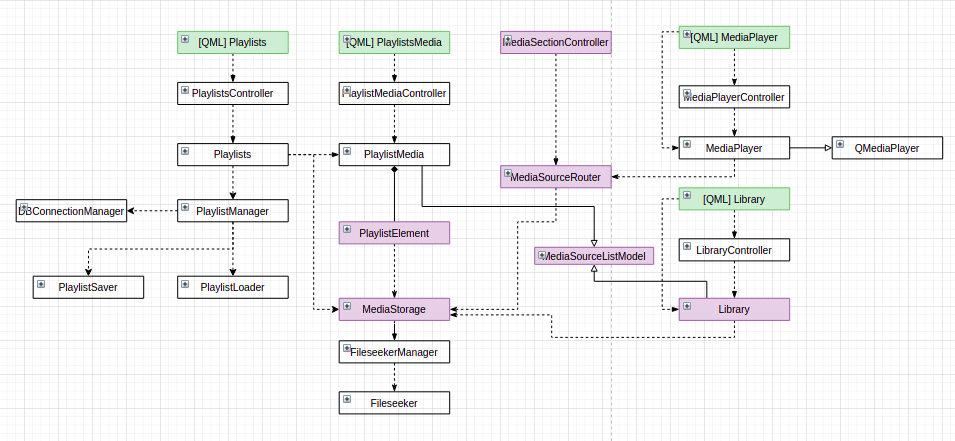
**Module description**



Pic. 1 UML class diagram

All the classes could be divided into 3 categories: models, controllers and helper classes used by the two previously mentioned types.

***Models***

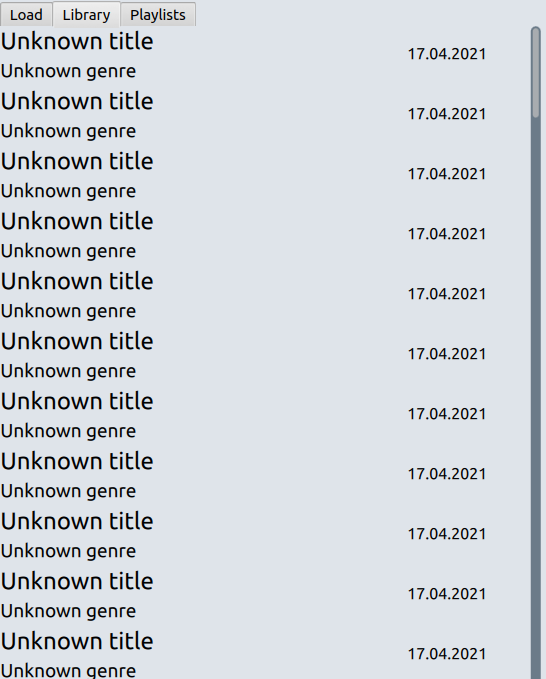
These classes were aimed to represent data containers. Library, MediaStorage, Playlists, PlaylistMedia, PlaylistElement are the representative of this type.

**MediaStorage**

Actually this class doesn`t have any visual representation straight - it is used by other models like PlaylistElement or Library as a main data hub. This class is responsible for indexing user`s machine for all allowed types of mediafiles in the background thread. When the new data arrives this class emits signals to Library notificating that it should update its view - representing found data to user.

**Library**

This class represents the section of loaded mediafiles:



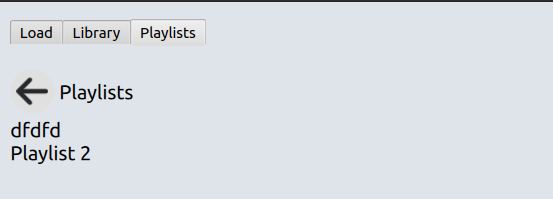
Being implementation for MediaSourceListModel class which in its turn inherits from QAbstractListModel, this class provides 3 core functions for accessing next/previous/by index mediafile. This functions are used by the mediaplayer (not straight) to acess appropriative mediafile. All the data is taken from the MediaStorage class.

**Playlists**

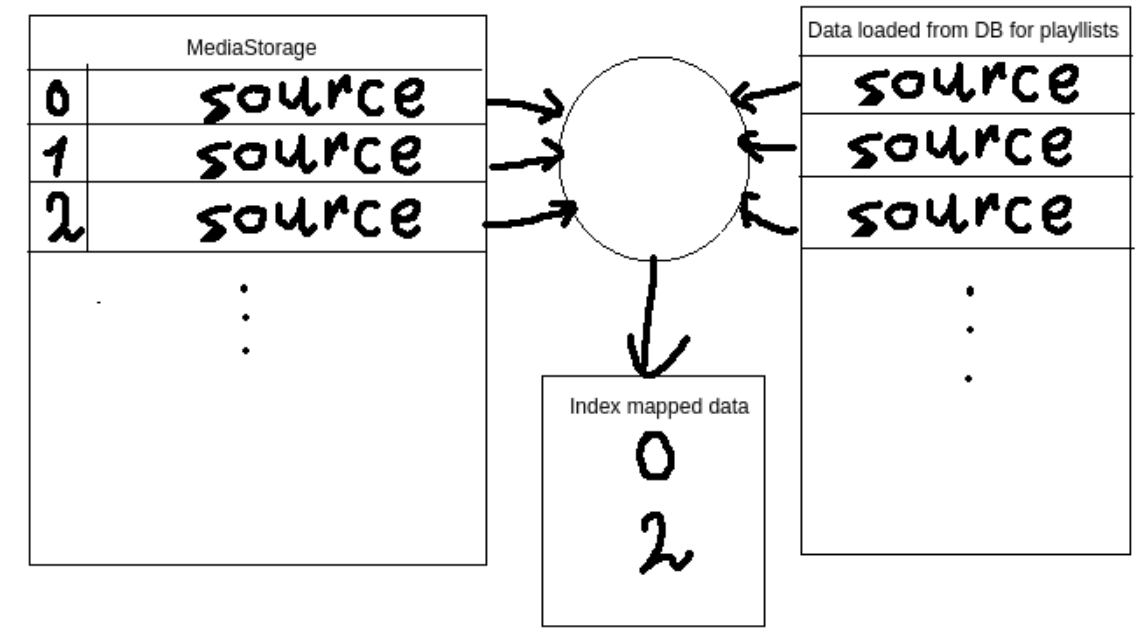
This class represents “headers” for playlists, main responsibilities:

- Operates on the data loaded from the database using PlaylistManager class

- Save all the data related to playlists` mediailes.



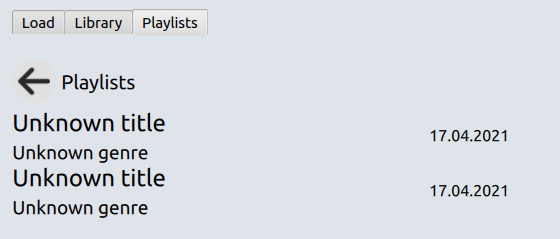
Using helper in-classes such as PlaylistManager this class loads the ready-to-use data from database. All the loading happens in the background thread after MediaStorage working thread which indexes all the mediafiles on the user`s machine is ended (the signal and slot mechanism is used to notify Playlists to start operating). All the available system mediafiles are passed to the manager where it uses two additional classes to load/save data to DB. Having those files in place playlist content is loaded and one by one compared with the available set to determine which pieces of playlists could be played successfully.On to the top level only the “index mapping” is passed (each found mediafile has the concrete index in MediaStorage).



PlaylistSaver and PlaylistLoader divide saving/loading db operations incapsulating SQL queries (few of them I left in the PLaylistManager as I was not sure whtether that class division is OK).

**PlaylistMedia**

This class represents the concrete playlist`s mediafiles.



Implementing MediaSourceListModel class this one will be used in the same way as the Library class. When the user clicks on some playlist name PlaylistElement object is loaded from Playlists and set as source for the PlaylistMedia.

**PlaylistElement**

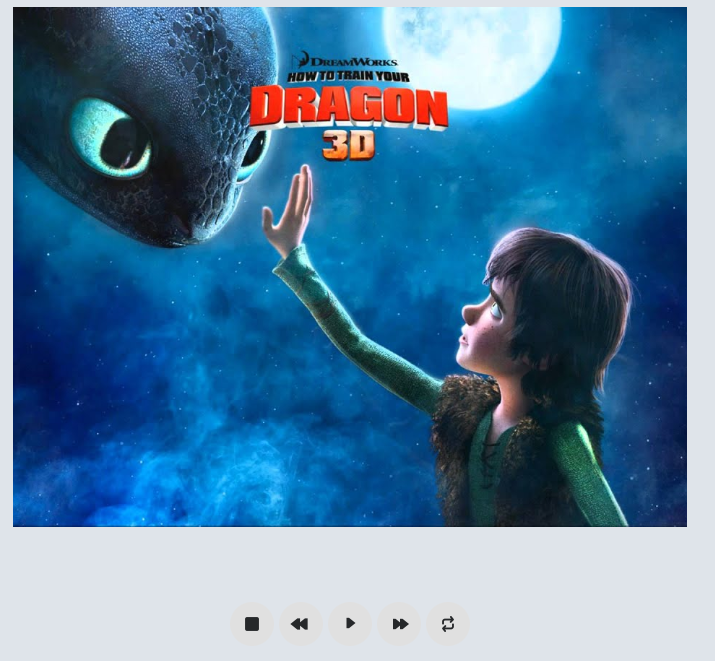
This class represents the container for playlist mediafiles. So techinically it is just a wrapper over the std::vector with some additional checks conditions implemented.

***Controllers***

Controllers classes are used to provide kind of a bridge between the view and model.

**MediaPlayerController**

This class is used for connecting MediaPlayer class which represents kind of a backend for a player and the view consisted of cover mediafile image/video output as well as incapsulating all needed control buttons to interact with the mediafiles.



For example, when the user clicks play/pause button (is represented via one view element) the appropriative signal is catched on the view and special controler`s slot is triggered to process this action. Then controller decides on which state should be assigned to this button (either play or paused), trigger approprative method in MediaPlayer and emit signal to the view where it will be catched and processed as well (change image for this button). In the same way the interaction with the slider is built - if the slider is moved it should stop playing the mediafile and wait until it will be free to continue playing the source.

**LibraryController**

This class is used to notify user about status of searching data on the user`s machine. Maybe it also would have a sense to use it when the user interacts with the actual mediafiles (f.e when user clicks on the mediafile which was loaded while searching but then deleted - we should check before trying to play that file).

**PlaylistsController**

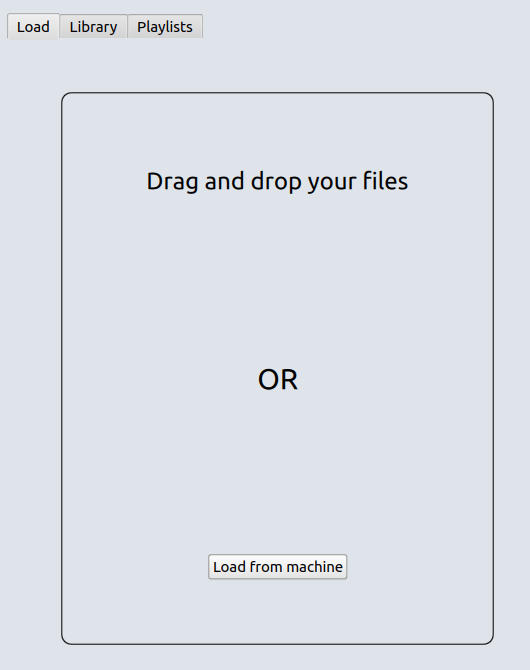
This class’s main responsibility is the same as for the MediaPlayerControoller. The difference - contex in which this piece is used - for playlists` work coordination. In general, I used this class to control playlists` creation (notify user if name for the newly created playlist is already taken or it is empty) as well as for controlling playlists` loading from the db.

**MediaSectionController**

This class was created to control switches between tabs. (However than I spot a bug which was produced by its work so the meaning was slightly changed. Actually it shouldn`t be called as it is anymore as the actual media sources are changed when the mediafile is chosen in some section (library/playlists).)

**MediaLoaderController**

This class is used to control loading files from the pc in the iteractive tab:



***Helper classes***

Also the bunch of helper classes are used. In this section I will describe only those which apply in the overall structure could be a bit fuzzy.

**VideoSurface**

As all the media manipulations are made using the Qt`s QMediaPlayer class we should also specify a special video adapter to show the actual video on the view. This is the main responsibility of this class - qml type VideoOutput uses this class to receive actual frames and render them to the screen.

**MediaSourceRouter**

This class is used to facilitate playing files from the different sources (library/playlist). It is kind of a unified place to get the next/prev mediafile without caring about from where the file would came. MediaSectionController class considered above is actually the trigger to control the source type specified in that class (either library or playlist).