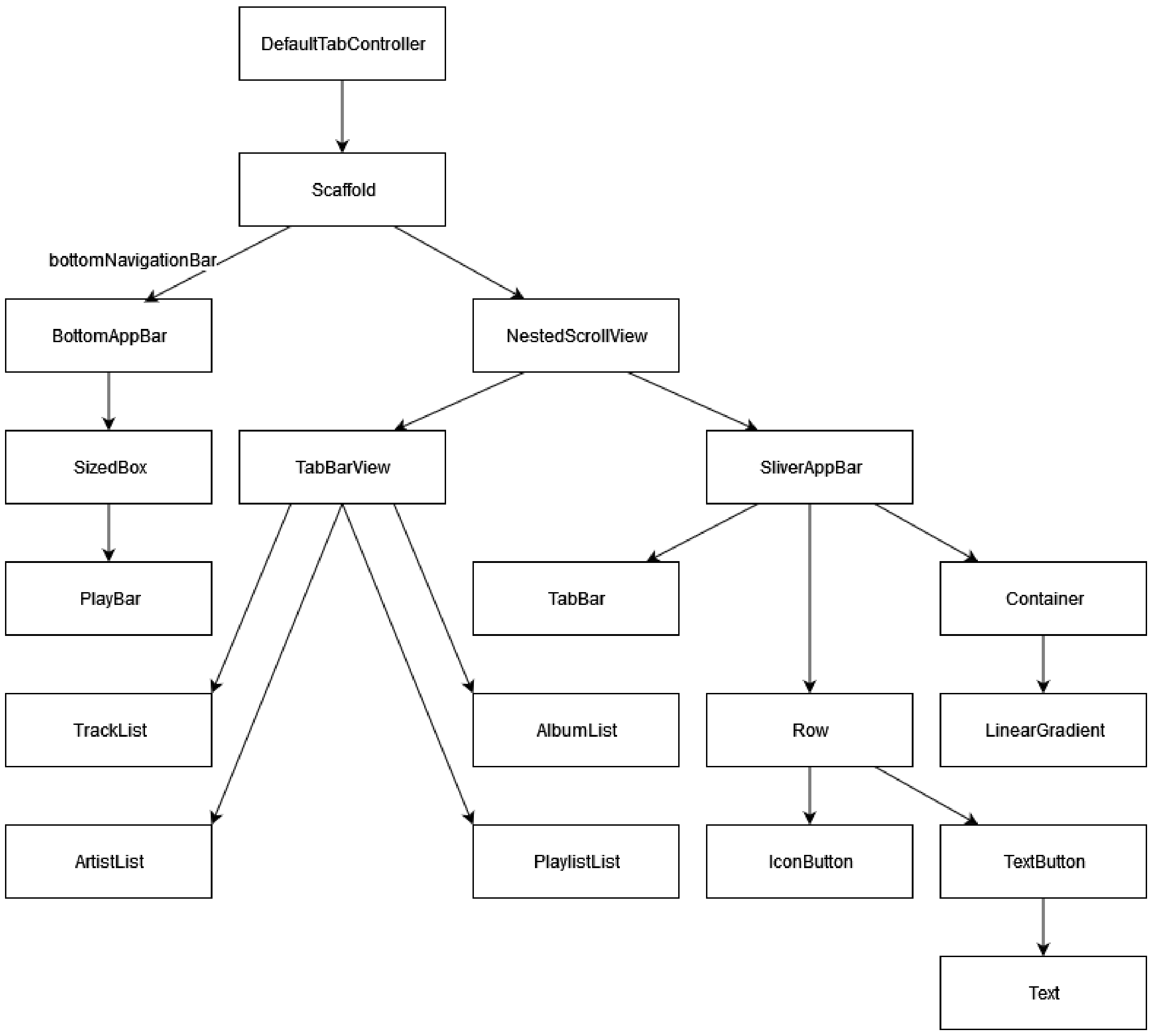

Figure 2: Home

The home component is the component shown when the application is first opened. It is composed of the PlayBar, that allows to control playback and open the track queue to modify it, and the NestedScrollView, that contains the TabBarView and the SliverAppBar.

The TabBarView is the widget that shows the available sections of the music library (TrackList, ArtistList, AlbumList and PlaylistList).

The SliverAppBar contains the TabBar that actually allows to switch between the available sections. It also hides itself when the TabBarView is scrolled down for a sufficient amount.

3.3.2 AlbumList

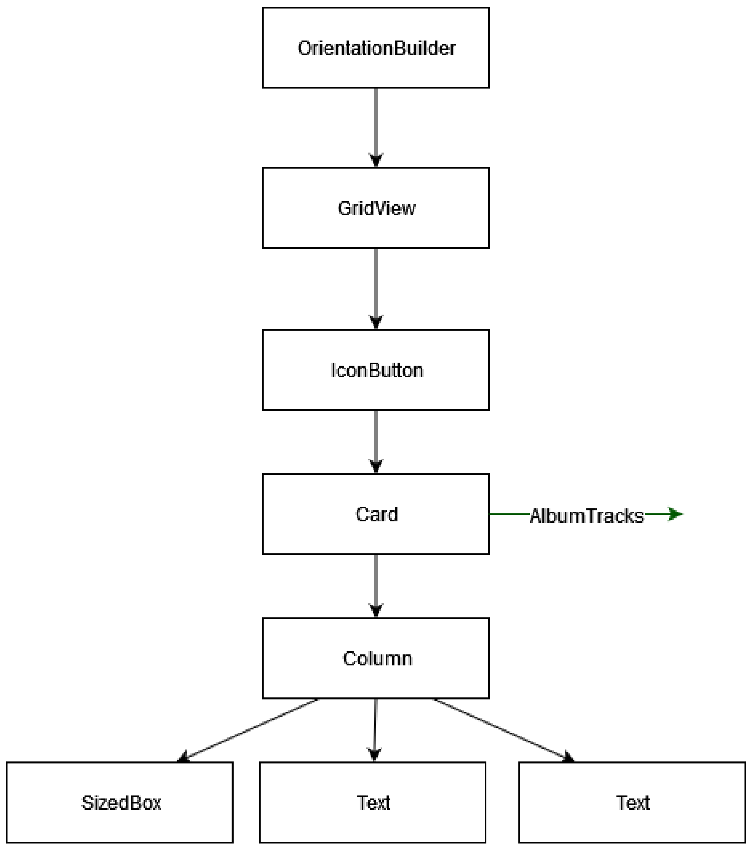

Figure 3: AlbumList

This component displays in a grid (GridView) the list of albums (Card) present on the device. The card contains the cover of the Album, its name and artist.

If a card is tapped (IconButton) the application navigates to the tracklist of the selected album.

The OrientationBuilder allows to choose an appropriate layout for the device orientation.

3.3.3 ArtistList

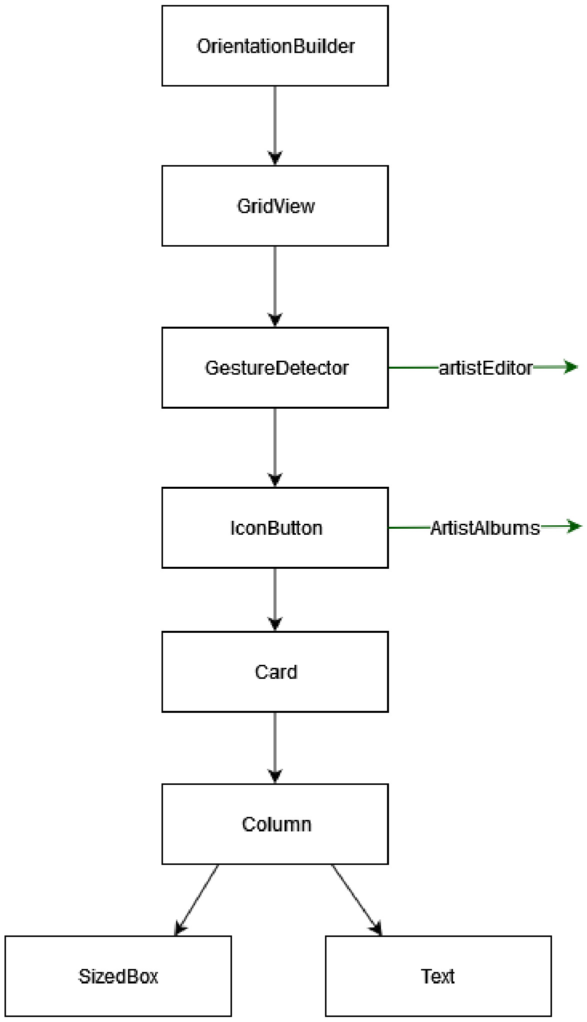

Figure 4: ArtistList

This component displays a GridView of the Artist Cards, that contain the artist name and image. The GestureDetector allows to edit the artist image, while the IconButton allows to access the ArtistAlbum screen.

The OrientationBuilder allows to choose an appropriate layout for the device orientation.

3.3.4 PlaylistList

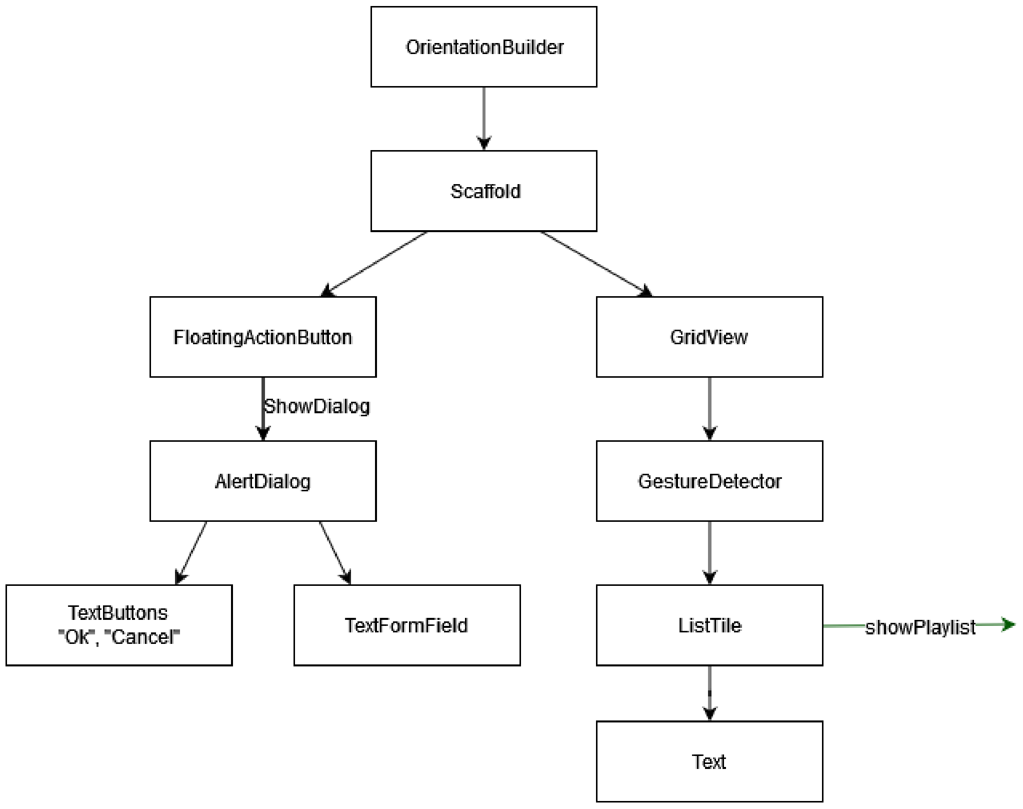

Figure 5: PlaylistList

This component displays a GridView of ListTiles. When tapped, the ListTile navigates to the ShowPlaylist screen. The GestureDetector detects long presses and shows accordingly a popup menu to delete the selected playlist if desired.

There is a FloatingActionButton in the bottom right corner of the screen that allows creating new playlists.

The OrientationBuilder allows to choose an appropriate layout for the device orientation.

3.3.5 PlayBar

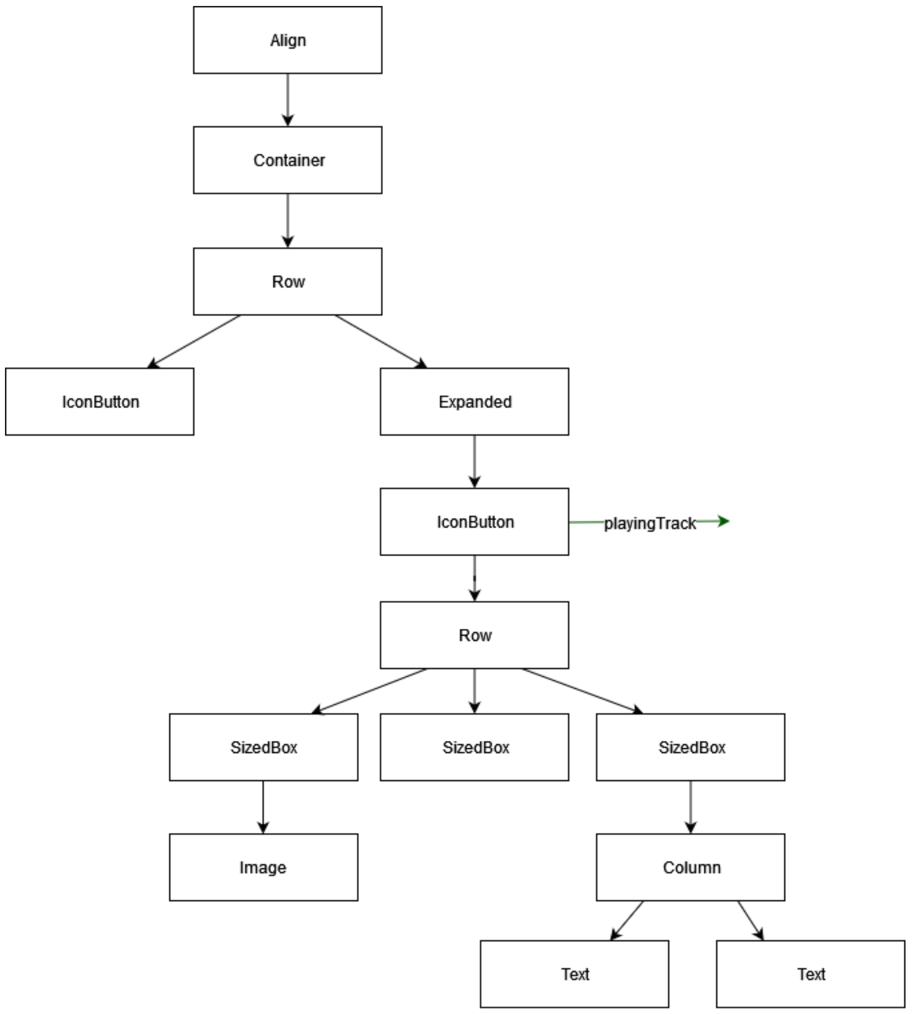

Figure 6: PlayBar

The PlayBar contains on the right a button that allow to play or pause music, depending on current track queue state. On the left there is the description of the currently playing track, with album cover, track name and artist.

When tapping the track description the application navigates to the PlayingTrack screen.

3.3.6 TrackList

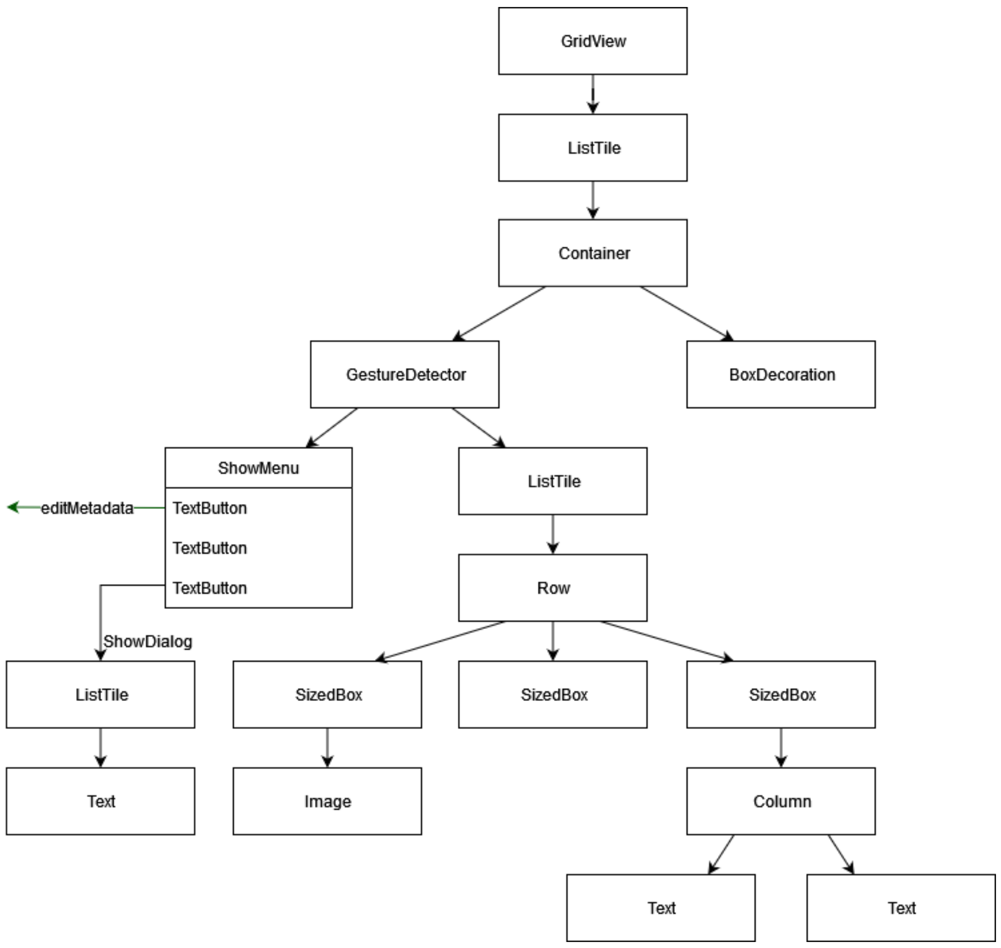

Figure 7: TrackList

The TrackList is the component that organizes tracks received as argument in a grid. The grid is composed of GestureDetectors, longtapping which the popup menu can be opened.

The menu allows to execute operations on the selected track: view its metadata, add it to queue or add it to playlist. Instead a short press on the ListTile brings the user to the PlayingTrack screen and immediately starts playing the track.

The ListTile components have on the left the album cover, and on the left the name and artist of the track.

3.3.7 ProgressBar

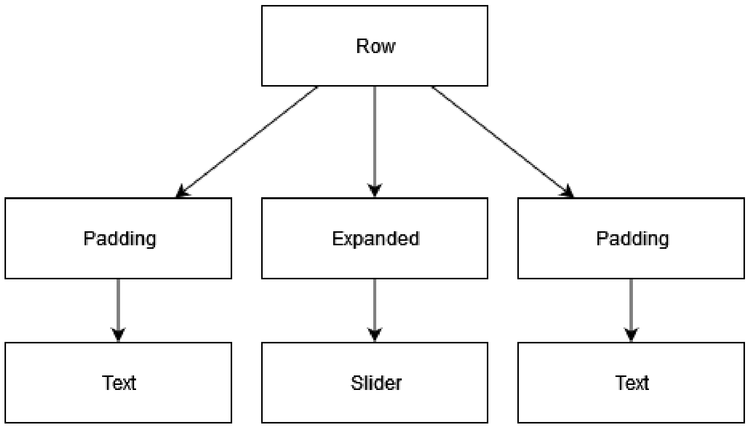

Figure 8: ProgressBar

The ProgressBar is the bar that allows monitoring the progress of the currently playing track.

It also allows to move the track position forward or backward, using the Slider. Text at the left side shows the track position, and at the right shows the track length.

3.3.8 CoverButton

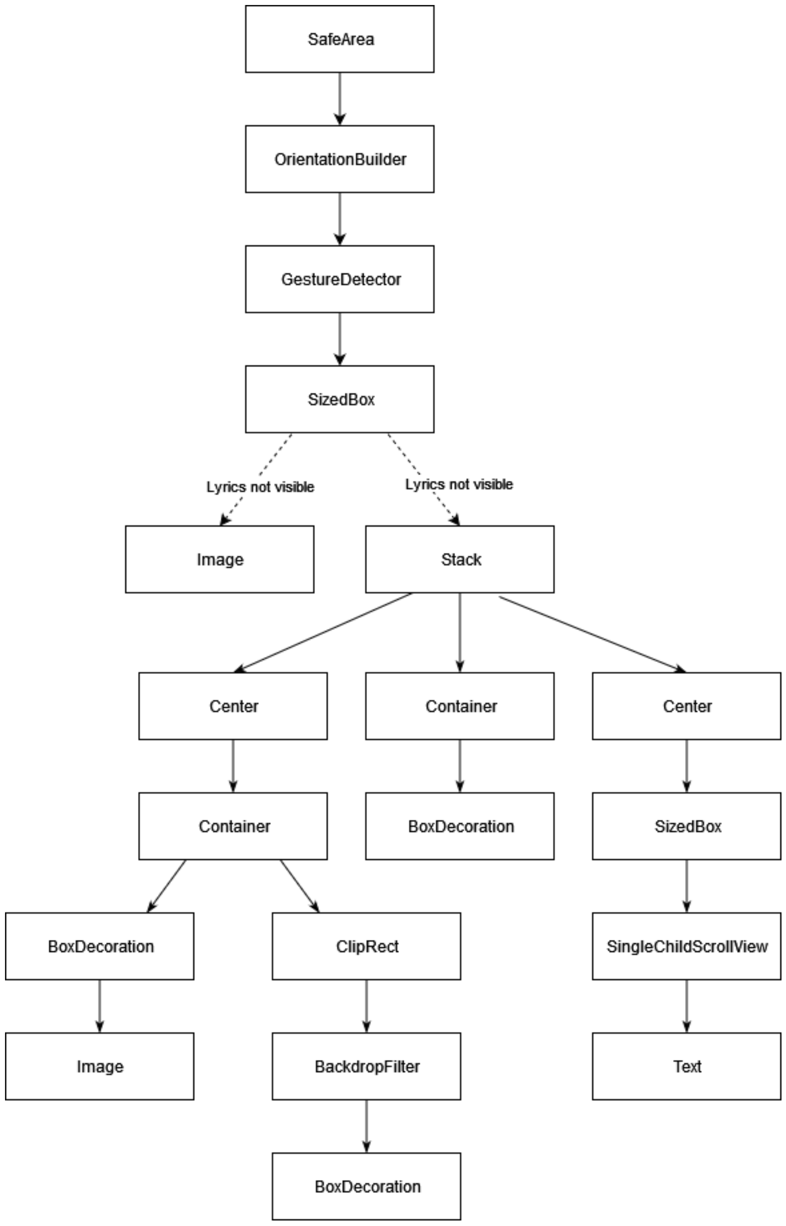

Figure 9: CoverButton

The CoverButton is the widget that displays the cover art of an album inside the PlayingTrack screen. If pressed (GestureDetector), it shows the lyrics (scrollable

Text) of the currently playing track, if available.

The background of the lyrics scroll view is transparent, so that the artwork can be glimpsed behind them.

The CoverButton is subscribed to the Player, so that whenever a change in the playing track occurs, that change is shown.

3.3.9 ArtistEditor

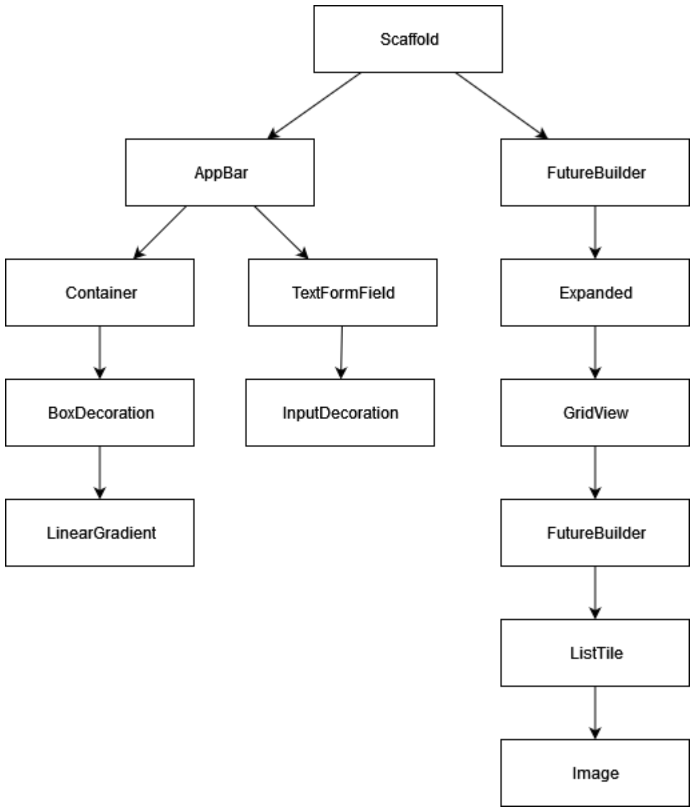

Figure 10: ArtistEditor