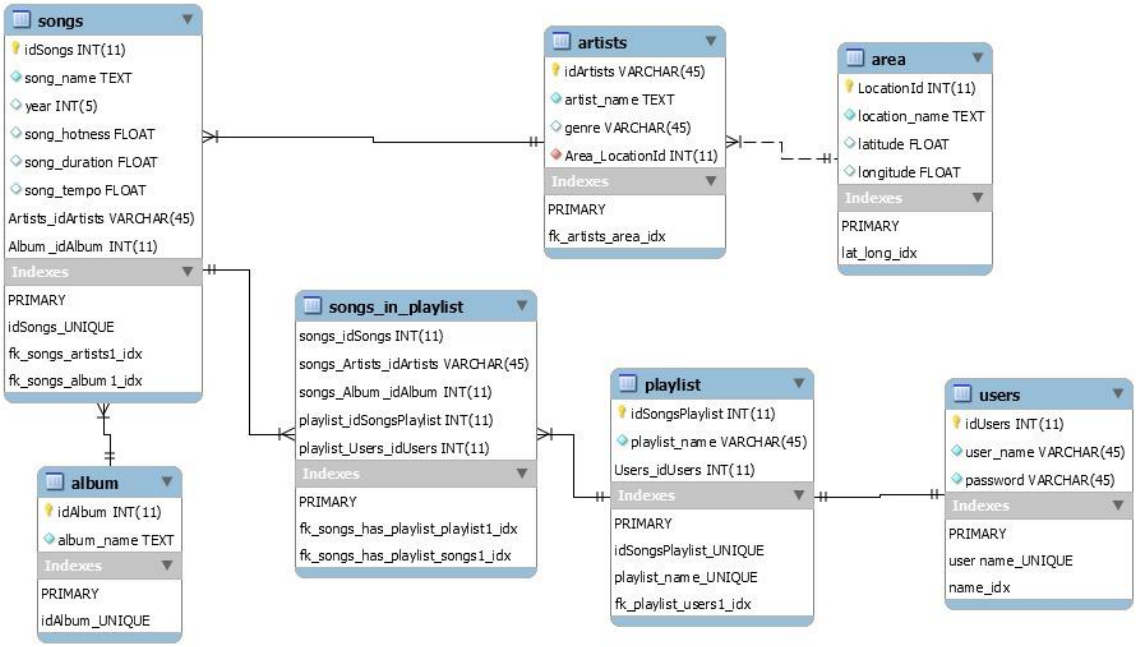
Softwate Documntaion



songs – include all songs and their properties. Album_id and Artists_idArtists (Artist_id) column are *foreign* key. idSongs is a primary key. Album_id and Artists_idArtists are also indexes.

Song name – the song name, type text.

Year – song creation year, type int(5).

Song_hotness – the song hotness as float from 0 to 1 , -1 if lack of info.

Song_duration – soung length by second, type float.

Song_tempo – song tempo, type float.

Area – include all area and their longitude and latitude values. LocationId is *Primary* key. Latitude and longitude are indexes.

Location_name – the location name , type text.

Latitude- the area latitude, type float.

Longitude- the area longitude, type float.

Album – include all album names, idAlbum is *Primary* key.

Album_name- the album name, type text.

Users – include all users and their password. idUsers *is Primary* Key. User_name is index.

User_name – type text.

Password – type text.

Playlist – a table that connect between users and their playlists. Users_idUsers(user_id) is *Foreign* key, idSongsPlaylist is *Primary* key. Playlist_user is an index.

Playlist_name- the playlist name as varchar, unique field.

Songs_in_playlist – table that contains all songs that belong to any user's playlist . song_id is *Foreign* key.

artists – table that contains all the artist of the song in the song table. Id_artsits is the primary key with type varchar. Areas_Location_Id is the index.

Artist_name – type text.

Genre- the artist genre, type varchar(45). (all songs from an artist will have his genre).

Areas_Location_Id – the artist area, type int.(all songs from an artist will have his arear)

SQL Queries

1) `SELECT * FROM Users WHERE user_name = '{Username}'`

A query that should verify if username is already exist (if the return table count > 0).

2) `INSERT INTO Users (user_name, password) VALUES ('{Username}', '{Password}')`

A query that insert the desired username and password of the user.

3) `SELECT idUsers FROM users WHERE user_name = '{Username}' AND password = '{Password}'`

A query that retrieve the user id that is getting from the table when inserted

4)
```
Select LocationId, location_name, count(location_name) FROM
  (SELECT area.LocationId,area.location_name FROM music_area_playlist.area
WHERE area.latitude != 0 AND area.longitude != 0
GROUP BY area.location_name
ORDER BY(6371 * acos( cos( radians(area.latitude) ) * cos( radians({area.Latitude}))
* cos( radians({Area.Longtitude} - radians(area.longitude) ) + sin(
radians(area.latitude) ) * sin(radians({Area.Latitude} Asc LIMIT 20) AS country
JOIN artists JOIN Songs WHERE songs.artists_idArtists = artists.idArtists AND
artists.Area_LocationId = country.LocationId
GROUP BY location_name order by count(location_name) desc LIMIT 10
```

A query that composed of sub-query.
First, the sub-query returns a 20 rows of area name and id where have details about latitude and longitude, and have the minimal destination from the point that the user choose (using a specific formula that will be explained later in the documentation).
Than the external query show also for each area that is retrieve from the sub query – the number of songs that this area include, and take the 10 "richest" areas.

5)
```
SELECT idSongs,song_name,year,song_hotness,song_duration,song_tempo,
artists_from_areas.idArtists,artists_from_areas.artist_name,
 artists_from_areas.genre,album_name,idAlbum
FROM songs JOIN (SELECT idArtists, artist_name, genre FROM artists WHERE
 Area_LocationId in (" + str + ") GROUP BY idArtists)
 AS artists_from_areas join album
WHERE songs.Artists_idArtists = artists_from_areas.idArtists and
songs.Album_idAlbum = album.idAlbum GROUP BY idSongs order by idSongs
```

Str- a dynamic string that represent a group of location_id that the user

choose (according to their name).
The query create a playlist.
The query return each song and his properties (genre etc. their retrieve from the join command) that his location_id is in the group of Str below.

6)
```
SELECT LocationId FROM music_area_playlist.area WHERE area.longitude = 0 AND
area.latitude = 0 order by rand() limit 6
```

A query that returns 6 areas_id that have longitude and latitude 0

7)
```
SELECT idSongs,song_name,year,song_hotness,song_duration,song_tempo,
 artists_from_areas.idArtists,artists_from_areas.artist_name
,artists_from_areas.genre,album_name,idAlbum
FROM songs JOIN (SELECT idArtists, artist_name, genre FROM artists WHERE
 Area_LocationId in " + CheckForRandomAreas() + " GROUP BY idArtists)
AS artists_from_areas
join album WHERE songs.Artists_idArtists = artists_from_areas.idArtists and
songs.Album_idAlbum = album.idAlbum GROUP BY idSongs order by idSongs
```

CheckForRandomAreas() – return a string represent the group of location_id that is retrieved from query NO. 6

A query that returns a song properties for building playlist in a random way.

8)
```
Insert into playlist(playlist_name, Users_idUsers)
values('{playlist.User.Name}',{playlist.User.ID})
```

A query that insert into the playlist table the name of playlist(username of user) and the id of their user.

9)
```
SELECT idSongsPlaylist from playlist where playlist_name =
'{playlist.User.Name}' AND Users_idUsers = {playlist.User.ID}
```

A query that retrieve from the table the playlist_id that the playlist got when inserted.

10)
```
Select idSongsPlaylist from playlist where Users_idUsers
={playlist.User.ID}
```

A query that check if the playlist is already exists (if the No. of rows affected is 0 -> mean that it doesn’t exist).

11) `select Songs_idSongs from songs_in_playlist where Playlist_idSongsPlaylist = {playlist.ID}`

A query that return a table contain all song_id of a given playlist.

12) `Delete FROM songs_in_playlist WHERE Songs_idSongs = {dataRow.Field<int>(0)} AND Playlist_idSongsPlaylist = {playlist.ID}`

A query that delete a single song from the songs_in_playlist table according to song_id and playlist_id given.

13) `Insert into songs_in_playlist (Songs_idSongs, Songs_Album_idAlbum, Songs_Artists_idArtists , Playlist_idSongsPlaylist, Playlist_Users_idUsers) values({song.ID}, {song.Album.ID}, {song.Artist.ID}, {playlist.ID}, {playlist.User.ID}`

A query that insert a single song to a playlist by a given song properties and playlist id.

Code Structure

Our project is composed of 6 packges (or Namespace) written in c# using WPF :

View – include all the xaml files and a c# file for each window in the application. The xaml include the design of window and the c# files include the the methods of buttons.

ViewModel **–**

Include the classes that are responsible of binding the model properties with the window.

BaseVM – holds the all classes of ViewModels. Via enum, create the appropriate viewmodel and also responsible of passing parameters between Viewmodels.

IVM **–** an interface that all viewModels implemets, include methods that are connected to passing parameters.

CountryChooserVM **–** launches all the methods of his appropriate model, and according to the result - > show the message (if should be) to the user, else pass to next viewModel (with BaseVM).

LocationMapChooserVM **-** launches all the methods of his appropriates model and draw a red circle where the user choose his spot.

LoginVM **–** launches all the methods of his appropriate model and pass the parameters into next window. If there is and error in process – show a message to user.

RegistrationVM **–** same as LoginVM.

PlayListVM **–** show the initial playlist and pass to the next window, according to the user's choice.

PlayListEditorVM **–** show the playlist of user. And when the user decied to edit this playlist (via Filter button), reset to initial (via reset button) or remove a single song (via remove button) -> bind it with the model**.**

DataBase **–**

Include the interface of the code with the DataBase

DataBaseConnection – responsible of the actual connection with the database. Holds the info (username, password etc.) that he get from the AppConfig. Also can connect and disconnect (close ()) to the database. This class is controlled by the DB_Executer class (later).

DB_Executer – responsible of execute all the queries and interact with database. Holds a DataBaseConnection that enable him to connect to database, and also convert the string query to actual MySql command, and execute the query with multiple choice – with a result (table) or without and a quick query without disconnect.

Model –

Include all the classes that responsible of the Calculation of application, building the logic of the queries to the databases, and holds the entities that eventually will be show on the screen.

LoginModel – check if the username&password that the user submitted is valid, and if it is – generate his own playlist.

RegistrationModel – register the user with the desired username and password that the user submitted.

LocationMapChooserModel - retrieve from user's spot choice the x,y values and convert it to Latitude and Longtitude. Than retrieve the 8 closest area's to that spot with the add of the longtitude and latitude info.

CountryChooserModel – get the desired areas that the user choose and generate a playlist that all of it's songs are from the those areas.

PlayListModel - his main function is to save the playlist that the user edited (or maybe not) not before to check if it should be save for the first time or just updated.

PlayListEditorModel – responsible of editing the user's playlist. Filter the playlist by the genre or just remove a single user, and also can reset to the initial playlist.

Entities **–**

Include all clases that represent a column in the data base (such as Song, Artist etc.) + helpers classes.

QueryInterpreter – this class is responsible of converting a DataTable result into an appropriate objects. According to the window (Via enum) – convert the dataTable rows into objects. In the end serialize the objects into Json String and return it.

Map – hold the information about the map pictures and do internal calculation.

ExtensionInfo – helper class that can be attached to an object and bind to the window component.

Data Source

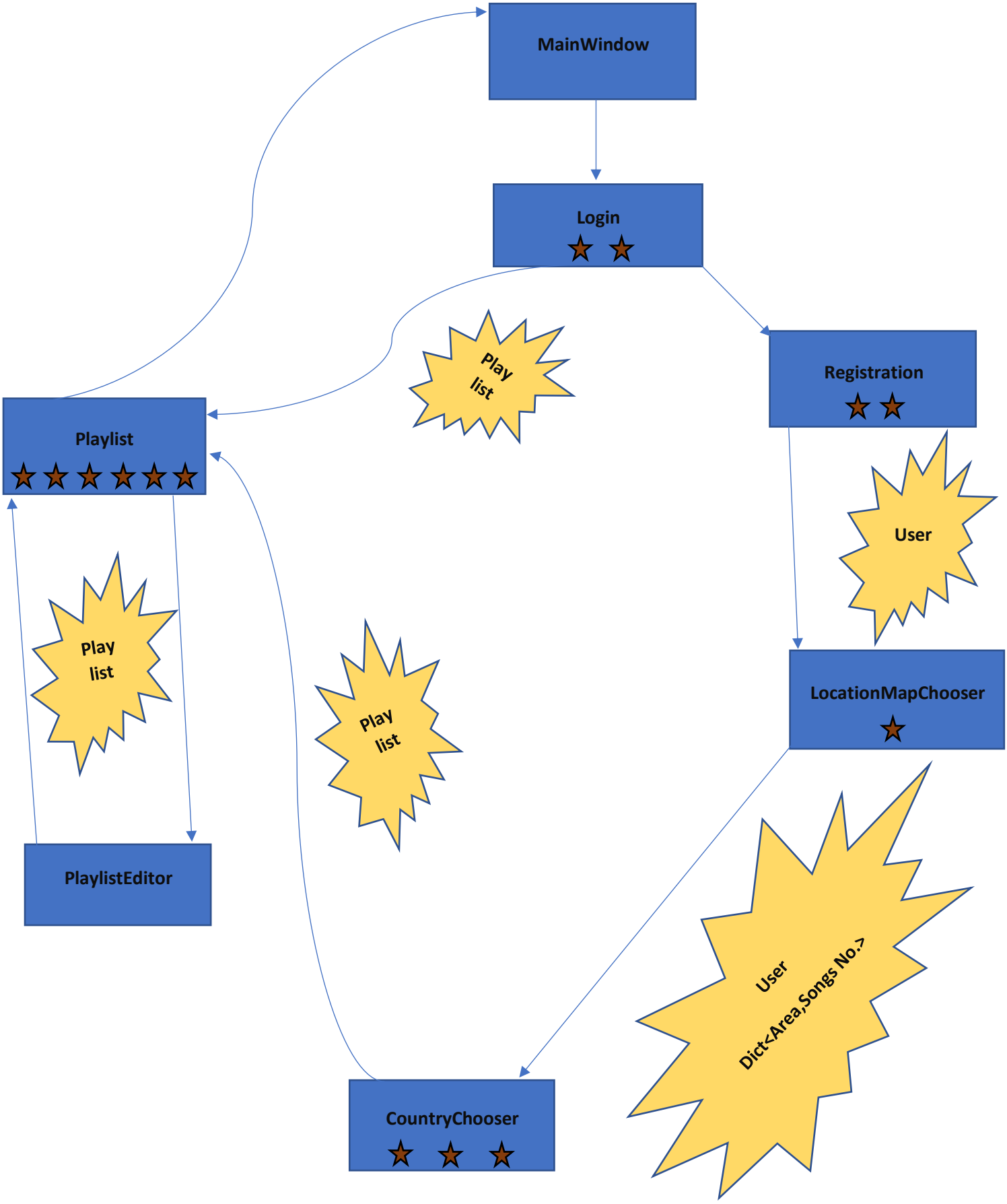
We use the Data Source *Million Songs Dataset*.

Although there is a 280 GB (but not free), we use the subset of data that the site suggest, and his size is 1.8 GB compressed. Then we extracted the data that was in h5 file into a single big CSV file with a python scripts. After that, we took the specific colmuns that we need, and create multiple CSV file that in each file is actually a future table. In the end we built the tables with MySql workbench.

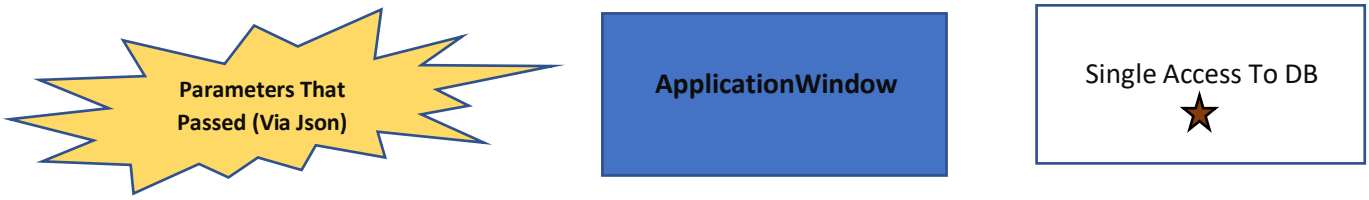
The colmuns that we used:

- Genre
- Hotness
- Year
- Title
- Duration
- ArtistName
- ArtistLongtitude
- ArtistLatitude
- AlbumName
- Tempo
- Location_name

General Flow of Application



Flow Dictionary



External Packages

MySql.Data – convert the string queries into real SqlCommands and execute it in database.

NewtonSoft.Json – we use Json for passing parameters between windows in application, using Jarray (composed of Jtokens).

Prism – for the binding between the xaml and the properties in code.

Algorithms

In our Application we used the *Haversine* Formula to calculate a distance in km between two spots in the map, by their latitude and longtitude values.

The Formula is:

Distance $= 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$

λ_1 = longtitude of point 1

λ_2 = longtitude of point 2

φ_2 = latitude of point 2

φ_1 = latitude of point 1

r = 6371 (in km), radius of earth

Nir Azaria 204796338

Natan Furer 204594428

Omer Yaso 307204247

Gal Eini 305216962