MyTunes Assignment

Group: FOLLOW GREG ON GITHUB

Group members:

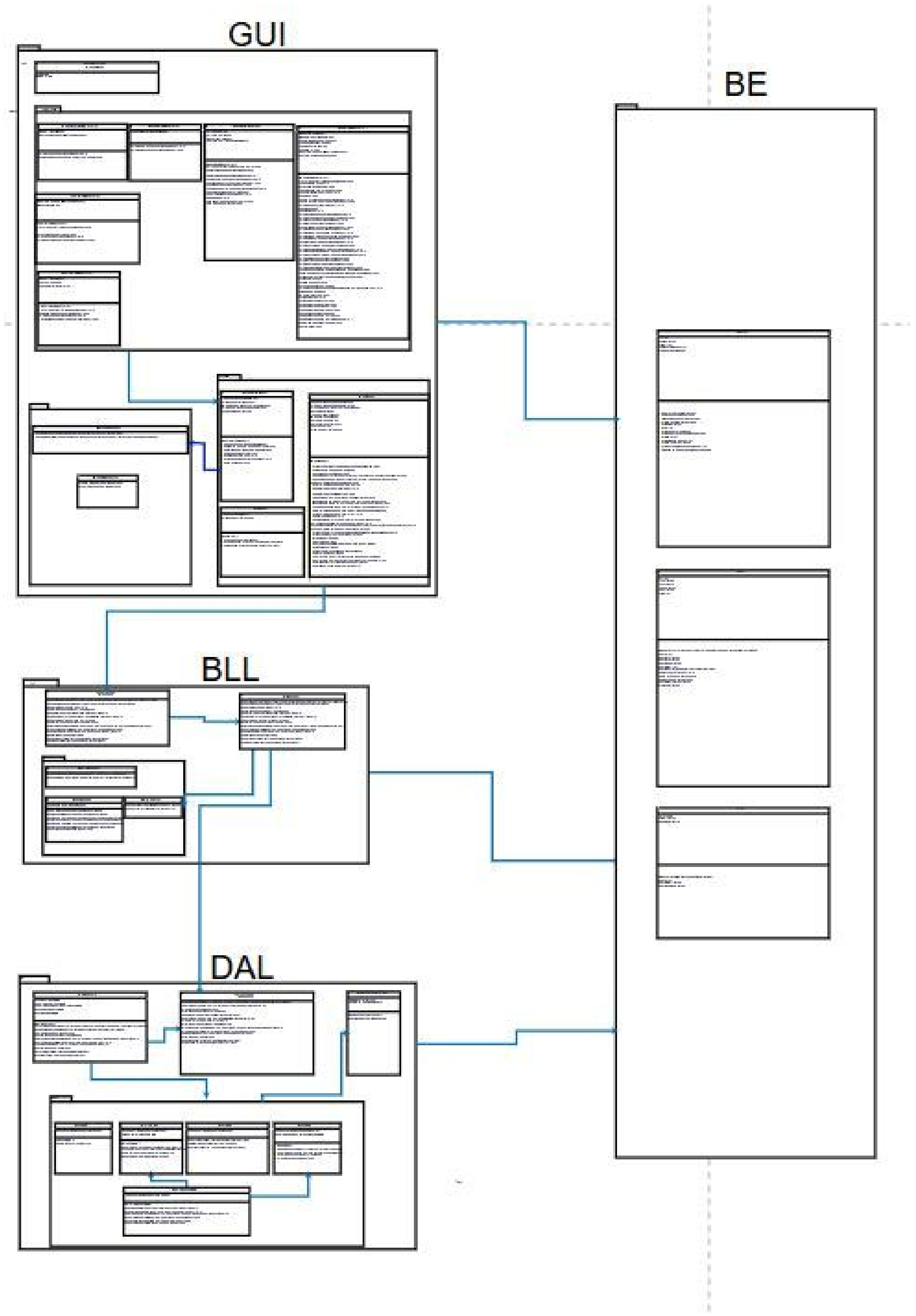
- Grzegorz Charyszczak
- Matti Brandt
- David Kalatzis

State of delivery

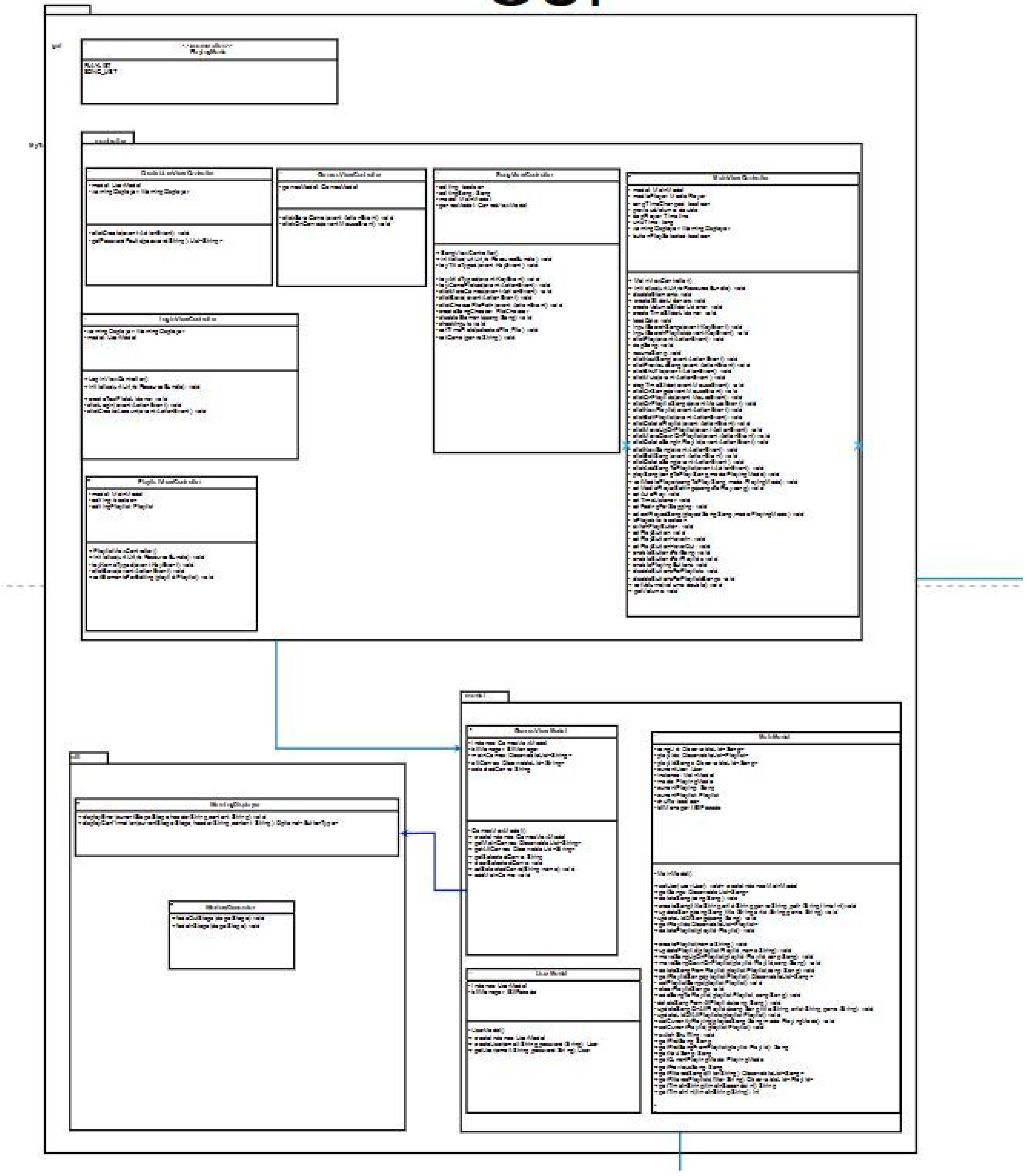
Both the functional and non-functional requirements have been fully met, as well as additional optional functionality that we have implemented in order to cement a more complete application.

- A mute button that enables the user to quickly mute and unmute the current song.
- A shuffle button, which when enabled will shuffle randomly between songs in the selected list, once the user clicks on the next/previous button or the song ends.
- A repeat button, if its enabled it will repeat the currently played song, but only if the song finishes automatically, else if the user clicks on the next/previous button it will act in the default way.
- A time slider allowing the user to control the time of the currently played song and adjust it accordingly.
- A textfield filter was added for the playlist as well.
- A subtle effect for fading in and out the current song when the user pauses and resumes.
- Improved user interface on a visual level with the use of simple CSS language.
- Changed the default window bar of the program to a unique customized one.
- Any pop-up window will cause the background window to lower slightly its opacity,as a way to differentiate the two windows from each other.
- When the app starts it has the same opacity effect as the pop-up windows, as well as when the user logs in and the next window appears.
- A system for disabling buttons so that the user won't perform actions that they shouldn't.
- Simple system for validating user's password and email address.
- A new user logging in window at the start up of the application, where the user has to sign in and log in with their account, thus enabling each user to have their own specific set of songs and playlists,unique to themselves.

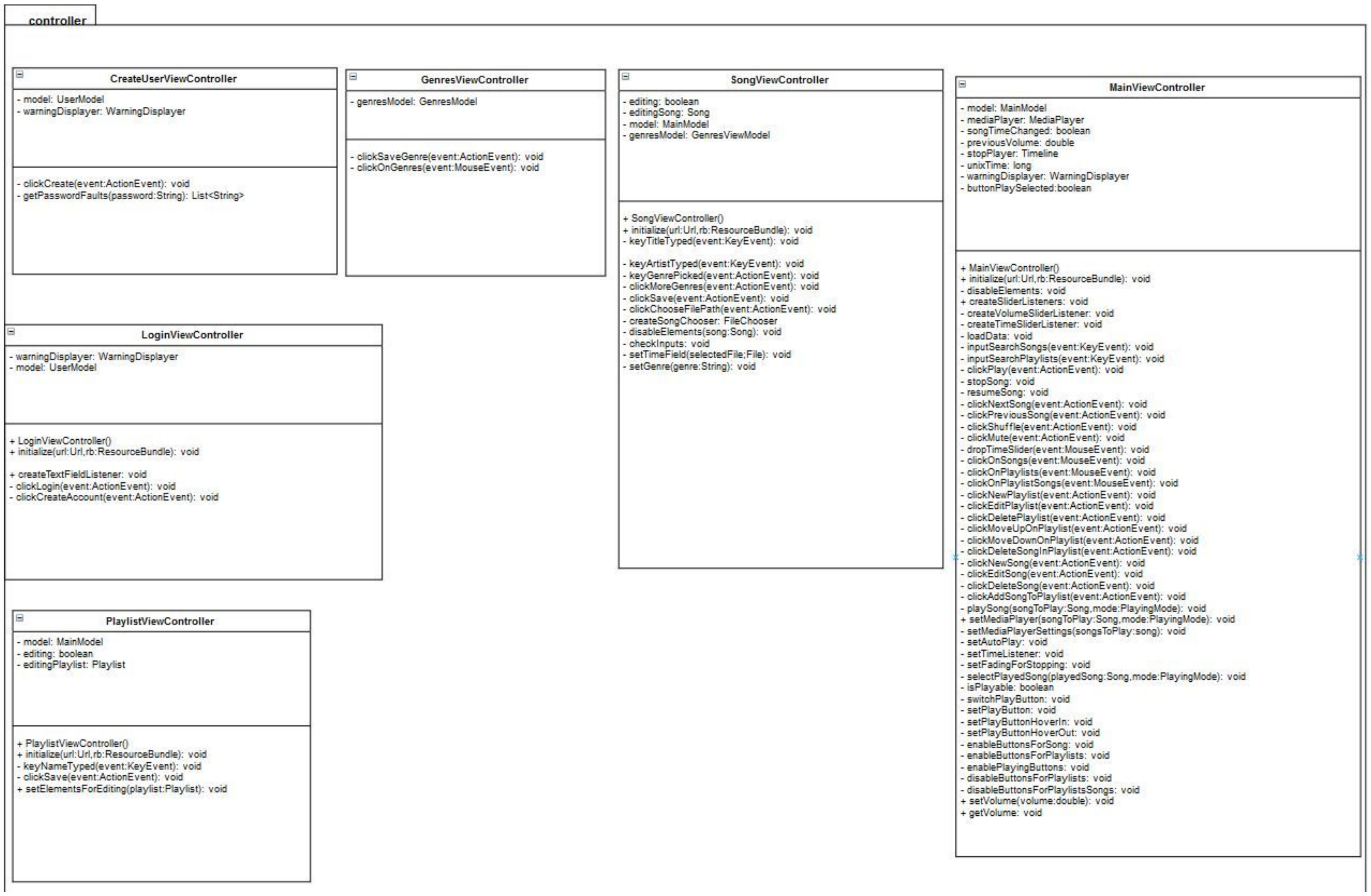
Application structure



GUI


gui
PlayingMode
PLAYLIST
SONG_LIST
controller
CreateUserViewController
GenreViewController
SongViewController
MainViewController
LoginViewController
PlaylistViewController
model
GenreViewModel
MainModel
UserModel
util
WarningDisplayer
WindowController

GUI - Controllers



CreateUserViewController

```
- model: UserModel
- warningDisplayer: WarningDisplayer

- clickCreate(event:ActionEvent): void
- getPasswordFaults(password:String): List<String>
```

GenresViewController

```
- genresModel: GenresModel

- clickSaveGenre(event:ActionEvent): void
- clickOnGenres(event:MouseEvent): void
```

SongViewController

```
- editing: boolean
- editingSong: Song
- model: MainModel
- genresModel: GenresViewModel

+ SongViewController()
+ initialize(url:Url,rb:ResourceBundle): void
- keyTitleTyped(event:KeyEvent): void

- keyArtistTyped(event:KeyEvent): void
- keyGenrePicked(event:ActionEvent): void
- clickMoreGenres(event:ActionEvent): void
- clickSave(event:ActionEvent): void
- clickChooseFilePath(event:ActionEvent): void
- createSongChooser: FileChooser
- disableElements(song:Song): void
- checkInputs: void
- setTimeField(selectedFile;File): void
- setGenre(genre:String): void
```

MainViewController

```
- model: MainModel
- mediaPlayer: MediaPlayer
- songTimeChanged: boolean
- previousVolume: double
- stopPlayer: Timeline
- unixTime: long
- warningDisplayer: WarningDisplayer
- buttonPlaySelected:boolean

+ MainViewController()
+ initialize(url:Url,rb:ResourceBundle): void
- disableElements: void
+ createSliderListeners: void
- createVolumeSliderListener: void
- createTimeSliderListener: void
- loadData: void
- inputSearchSongs(event:KeyEvent): void
- inputSearchPlaylists(event:KeyEvent): void
- clickPlay(event:ActionEvent): void
- stopSong: void
- resumeSong: void
- clickNextSong(event:ActionEvent): void
- clickPreviousSong(event:ActionEvent): void
- clickShuffle(event:ActionEvent): void
- clickMute(event:ActionEvent): void
- dropTimeSlider(event:MouseEvent): void
- clickOnSongs(event:MouseEvent): void
- clickOnPlaylists(event:MouseEvent): void
- clickOnPlaylistSongs(event:MouseEvent): void
- clickNewPlaylist(event:ActionEvent): void
- clickEditPlaylist(event:ActionEvent): void
- clickDeletePlaylist(event:ActionEvent): void
- clickMoveUpOnPlaylist(event:ActionEvent): void
- clickMoveDownOnPlaylist(event:ActionEvent): void
- clickDeleteSongInPlaylist(event:ActionEvent): void
- clickNewSong(event:ActionEvent): void
- clickEditSong(event:ActionEvent): void
- clickDeleteSong(event:ActionEvent): void
- clickAddSongToPlaylist(event:ActionEvent): void
- playSong(songToPlay:Song,mode:PlayingMode): void
+ setMediaPlayer(songToPlay:Song,mode:PlayingMode): void
- setMediaPlayerSettings(songsToPlay:song): void
- setAutoPlay: void
- setTimeListener: void
- setFadingForStopping: void
- selectPlayedSong(playedSong:Song,mode:PlayingMode): void
- isPlayable: boolean
- switchPlayButton: void
- setPlayButton: void
- setPlayButtonHoverIn: void
- setPlayButtonHoverOut: void
- enableButtonsForSong: void
- enableButtonsForPlaylists: void
- enablePlayingButtons: void
- disableButtonsForPlaylists: void
- disableButtonsForPlaylistsSongs: void
+ setVolume(volume:double): void
+ getVolume: void
```

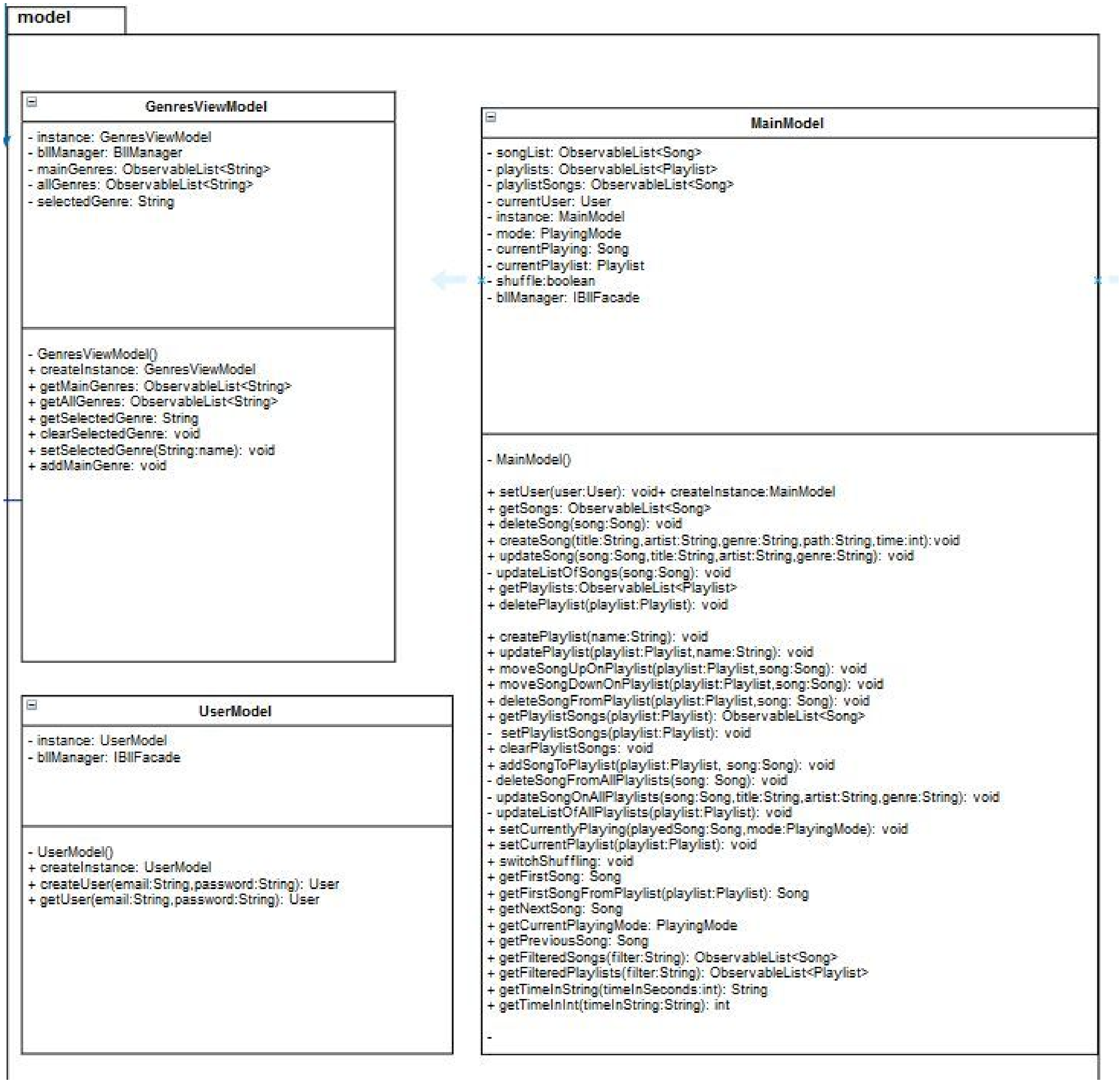
LoginViewController

```
- warningDisplayer: WarningDisplayer
- model: UserModel

+ LoginViewController()
+ initialize(url:Url,rb:ResourceBundle): void

+ createTextFieldListener: void
- clickLogin(event:ActionEvent): void
- clickCreateAccount(event:ActionEvent): void
```

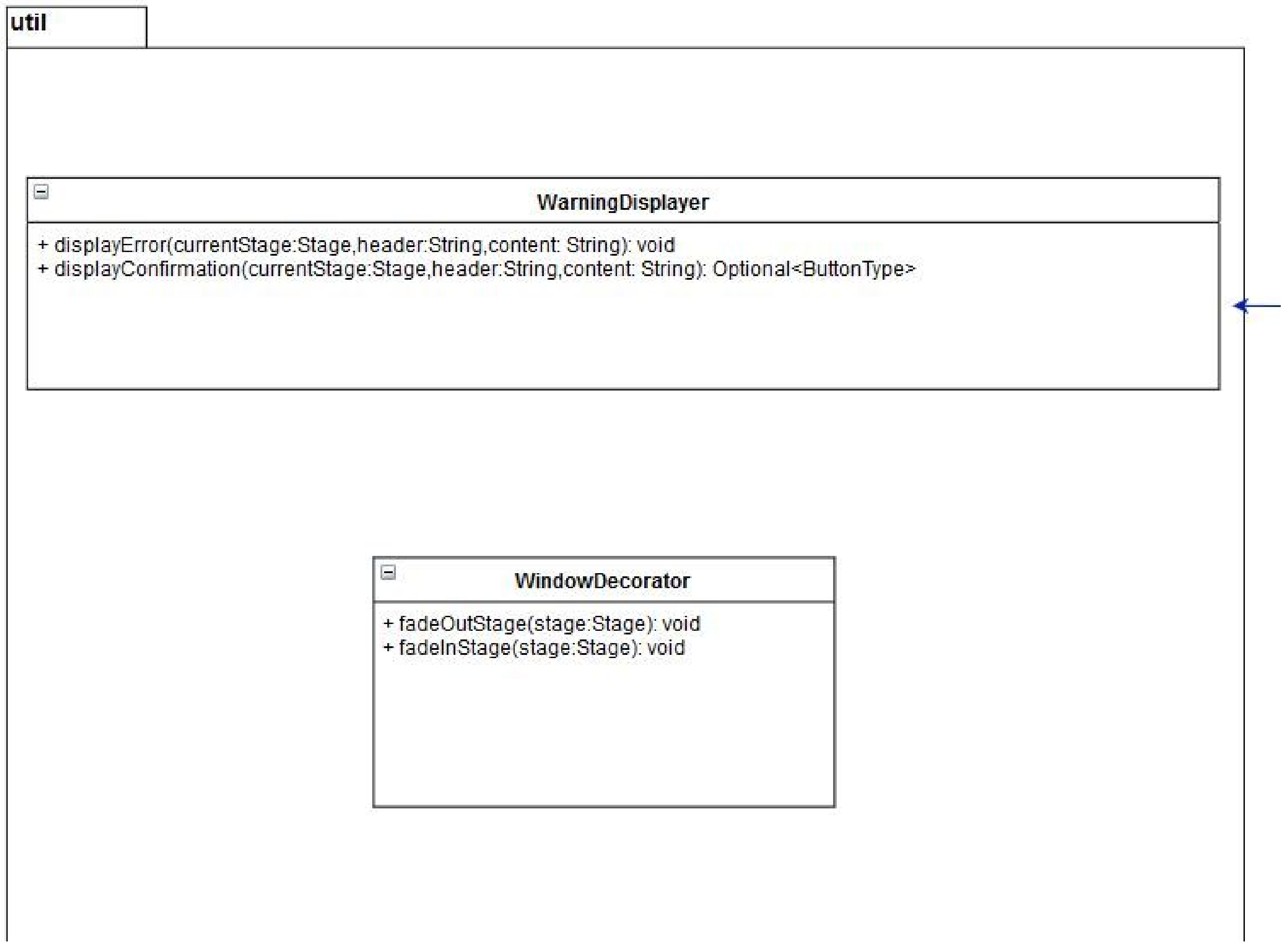
PlaylistViewController

```
- model: MainModel
- editing: boolean
- editingPlaylist: Playlist

+ PlaylistViewController()
+ initialize(url:Url,rb:ResourceBundle): void
- keyNameTyped(event:KeyEvent): void
- clickSave(event:ActionEvent): void
+ setElementsForEditing(playlist:Playlist): void
```

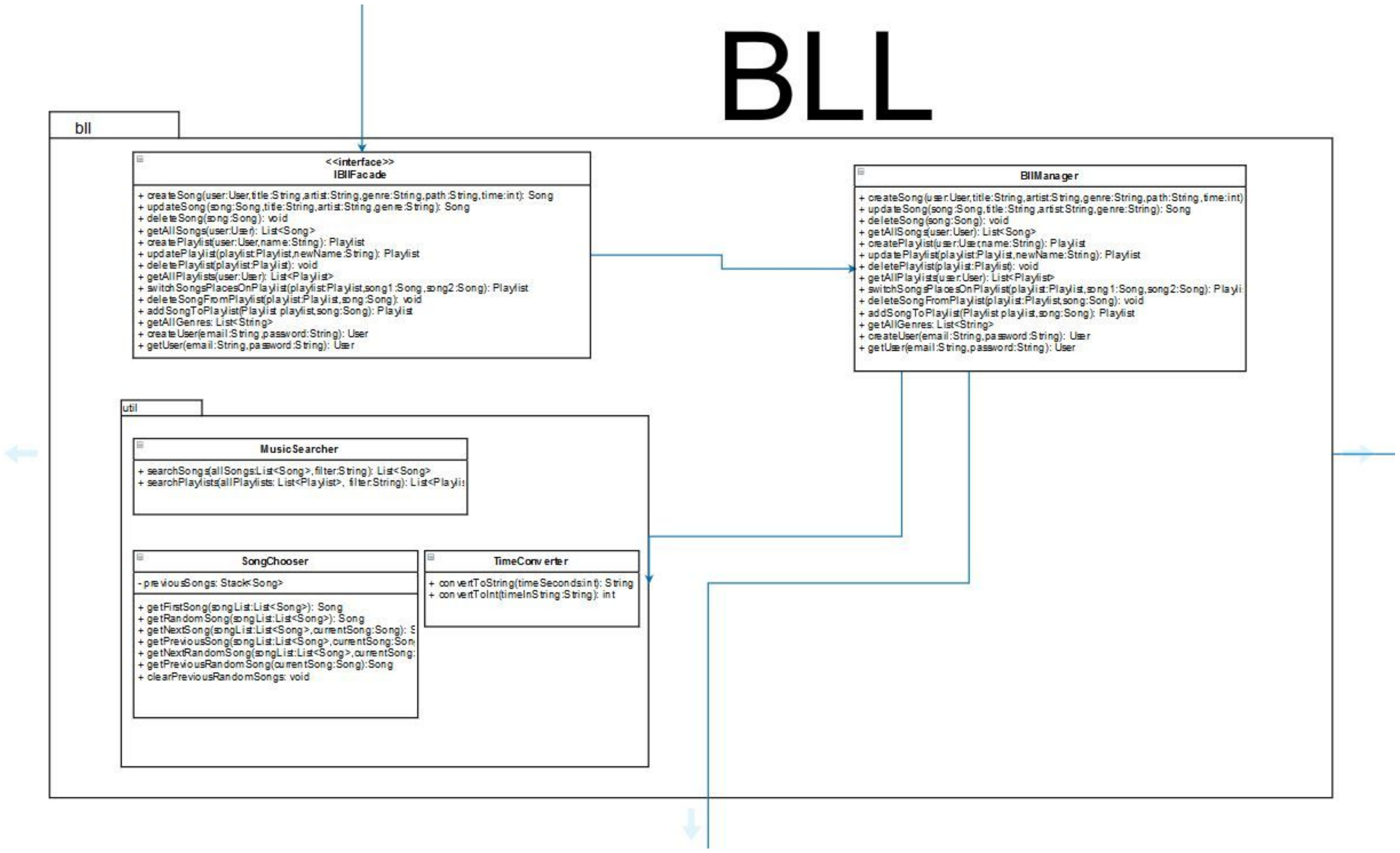
GUI - Models


model
GenresViewModel
- instance: GenresViewModel
- bllManager: BllManager
- mainGenres: ObservableList<String>
- allGenres: ObservableList<String>
- selectedGenre: String
- GenresViewModel()
+ createInstance: GenresViewModel
+ getMainGenres: ObservableList<String>
+ getAllGenres: ObservableList<String>
+ getSelectedGenre: String
+ clearSelectedGenre: void
+ setSelectedGenre(String:name): void
+ addMainGenre: void
UserModel
- instance: UserModel
- bllManager: IBllFacade
- UserModel()
+ createInstance: UserModel
+ createUser(email:String,password:String): User
+ getUser(email:String,password:String): User
MainModel
- songList: ObservableList<Song>
- playlists: ObservableList<Playlist>
- playlistSongs: ObservableList<Song>
- currentUser: User
- instance: MainModel
- mode: PlayingMode
- currentPlaying: Song
- currentPlaylist: Playlist
- shuffle:boolean
- bllManager: IBllFacade
- MainModel()
+ setUser(user:User): void+ createInstance:MainModel
+ getSongs: ObservableList<Song>
+ deleteSong(song:Song): void
+ createSong(title:String,artist:String,genre:String,path:String,time:int):void
+ updateSong(song:Song,title:String,artist:String,genre:String): void
- updateListOfSongs(song:Song): void
+ getPlaylists:ObservableList<Playlist>
+ deletePlaylist(playlist:Playlist): void
+ createPlaylist(name:String): void
+ updatePlaylist(playlist:Playlist,name:String): void
+ moveSongUpOnPlaylist(playlist:Playlist,song:Song): void
+ moveSongDownOnPlaylist(playlist:Playlist,song:Song): void
+ deleteSongFromPlaylist(playlist:Playlist,song: Song): void
+ getPlaylistSongs(playlist:Playlist): ObservableList<Song>
- setPlaylistSongs(playlist:Playlist): void
+ clearPlaylistSongs: void
+ addSongToPlaylist(playlist:Playlist, song:Song): void
- deleteSongFromAllPlaylists(song: Song): void
- updateSongOnAllPlaylists(song:Song,title:String,artist:String,genre:String): void
- updateListOfAllPlaylists(playlist:Playlist): void
+ setCurrentlyPlaying(playedSong:Song,mode:PlayingMode): void
+ setCurrentPlaylist(playlist:Playlist): void
+ switchShuffling: void
+ getFirstSong: Song
+ getFirstSongFromPlaylist(playlist:Playlist): Song
+ getNextSong: Song
+ getCurrentPlayingMode: PlayingMode
+ getPreviousSong: Song
+ getFilteredSongs(filter:String): ObservableList<Song>
+ getFilteredPlaylists(filter:String): ObservableList<Playlist>
+ getTimeInString(timeInSeconds:int): String
+ getTimeInInt(timeInString:String): int
-

GUI - Util

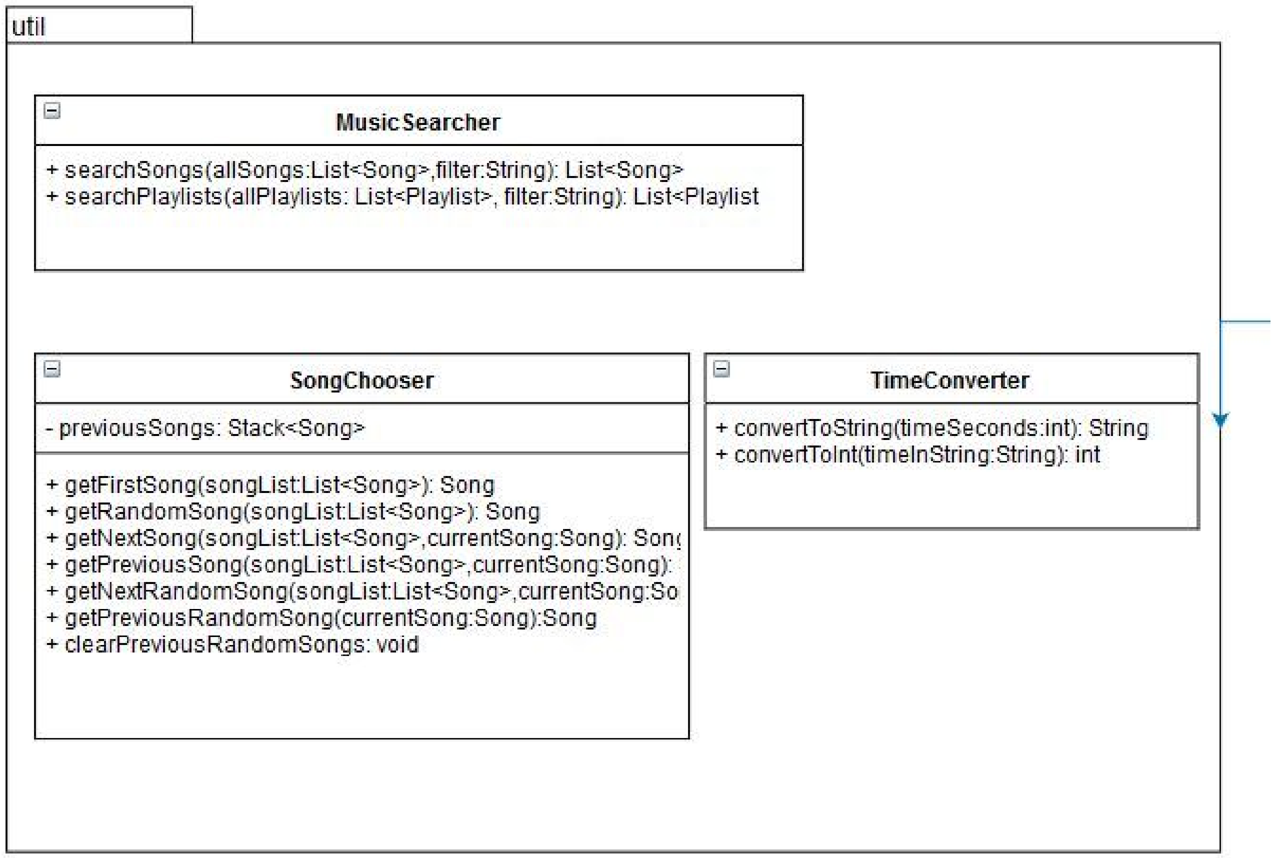

util
WarningDisplayer
+ displayError(currentStage:Stage,header:String,content: String): void
+ displayConfirmation(currentStage:Stage,header:String,content: String): Optional<ButtonType>
WindowDecorator
+ fadeOutStage(stage:Stage): void
+ fadeInStage(stage:Stage): void

BLL



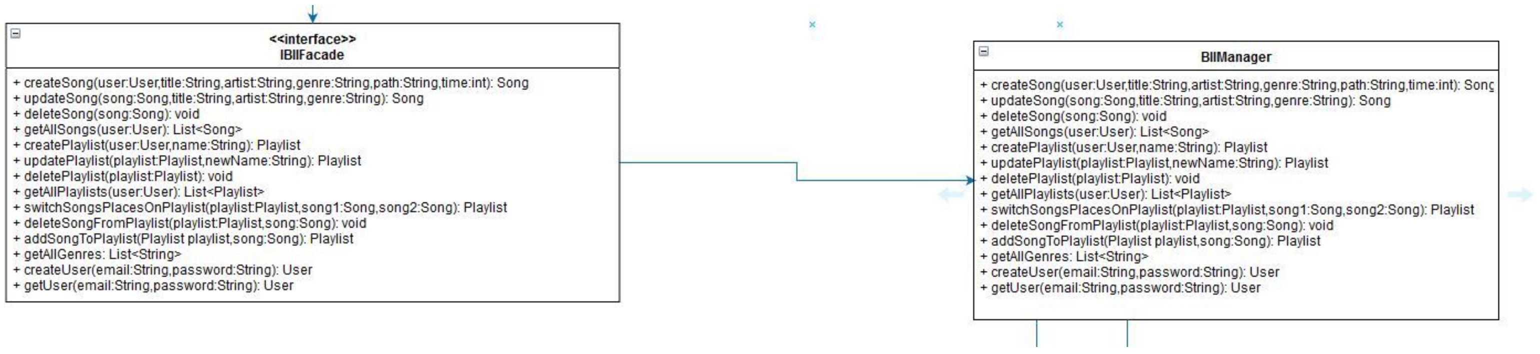
```
bll

<<interface>>
IBllFacade
+ createSong(user:User,title:String,artist:String,genre:String,path:String,time:int): Song
+ updateSong(song:Song,title:String,artist:String,genre:String): Song
+ deleteSong(song:Song): void
+ getAllSongs(user:User): List<Song>
+ createPlaylist(user:User,name:String): Playlist
+ updatePlaylist(playlist:Playlist,newName:String): Playlist
+ deletePlaylist(playlist:Playlist): void
+ getAllPlaylists(user:User): List<Playlist>
+ switchSongsPlacesOnPlaylist(playlist:Playlist,song1:Song,song2:Song): Playlist
+ deleteSongFromPlaylist(playlist:Playlist,song:Song): void
+ addSongToPlaylist(Playlist playlist,song:Song): Playlist
+ getAllGenres: List<String>
+ createUser(email:String,password:String): User
+ getUser(email:String,password:String): User

BllManager
+ createSong(user:User,title:String,artist:String,genre:String,path:String,time:int)
+ updateSong(song:Song,title:String,artist:String,genre:String): Song
+ deleteSong(song:Song): void
+ getAllSongs(user:User): List<Song>
+ createPlaylist(user:User,name:String): Playlist
+ updatePlaylist(playlist:Playlist,newName:String): Playlist
+ deletePlaylist(playlist:Playlist): void
+ getAllPlaylists(user:User): List<Playlist>
+ switchSongsPlacesOnPlaylist(playlist:Playlist,song1:Song,song2:Song): Playli
+ deleteSongFromPlaylist(playlist:Playlist,song:Song): void
+ addSongToPlaylist(Playlist playlist,song:Song): Playlist
+ getAllGenres: List<String>
+ createUser(email:String,password:String): User
+ getUser(email:String,password:String): User

util

MusicSearcher
+ searchSongs(allSongs:List<Song>,filter:String): List<Song>
+ searchPlaylists(allPlaylists: List<Playlist>, filter:String): List<Playlis

SongChooser
- previousSongs: Stack<Song>
+ getFirstSong(songList:List<Song>): Song
+ getRandomSong(songList:List<Song>): Song
+ getNextSong(songList:List<Song>,currentSong:Song): S
+ getPreviousSong(songList:List<Song>,currentSong:Son
+ getNextRandomSong(songList:List<Song>,currentSong:
+ getPreviousRandomSong(currentSong:Song):Song
+ clearPreviousRandomSongs: void

TimeConverter
+ convertToString(timeSeconds:int): String
+ convertToInt(timeInString:String): int
```

BLL - Util



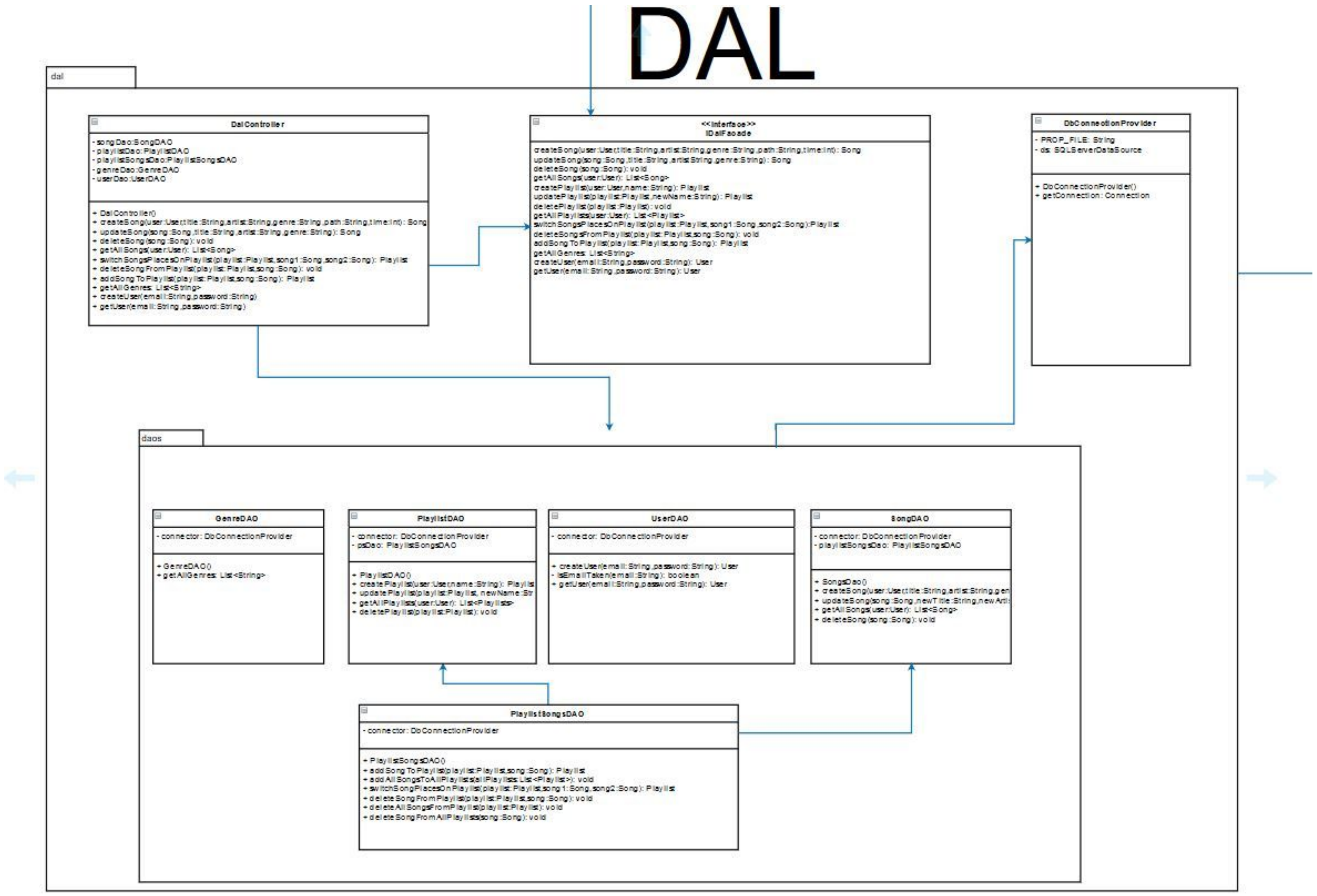
```
MusicSearcher
+ searchSongs(allSongs:List<Song>,filter:String): List<Song>
+ searchPlaylists(allPlaylists: List<Playlist>, filter:String): List<Playlist

SongChooser
- previousSongs: Stack<Song>
+ getFirstSong(songList:List<Song>): Song
+ getRandomSong(songList:List<Song>): Song
+ getNextSong(songList:List<Song>,currentSong:Song): Son
+ getPreviousSong(songList:List<Song>,currentSong:Song):
+ getNextRandomSong(songList:List<Song>,currentSong:So
+ getPreviousRandomSong(currentSong:Song):Song
+ clearPreviousRandomSongs: void

TimeConverter
+ convertToString(timeSeconds:int): String
+ convertToInt(timeInString:String): int
```

BLL - Facade & Manager

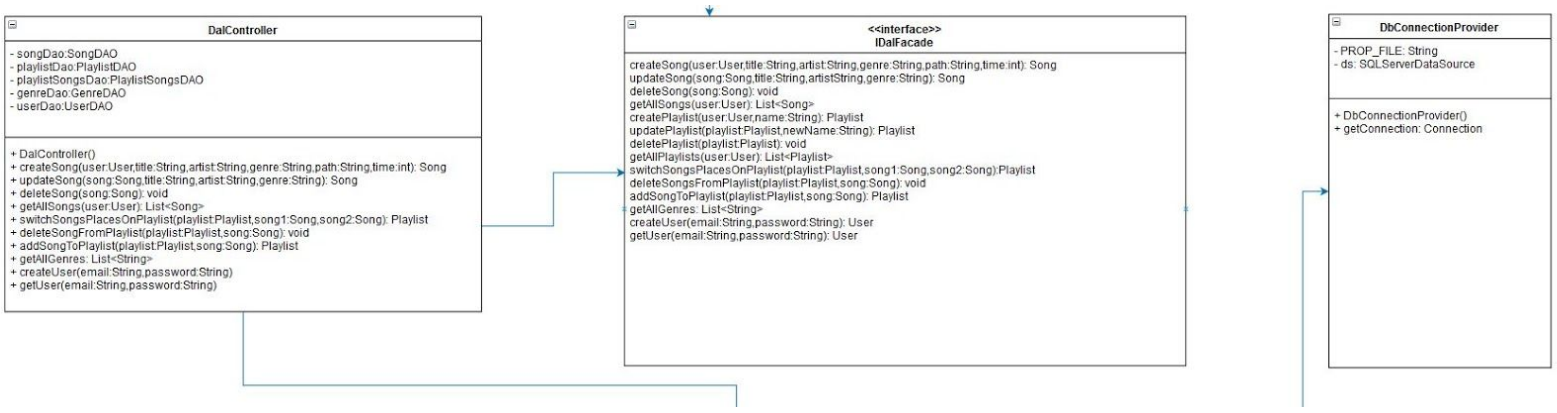


```
<<interface>>
IBllFacade
+ createSong(user:User,title:String,artist:String,genre:String,path:String,time:int): Song
+ updateSong(song:Song,title:String,artist:String,genre:String): Song
+ deleteSong(song:Song): void
+ getAllSongs(user:User): List<Song>
+ createPlaylist(user:User,name:String): Playlist
+ updatePlaylist(playlist:Playlist,newName:String): Playlist
+ deletePlaylist(playlist:Playlist): void
+ getAllPlaylists(user:User): List<Playlist>
+ switchSongsPlacesOnPlaylist(playlist:Playlist,song1:Song,song2:Song): Playlist
+ deleteSongFromPlaylist(playlist:Playlist,song:Song): void
+ addSongToPlaylist(Playlist playlist,song:Song): Playlist
+ getAllGenres: List<String>
+ createUser(email:String,password:String): User
+ getUser(email:String,password:String): User

BllManager
+ createSong(user:User,title:String,artist:String,genre:String,path:String,time:int): Son
+ updateSong(song:Song,title:String,artist:String,genre:String): Song
+ deleteSong(song:Song): void
+ getAllSongs(user:User): List<Song>
+ createPlaylist(user:User,name:String): Playlist
+ updatePlaylist(playlist:Playlist,newName:String): Playlist
+ deletePlaylist(playlist:Playlist): void
+ getAllPlaylists(user:User): List<Playlist>
+ switchSongsPlacesOnPlaylist(playlist:Playlist,song1:Song,song2:Song): Playlist
+ deleteSongFromPlaylist(playlist:Playlist,song:Song): void
+ addSongToPlaylist(Playlist playlist,song:Song): Playlist
+ getAllGenres: List<String>
+ createUser(email:String,password:String): User
+ getUser(email:String,password:String): User
```

DAL


dal
DalController
- songDao:SongDAO
- playlistDao:PlaylistDAO
- playlistSongsDao:PlaylistSongsDAO
- genreDao:GenreDAO
- userDao:UserDAO
+ DalController()
+ createSong(user:User,title:String,artist:String,genre:String,path:String,time:Int): Song
+ updateSong(song:Song,title:String,artist:String,genre:String): Song
+ deleteSong(song:Song): void
+ getAllSongs(user:User): List<Song>
+ switchSongsPlacesOnPlaylist(playlist:Playlist,song1:Song,song2:Song): Playlist
+ deleteSongFromPlaylist(playlist:Playlist,song:Song): void
+ addSongToPlaylist(playlist:Playlist,song:Song): Playlist
+ getAllGenres: List<String>
+ createUser(email:String,password:String)
+ getUser(email:String,password:String)
<<interface>>
IDalFacade
createSong(user:User,title:String,artist:String,genre:String,path:String,time:Int): Song
updateSong(song:Song,title:String,artistString,genre:String): Song
deleteSong(song:Song): void
getAllSongs(user:User): List<Song>
createPlaylist(user:User,name:String): Playlist
updatePlaylist(playlist:Playlist,newName:String): Playlist
deletePlaylist(playlist:Playlist): void
getAllPlaylists(user:User): List<Playlist>
switchSongsPlacesOnPlaylist(playlist:Playlist,song1:Song,song2:Song):Playlist
deleteSongsFromPlaylist(playlist:Playlist,song:Song): void
addSongToPlaylist(playlist:Playlist,song:Song): Playlist
getAllGenres: List<String>
createUser(email:String,password:String): User
getUser(email:String,password:String): User
DbConnectionProvider
- PROP_FILE: String
- ds: SQLServerDataSource
+ DbConnectionProvider()
+ getConnection: Connection
daos
GenreDAO
- connector: DbConnectionProvider
+ GenreDAO()
+ getAllGenres: List<String>
PlaylistDAO
- connector: DbConnectionProvider
- psDao: PlaylistSongsDAO
+ PlaylistDAO()
+ createPlaylist(user:User,name:String): Playlis
+ updatePlaylist(playlist:Playlist, newName:Str
+ getAllPlaylists(user:User): List<Playlists>
+ deletePlaylist(playlist:Playlist): void
UserDAO
- connector: DbConnectionProvider
+ createUser(email:String,password:String): User
- isEmailTaken(email:String): boolean
+ getUser(email:String,password:String): User
SongDAO
- connector: DbConnectionProvider
- playlistSongsDao: PlaylistSongsDAO
+ SongsDao()
+ createSong(user:User,title:String,artist:String,gen
+ updateSong(song:Song,newTitle:String,newArti
+ getAllSongs(user:User): List<Song>
+ deleteSong(song:Song): void
PlaylistSongsDAO
- connector: DbConnectionProvider
+ PlaylistSongsDAO()
+ addSongToPlaylist(playlist:Playlist,song:Song): Playlist
+ addAllSongsToAllPlaylists(allPlaylists:List<Playlist>): void
+ switchSongPlacesOnPlaylist(playlist:Playlist,song1:Song,song2:Song): Playlist
+ deleteSongFromPlaylist(playlist:Playlist,song:Song): void
+ deleteAllSongsFromPlaylist(playlist:Playlist): void
+ deleteSongFromAllPlaylists(song:Song): void

DAL - Facade & Manager

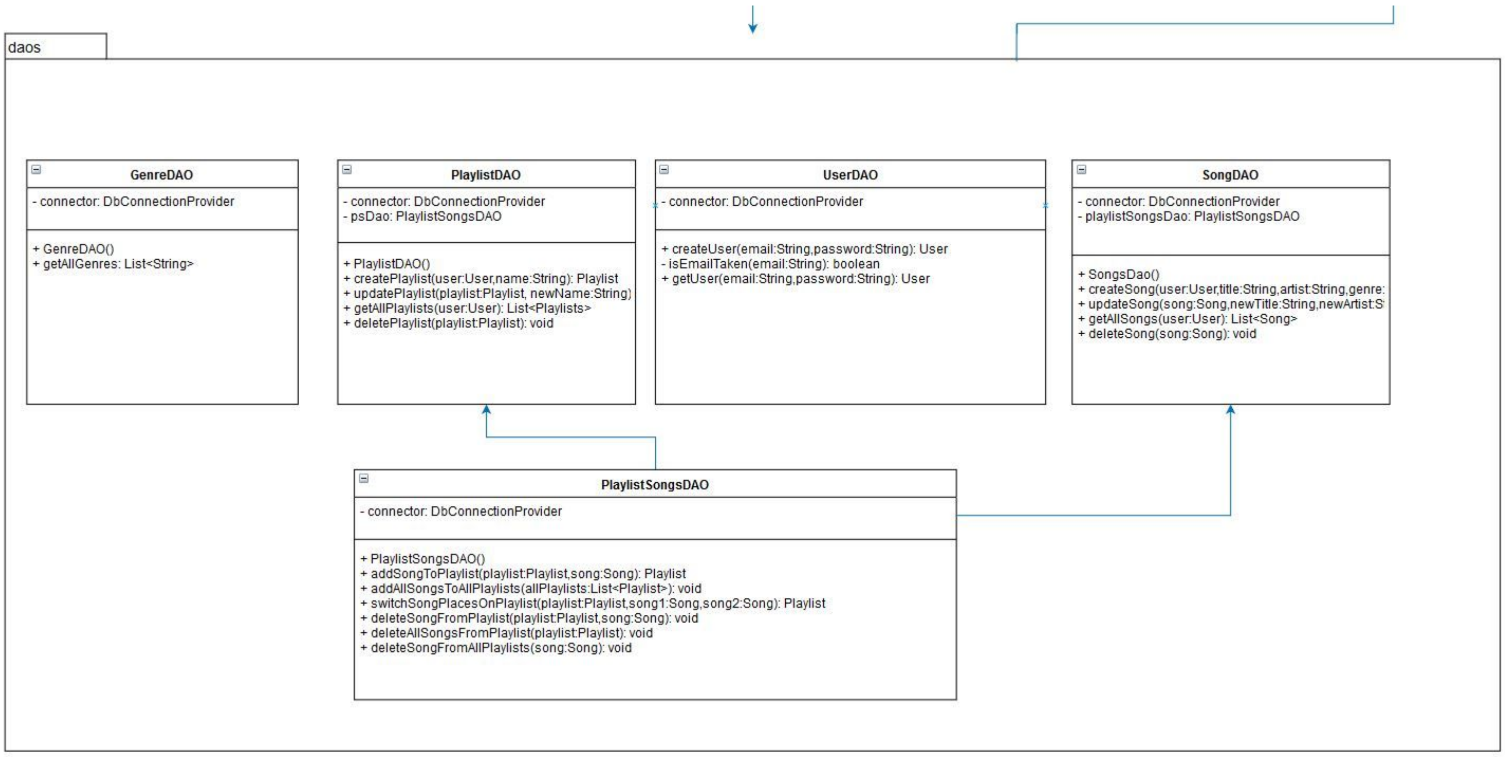


```
DalController
- songDao:SongDAO
- playlistDao:PlaylistDAO
- playlistSongsDao:PlaylistSongsDAO
- genreDao:GenreDAO
- userDao:UserDAO

+ DalController()
+ createSong(user:User,title:String,artist:String,genre:String,path:String,time:int): Song
+ updateSong(song:Song,title:String,artist:String,genre:String): Song
+ deleteSong(song:Song): void
+ getAllSongs(user:User): List<Song>
+ switchSongsPlacesOnPlaylist(playlist:Playlist,song1:Song,song2:Song): Playlist
+ deleteSongFromPlaylist(playlist:Playlist,song:Song): void
+ addSongToPlaylist(playlist:Playlist,song:Song): Playlist
+ getAllGenres: List<String>
+ createUser(email:String,password:String)
+ getUser(email:String,password:String)
```

```
<<interface>>
IDalFacade
createSong(user:User,title:String,artist:String,genre:String,path:String,time:int): Song
updateSong(song:Song,title:String,artistString,genre:String): Song
deleteSong(song:Song): void
getAllSongs(user:User): List<Song>
createPlaylist(user:User,name:String): Playlist
updatePlaylist(playlist:Playlist,newName:String): Playlist
deletePlaylist(playlist:Playlist): void
getAllPlaylists(user:User): List<Playlist>
switchSongsPlacesOnPlaylist(playlist:Playlist,song1:Song,song2:Song):Playlist
deleteSongsFromPlaylist(playlist:Playlist,song:Song): void
addSongToPlaylist(playlist:Playlist,song:Song): Playlist
getAllGenres: List<String>
createUser(email:String,password:String): User
getUser(email:String,password:String): User
```

```
DbConnectionProvider
- PROP_FILE: String
- ds: SQLServerDataSource

+ DbConnectionProvider()
+ getConnection: Connection
```

DAL - DAOs



```
GenreDAO
- connector: DbConnectionProvider

+ GenreDAO()
+ getAllGenres: List<String>
```

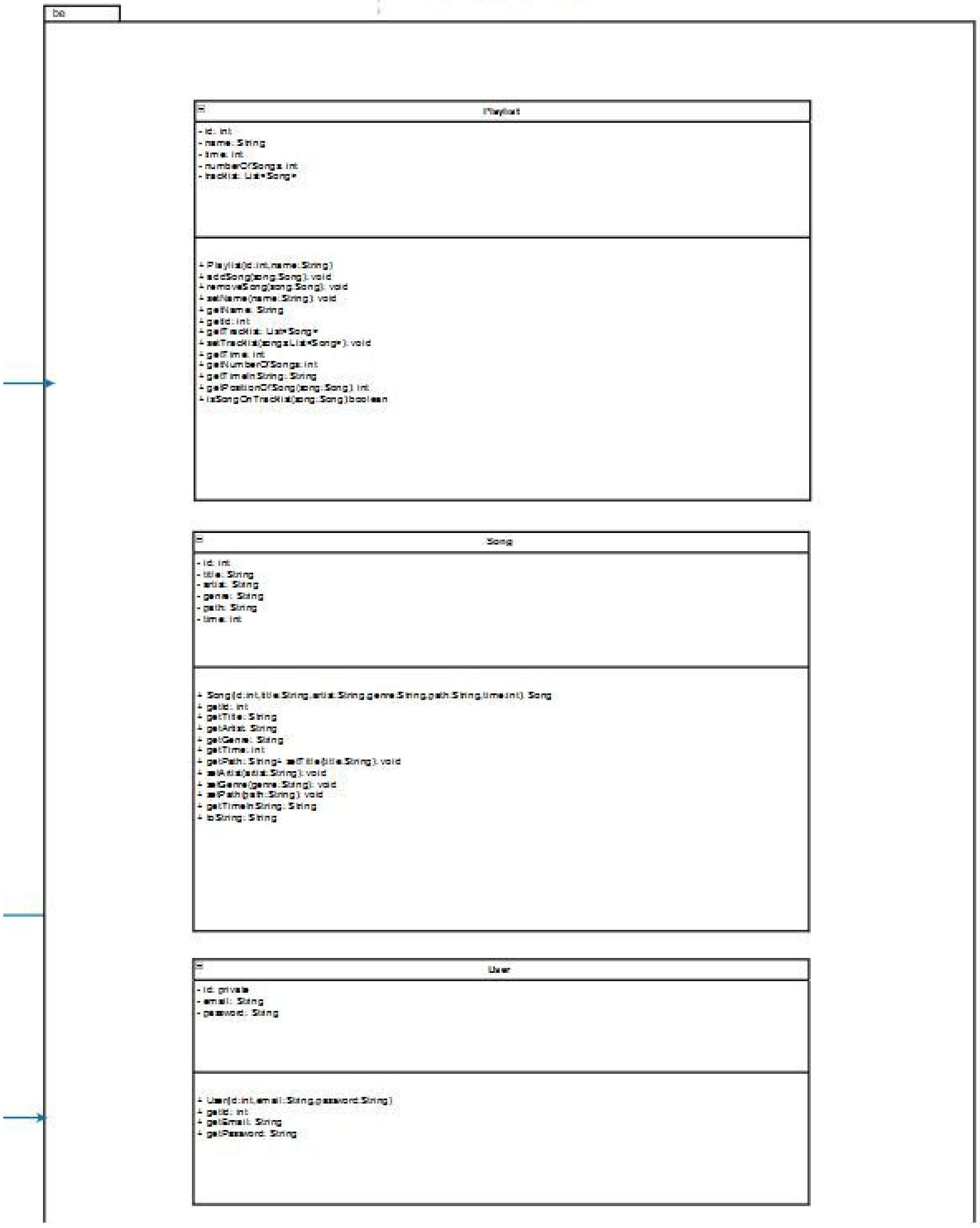
```
PlaylistDAO
- connector: DbConnectionProvider
- psDao: PlaylistSongsDAO

+ PlaylistDAO()
+ createPlaylist(user:User,name:String): Playlist
+ updatePlaylist(playlist:Playlist, newName:String)
+ getAllPlaylists(user:User): List<Playlists>
+ deletePlaylist(playlist:Playlist): void
```

```
UserDAO
- connector: DbConnectionProvider

+ createUser(email:String,password:String): User
- isEmailTaken(email:String): boolean
+ getUser(email:String,password:String): User
```

```
SongDAO
- connector: DbConnectionProvider
- playlistSongsDao: PlaylistSongsDAO

+ SongsDao()
+ createSong(user:User,title:String,artist:String,genre:
+ updateSong(song:Song,newTitle:String,newArtist:S
+ getAllSongs(user:User): List<Song>
+ deleteSong(song:Song): void
```

```
PlaylistSongsDAO
- connector: DbConnectionProvider

+ PlaylistSongsDAO()
+ addSongToPlaylist(playlist:Playlist,song:Song): Playlist
+ addAllSongsToAllPlaylists(allPlaylists:List<Playlist>): void
+ switchSongPlacesOnPlaylist(playlist:Playlist,song1:Song,song2:Song): Playlist
+ deleteSongFromPlaylist(playlist:Playlist,song:Song): void
+ deleteAllSongsFromPlaylist(playlist:Playlist): void
+ deleteSongFromAllPlaylists(song:Song): void
```

BE

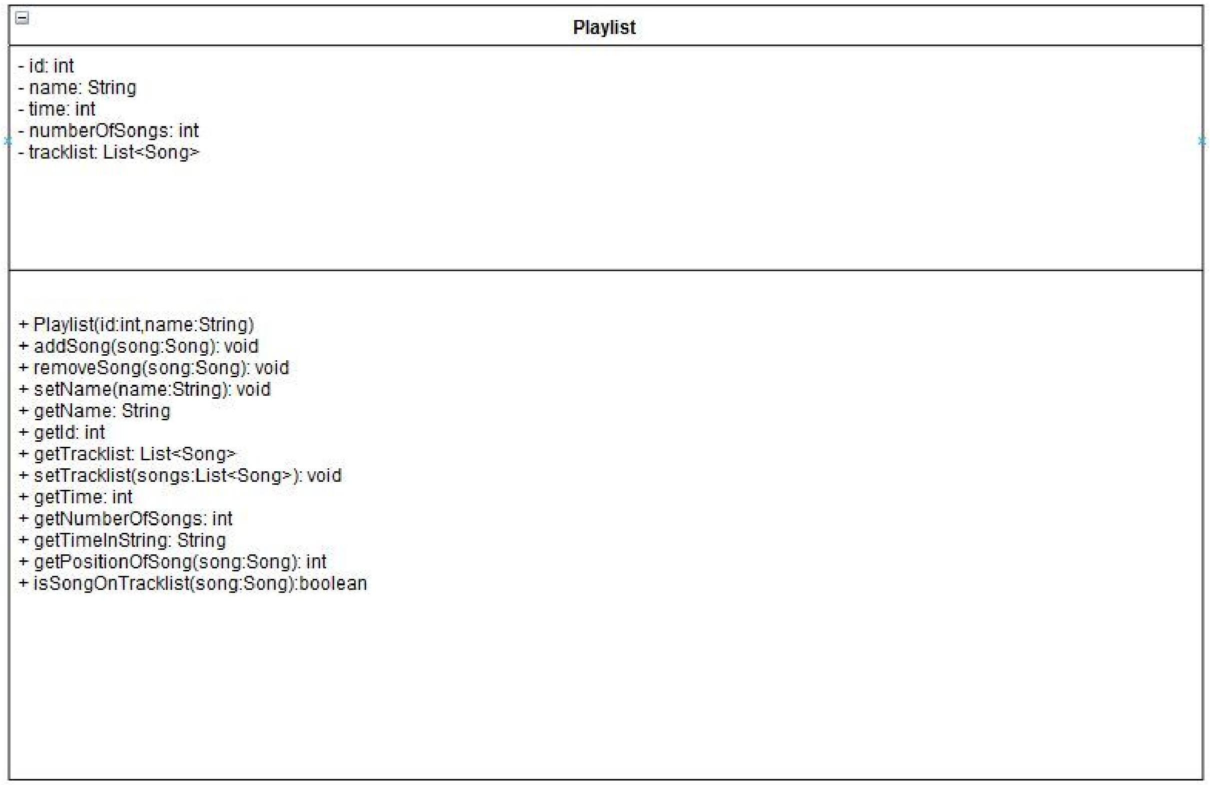


```
Playlist
- id: int
- name: String
- time: int
- numberOfSongs: int
- trackList: List<Song>

+ Playlist(id:int,name:String)
+ addSong(song:Song): void
+ removeSong(song:Song): void
+ setName(name:String): void
+ getName: String
+ getId: int
+ getTrackList: List<Song>
+ setTrackList(songs:List<Song>): void
+ getTime: int
+ getNumberOfSongs: int
+ getTimeInString: String
+ getPositionOfSong(song:Song): int
+ isSongOnTrackList(song:Song):boolean
```

```
Song
- id: int
- title: String
- artist: String
- genre: String
- path: String
- time: int

+ Song(id:int,title:String,artist:String,genre:String,path:String,time:int): Song
+ getId: int
+ getTitle: String
+ getArtist: String
+ getGenre: String
+ getTime: int
+ getPath: String+ setTitle(title:String): void
+ setArtist(artist:String): void
+ setGenre(genre:String): void
+ setPath(path:String): void
+ getTimeInString: String
+ toString: String
```

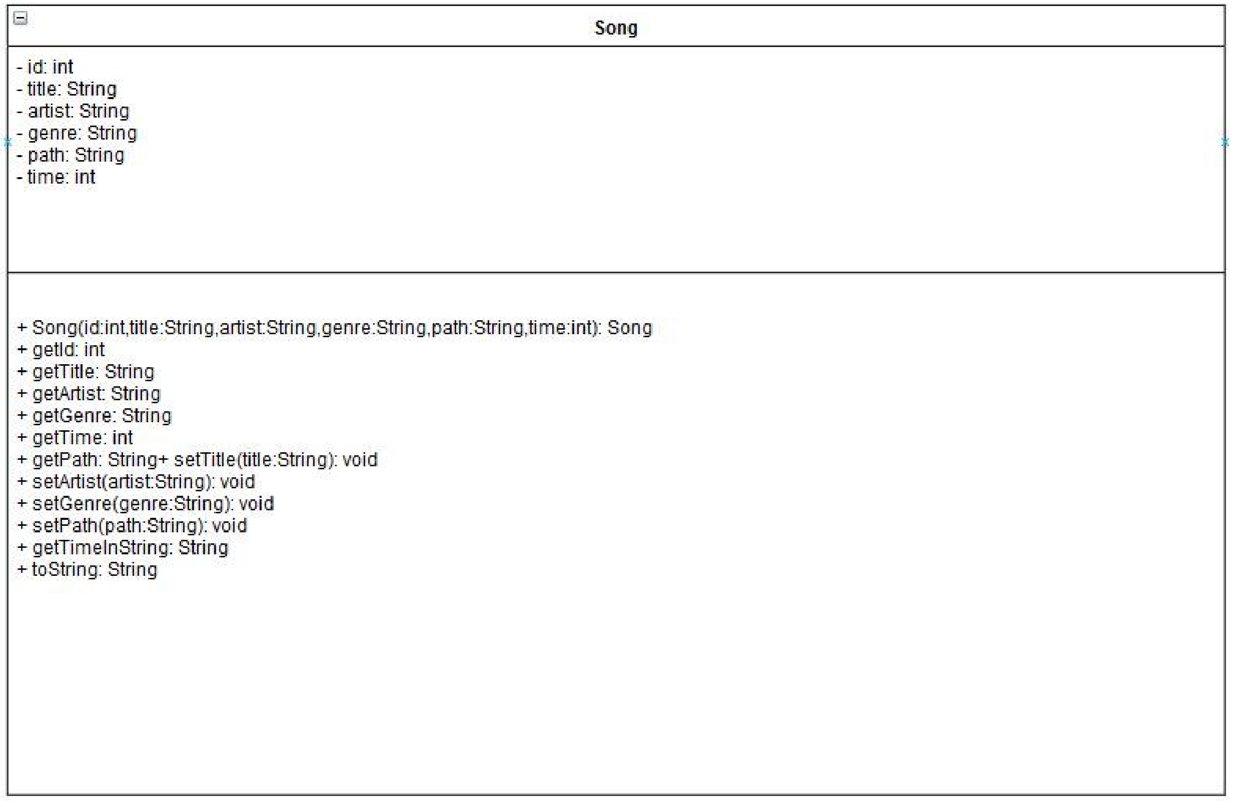
```
User
- id: private
- email: String
- password: String

+ User(id:int,email:String,password:String)
+ getId: int
+ getEmail: String
+ getPassword: String
```

BE - Playlist



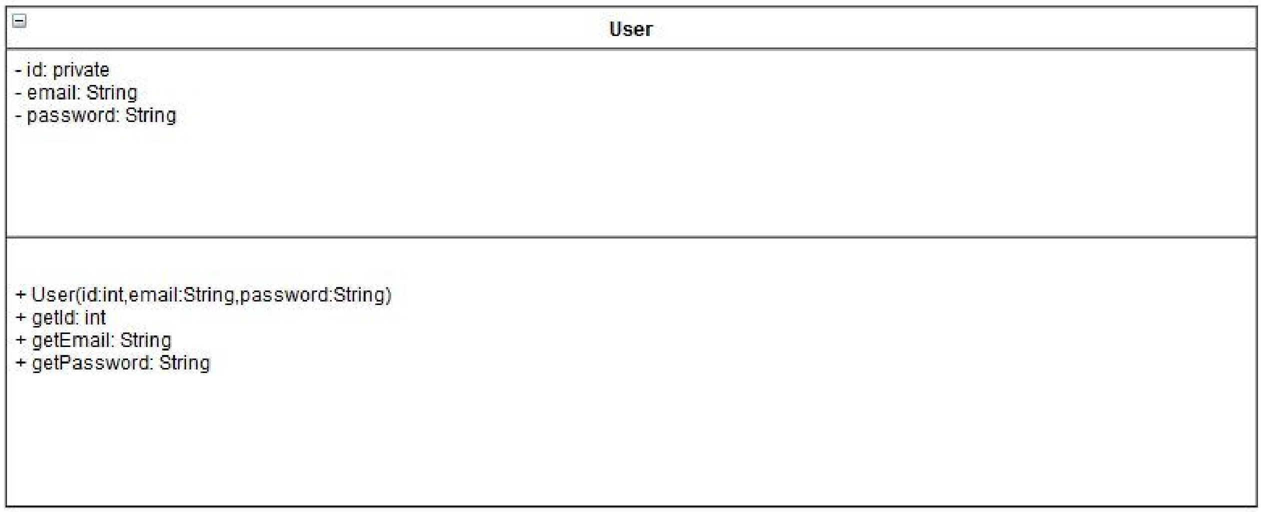
```
Playlist
- id: int
- name: String
- time: int
- numberOfSongs: int
- tracklist: List<Song>

+ Playlist(id:int,name:String)
+ addSong(song:Song): void
+ removeSong(song:Song): void
+ setName(name:String): void
+ getName: String
+ getId: int
+ getTracklist: List<Song>
+ setTracklist(songs:List<Song>): void
+ getTime: int
+ getNumberOfSongs: int
+ getTimeInString: String
+ getPositionOfSong(song:Song): int
+ isSongOnTracklist(song:Song):boolean
```

BE - Song

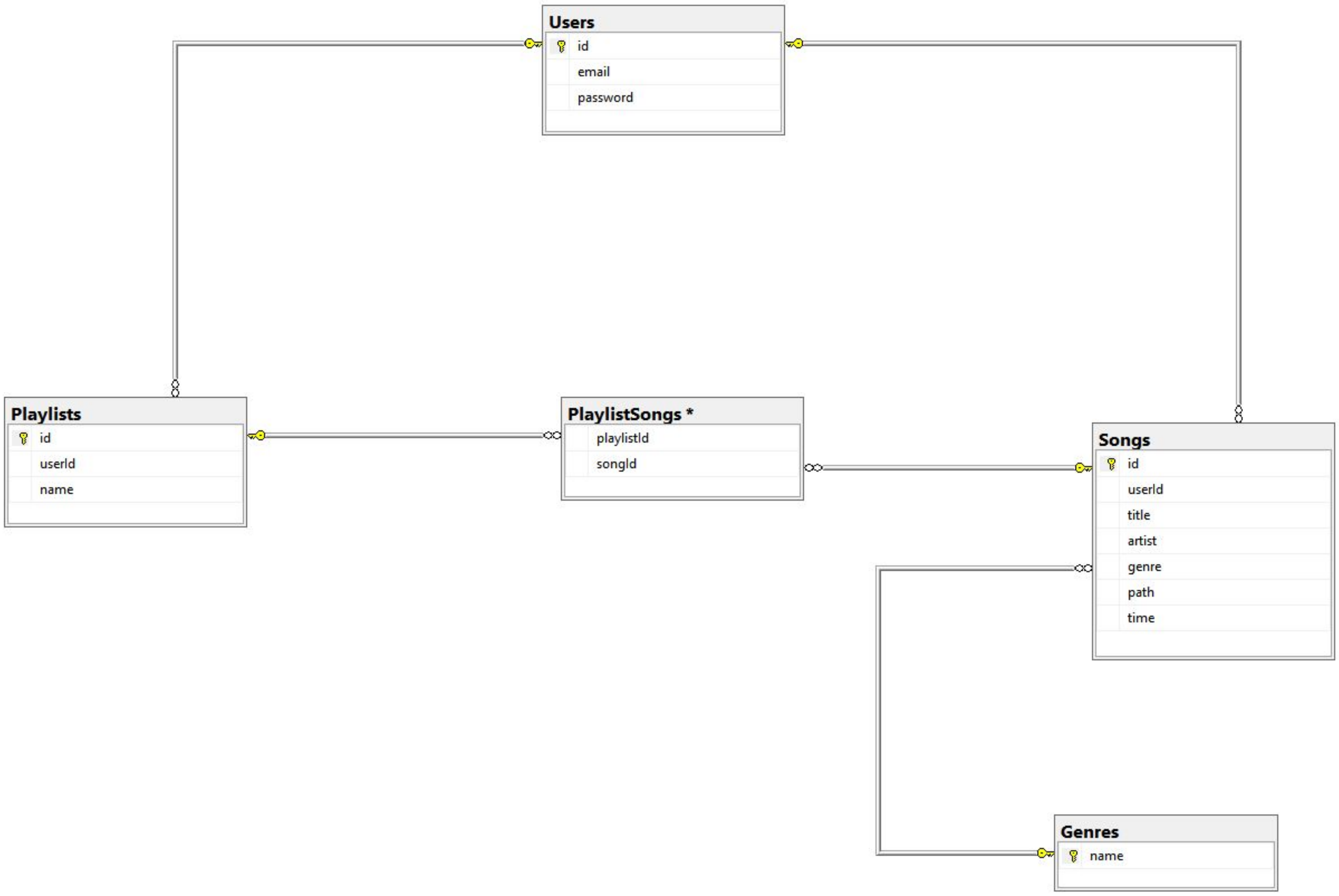


```
Song
- id: int
- title: String
- artist: String
- genre: String
- path: String
- time: int

+ Song(id:int,title:String,artist:String,genre:String,path:String,time:int): Song
+ getId: int
+ getTitle: String
+ getArtist: String
+ getGenre: String
+ getTime: int
+ getPath: String+ setTitle(title:String): void
+ setArtist(artist:String): void
+ setGenre(genre:String): void
+ setPath(path:String): void
+ getTimeInString: String
+ toString: String
```

BE - User


User
- id: private
- email: String
- password: String
+ User(id:int,email:String,password:String)
+ getId: int
+ getEmail: String
+ getPassword: String

Data storage



Implementation details

Since we made the decision to include optional functionalities, we faced a lot of issues that we weren't expecting, but even without the side features there were problems that we never thought of in the compulsory features as well.

- Time slider
  - Everytime the user moves the time slider, the label that displays the current song time changes dynamically.
  - It is directly connected with the time property of the media player so that the time player changes everytime we change the time slider.
  - The time of the song doesn't change until the user will drops the slider.
- Volume slider
  - Everytime the user moves the volume slider, the label that displays the volume changes dynamically.
  - It is connected with the mute button, so that when the user clicks the mute button, the volume slider will adjust accordingly and reach point zero.
- Windows effects
  - Displays effects when the user opens a new pop-up window, as well as when that window closes.
- Fading the volume
  - Instead of abruptly cutting the volume when the user stops and resumes the song, we are slightly fading the volume for a better effect.
- Css
  - We enhanced the visuals of the app with the inclusion of simple CSS.
- Shuffle
  - A shuffle button that allows the user to play songs randomly.
- Mute
  - A mute button allowing the user to quickly mute and unmute the song.
- Repeat
  - A repeat button giving the possibility for the user to repeat the current song.
- Disabling buttons
  - In every behavior of the controller we are handling the disabling of specific buttons to prevent unwanted behavior from the user.
- Our own window frame
  - We removed the basic window frame, because it didn't fit in our design, and there was no customization for the basic frame, and so we made our own.
  - Our custom frame is able to minimize close, as well as get dragged.
- Simple system for validating password and e-mail
  - A simple system for checking the e-mail that the user inputs, in case its already in use.
  - It also detects whether the password the user has written is not secure enough and prevents the creation of the account.

- PlayingMode enum
    - The use of playmode is so that we can securely know if the currently playing song is playing from the playlists or from the list of currently playing songs.
    - This enum was needed so that we could return the proper next and previous song depending on the source of playing.
- Util packages
    - Separated packages for the functional classes, from the gui and bll packages.
- Song Chooser class
    - It is used for returning the proper song based on the user’s request from the given source.
    - One of the difficult parts was returning the previous song while the shuffling was enabled, to solve this we used stack, which is storing the previously played random song.
- Logo and name
    - To separate our application from others and to complete our work we decided to create a logo and a unique name for our application.
    - At first the concept of the name was “bonebeat” but by accident we spelled it in the opposite as “beatbone” and realized that it had a punchier sound and reminded us of beatbox so we decided to keep it as Beatbone.

During the creation of this application we were highly inspired by the design and functionality of Spotify, and so we tried to imitate every single behavior, to be on point to that of Spotify.
The effects of this are visible in a lot of places in our application

- In the functionality of the shuffle button.
- In the functionality of the mute button.
- In the functionality of the time slider and volume slider.

And in a lot of smaller details.

Source control

In order to track our work and coordinate together we have used the Git control system, and have given it the name “MyTunes” to our repository.

Our team has used Git from the very start of the project, we decided to commit every change through one account, since we are not experienced enough in the use of Git, and it caused us problems when we tried to do it otherwise.

We sticked to the mentality of committing every small change and commenting it out, not only as a way to keep track of what we have done and have a timeline of the project,but also a good backup in case of problems and a way to look back at what we have achieved each day and understand it clearly.

Source code

Repo:https://github.com/schemabuoi/MyTunes