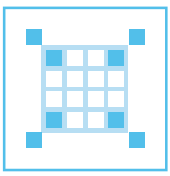

DEEC
DEPARTAMENTO DE ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

Base de Dados
2018/2019

Drop Music
Meta 2

Autoria:

Filipe João A. S. Conceição
2014196660
uc2014196660@student.uc.pt

João Carlos da C. Barreiros
2014196880
uc2014196880@student.uc.pt

Turma PL6

28 de Novembro de 2018

Conteúdo

1 Introdução

Este projeto[1] tem como objetivo criar um sistema de gestão e partilha de músicas, isto é, um guia de música com funcionalidades semelhantes aos arquivos *AllMusic.com* e *IMDb.com*, acrescentando a possibilidade de partilhar ficheiros entre utilizadores individuais do sistema, tal como o serviço *Dropbox*.

O sistema desenvolvido tem diversas informações relevantes sobre as músicas, tais como os autores associados, que podem ser grupos musicais (bandas) ou músicos individuais e, consequentemente os concertos realizados por estes. O sistema inclui, também, informações sobre os álbuns associados aos autores, os quais possuem as respetivas músicas bem como os compositores e letras das músicas.

Um utilizador pode pesquisar músicas não só pelo seu nome, respetivo autor, compositor ou álbum, mas também pelo respetivo género, data de lançamento, pontuação ou letra. Este ao consultar um determinado álbum, obtém informação sobre a duração deste, bem como a listagem de músicas, ou seja, o número total de músicas pertencentes a esse álbum, cada uma com as respetivas informações, durações, críticas escritas por outros utilizadores, bem como informações gerais sobre os géneros musicais, a data de lançamento desta e a sua letra.

Um utilizador para utilizar o sistema deve registar-se com um nome de utilizador e palavra-passe e *email*, podendo acrescentar outro tipo de informação pessoal, tal como o seu nome, país ou género. Existem dois tipos de utilizadores:

- Utilizadores normais que são aqueles que apenas podem consultar as informações das músicas, álbuns, autores, etc.
- Utilizadores com privilégios, isto é, editores que podem acrescentar e alterar informações no sistema, tal como informações sobre álbuns, músicas e autores.

É possível a cada utilizador criar *playlists* das suas músicas favoritas, podendo estas serem privadas ou públicas (ficando disponíveis para todos os utilizadores do sistema), bem como transferir ficheiros musicais (MP3, FLAC, etc.) que ficarão associados à sua própria conta. Igualmente, podem ser partilhados com outros utilizadores através do sistema. Para além disso, qualquer utilizador (editor ou não) pode escrever uma crítica a um música de um álbum, que consiste numa pontuação e numa justificação textual.

Para a realização dos diagramas foi utilizada a ferramenta ONDA[2].

2 Diagrama conceptual

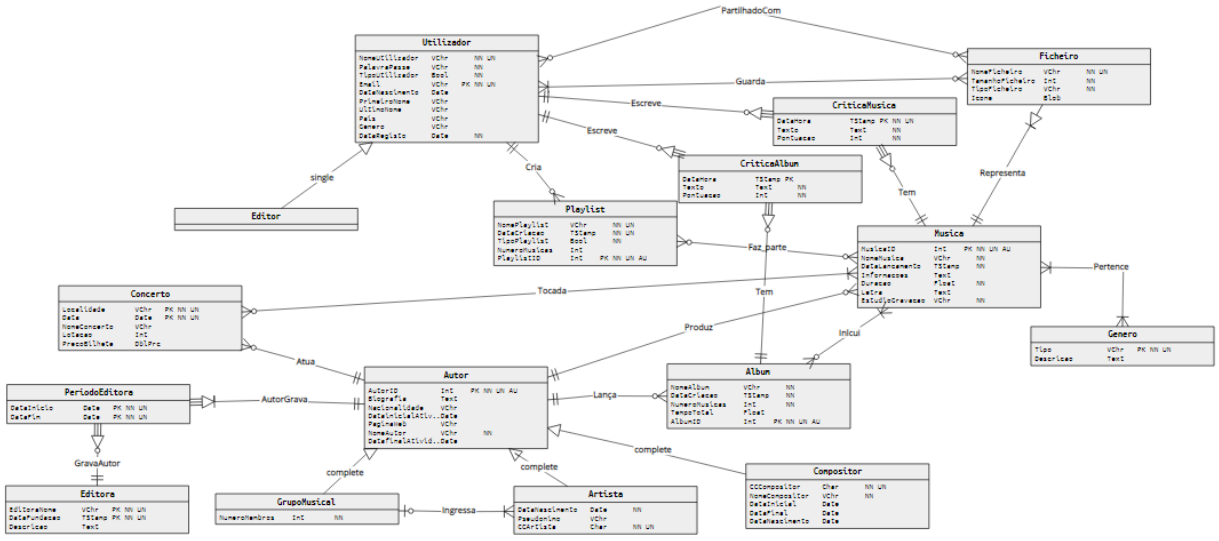

Figura 1: Diagrama conceptual

3 Descrição das entidades, atributos e regras de integridade

Para a criação do diagrama conceptual ou entidade-relacionamento (ER), introduzimos as seguintes entidades:

- **Música** que representa um conjunto de músicas na base de dados, possuindo informações sobre:
  1. MusicaID - atributo do tipo inteiro identificador da música, utilizado como **chave primária**, devendo, consequentemente, ser único, obrigatório (não nulo) e incrementado automaticamente.
  2. NomeMusica - atributo do tipo *VarChar* que representa o nome da música, devendo ser obrigatório. É verificado se a primeira letra do nome da música está em maiúscula.
  3. DataLançamento - atributo do tipo *TimeStamp* que representa a data de lançamento da música, devendo ser obrigatório. É verificado se a data de lançamento da música não é futura à data presente.
  4. Informações - atributo do tipo *Text* que representa informações variadas acerca da música.
  5. Duração - atributo do tipo *Float* que representa a extensão temporal da música, devendo ser obrigatório. É verificado se a duração da música é maior que 0.
  6. Letra - atributo to tipo *Text* que representa a letra da música, caso esta o tenha.
  7. EstudioGravação - atributo do tipo *VarChar* que representa o estúdio onde a música foi gravada, devendo ser obrigatório. É verificado se a primeira letra do nome do estúdio de gravação está em maiúscula.
- **Autor** que representa um conjunto de autores (que podem ser artistas solo ou grupos musicais) na base de dados. Esta entidade apresenta uma hierarquia, sendo uma super-entidade das entidades **Grupo Musical** e **Artista** e possuindo informações sobre:

1. AutorID - atributo do tipo inteiro identificador do autor, utilizado como **chave primária**, devendo, consequentemente, ser único, obrigatório (não nulo) e incrementado automaticamente.
2. Biografia - atributo do tipo *Text* que representa a biografia do autor.
3. Nacionalidade - atributo do tipo *Varchar* que representa a nacionalidade do autor. Caso não seja nulo, é verificado se a primeira letra da nacionalidade do autor está em maiúscula.
4. DataInícioAtividade - atributo do tipo *Date* que representa a data de início da atividade do autor. Caso não seja nulo, é verificado se a data de início de atividade do autor não é futuro à data presente.
5. DataFinalAtividade - atributo do tipo *Date* que representa a data de fim da atividade do autor.
6. PáginaWeb - atributo do tipo *VarChar* que representa a página *web* de um determinado autor.
7. NomeAutor - atributo do tipo *VarChar* que representa o nome do autor, devendo ser obrigatório. È verificado se a primeira letra do nome do autor está em maiúscula.

- **Utilizador** que representa um conjunto de utilizadores na base de dados que fazem uso do sistema de gestão e partilha de músicas desenvolvido. Esta entidade apresenta uma hierarquia, sendo uma super-entidade da entidade **Editor** e possuindo informações sobre:
  1. NomeUtilizador - atributo do tipo *VarChar* que representa o nome escolhido pelo utilizador no momento do registo deste, devendo ser obrigatório e único.
  2. PalavraPasse - atributo do tipo *VarChar* que representa a palavra-chave escolhida pelo utilizador no momento do registo deste, devendo ser obrigatório.
  3. TipoUtilizador - atributo do tipo *Bool* que representa o tipo de utilizador no sistema e que tem como objetivo permitir a distinção entre utilizadores normais do sistema e os editores deste, devendo ser obrigatório. Por *default* o seu valor é 0/false (significando que, inicialmente, o utilizador a registar-se não é editor).
  4. Email - atributo do tipo *VarChar* que representa o email eletrónico escolhido pelo utilizador no momento do registo deste, utilizado como **chave primária**, devendo, consequentemente, ser único e obrigatório (não nulo).
  5. DataNascimento - atributo do tipo *Date* que representa a data de nascimento do utilizador. Caso não seja nula, é verificado se a data de nascimento do utilizador não é futuro à data presente.
  6. PrimeiroNome - atributo do tipo *VarChar* que representa o primeiro nome do utilizador. Caso não seja nulo, é verificado se o primeiro nome do utilizador está em letras maiúsculas.
  7. ÚltimoNome - atributo do tipo *VarChar* que representa o último nome do utilizador. Caso não seja nulo, é verificado se o último nome do utilizador está em letras maiúsculas.
  8. País - atributo do tipo *VarChar* que representa o país de origem do utilizador. Caso não seja nulo, é verificado se a primeira letra do país de origem do utilizador está em maiúscula.
  9. Género - atributo do tipo *VarChar* que representa o género do utilizador.
  10. DataRegisto - atributo do tipo *Date* que representa a data do registo do utilizador no sistema, devendo ser obrigatório. Por *default* o seu valor corresponde à data atual do servidor no momento do registo do utilizador.

- **Editor** que representa um conjunto de editores ou utilizadores com privilégios na base de dados, sendo sub-entidade da entidade **Utilizador**
- **Álbum** que representa um conjunto de álbuns na base de dados, possuindo informações sobre:
    1. AlbumID - atributo do tipo inteiro identificador do álbum, utilizado como **chave primária**, devendo, consequentemente, ser único, obrigatório (não nulo) e incrementado automaticamente.
    2. NomeÁlbum - atributo do tipo *VarChar* que representa o nome do álbum, devendo ser obrigatório (não nulo). É verificado se a primeira letra do nome do álbum está em maiúscula.
    3. DataCriação - atributo do tipo *TimeStamp* que representa a data de criação do álbum, devendo ser obrigatório (não nulo). É verificado se a data de criação do álbum não é futura à data presente.
    4. NúmeroMúsicas - atributo do tipo inteiro que representa o número total de músicas que pertencem a um determinado álbum, devendo ser obrigatório. É verificado se o número total de músicas de um álbum é maior que 0.
    5. TempoTotal - atributo do tipo *Float* que representa a extensão temporal de um álbum, resultado da duração de todas as suas músicas. Caso seja não nulo, é verificado se a duração do álbum é maior que 0.
- **Ficheiro** que representa um conjunto de ficheiros de áudio na base de dados. Esta entidade é uma entidade fraca pois não têm, naturalmente, atributos que possam constituir chaves candidatas de forma a que a **chave primária** desta entidade depende inteiramente da **chave primária** da entidade identificadora **Música**. Esta entidade possui informações sobre:
    1. NomeFicheiro - atributo do tipo *VarChar* que representa o nome do ficheiro, devendo ser obrigatório e único. É verificado se a primeira letra do nome do ficheiro está em maiúscula.
    2. TamanhoFicheiro - atributo do tipo inteiro que representa o tamanho, em *bytes*, do ficheiro, devendo ser obrigatório. É verificado se o tamanho do ficheiro é maior ou igual a 0.
    3. TipoFicheiro - atributo do tipo *VarChar* que representa o tipo do ficheiro (MP3, FLAC, etc.), devendo ser obrigatório. É verificado se a primeira letra do tipo do ficheiro está em maiúscula.
    4. Ícone - atributo do tipo *Blob* que representa o ícone do ficheiro.
- **PlayList** que representa um conjunto de *playlists* associadas a um utilizador na base de dados, possuindo informações sobre:
    1. PlaylistID - atributo do tipo inteiro identificador da *playlist*, utilizado como **chave primária**, devendo, consequentemente, ser único, obrigatório (não nulo) e incrementado automaticamente.
    2. NomePlaylist - atributo do tipo *VarChar* que representa o nome da *playlist*, devendo ser único e obrigatório (não nulo). É verificado se a primeira letra do nome do *playlist* está em maiúscula.
    3. DataCriação - atributo do tipo *TimeStamp* que representa a data de criação da *playlist* devendo ser único e obrigatório (não nulo). É verificado se a data de criação da *playlist* não é futura à data presente. Por *default* o seu valor corresponde à data atual do servidor no momento da criação da *playlist*, por parte de um determinado utilizador.

4. TipoPlaylist - atributo do tipo *Bool* que representa o tipo da *playlist*, devendo ser obrigatório, tendo como objetivo permitir a distinção entre as *playlists* públicas e privadas do utilizador. Por *default* o seu valor é 0/false (significando que, inicialmente, a *playlist* criada pelo utilizador, é privada).
5. NumeroMusicas - atributo do tipo inteiro que representa o número total de músicas incluídas na *playlist*. Caso não seja nulo, é verificado se o número total de músicas na *playlist* é maior ou igual a 0.

- **CríticaMúsica** que representa um conjunto de críticas escritas por um utilizador (editor ou não) a uma música na base de dados. Esta entidade é uma entidade fraca pois não têm, naturalmente, atributos que possam constituir chaves candidatas de forma a que a **chave primária** desta entidade depende, portanto, das **chaves primárias** das entidades identificadoras **Utilizador** e **Música**. Esta entidade possui informações sobre:
  1. DataHora - atributo do tipo *TimeStamp* que representa a data e a hora de escrita da crítica relativa a uma música, por parte de um utilizador. Este atributo está incluído na **chave parcial**, devendo, consequentemente, ser único e obrigatório (não nulo). É verificado se a data e hora de escrita da crítica não é futura à data e hora presentes.
  2. Texto - atributo do tipo *Text* que representa a justificação textual da crítica de um utilizador a uma música, devendo ser obrigatório.
  3. Pontuação - atributo do tipo inteiro que representa a pontuação atribuída a uma música, por parte de um utilizador, devendo ser obrigatório. É verificado se a pontuação atribuída a música, por parte de um utilizador, é maior ou igual a 0 e menor ou igual a 100.
- **CríticaÁlbum** que representa um conjunto de críticas escritas por um utilizador (editor ou não) a um álbum na base de dados. Esta entidade é uma entidade fraca pois não têm, naturalmente, atributos que possam constituir chaves candidatas de forma a que a **chave primária** desta entidade depende, portanto, das **chaves primárias** das entidades identificadoras **Utilizador** e **Album**. Esta entidade possui informações sobre:
  1. DataHora - atributo do tipo *TimeStamp* que representa a data e a hora de escrita da crítica relativa a uma álbum, por parte de um utilizador. Este atributo está incluído na **chave parcial**, devendo, consequentemente, ser único e obrigatório (não nulo). É verificado se a data e hora de escrita da crítica não é futura à data e hora presentes.
  2. Texto - atributo do tipo *Text* que representa a justificação textual da crítica de um utilizador a um álbum, devendo ser obrigatório.
  3. Pontuação - atributo do tipo inteiro que representa a pontuação atribuída a um álbum, por parte de um utilizador, devendo ser obrigatório. É verificado se a pontuação atribuída a música, por parte de um utilizador, é maior ou igual a 0 e menor ou igual a 100.
- **Grupo Musical** que representa um conjunto de grupos musicais na base de dados, sendo sub-entidade da entidade **Autor** e possuindo informações sobre:
  1. NúmeroMembros - atributo do tipo inteiro que representa a o número total de membros de um grupo musical, devendo ser obrigatório. É verificado se o número total de membros do grupo musical é maior que 0.
- **Artista** que representa um conjunto de artistas na base de dados, sendo sub-entidade da entidade **Autor** e possuindo informações sobre:
  1. CCArtista - atributo do tipo inteiro identificador do artista, devendo ser único e obrigatório (não nulo).

2. DataNascimento - atributo do tipo *Date* que representa a data de nascimento do artista, devendo ser obrigatório. É verificado se a data de nascimento do artista não é futura à data presente.
3. Pseudónimo - atributo do tipo *VarChar* que representa o pseudónimo do artista. Caso seja não nulo, é verificado se a primeira letra do pseudónimo do artista está em maiúscula.

- **Compositor** que representa um conjunto de compositores na base de dados, sendo subentidade da entidade **Autor** possuindo informações sobre:
  1. CCCompositor - atributo do tipo *Char* identificador do compositor, devendo ser único, obrigatório (não nulo).
  2. NomeCompositor - atributo do tipo *VarChar* que representa o nome do compositor, devendo ser obrigatório. É verificado se a primeira letra do nome do compositor está em maiúscula.
  3. DataNascimento - atributo do tipo *Date* que representa a data de nascimento do compositor. Caso não seja nulo, é verificado se a data de nascimento do artista não é futura à data presente.
  4. DataInicial - atributo do tipo *Date* que representa a data de início da atividade do compositor. Caso não seja nulo, é verificado se a data de início de atividade do autor não é futuro à data presente.
  5. DataFinal - atributo do tipo *Date* que representa a data de fim da atividade do compositor.
- **Concerto** que representa um conjunto de concertos na base de dados, possuindo informações sobre:
  1. Localidade - atributo do tipo *VarChar* que representa o local do concerto. Este atributo está incluído na **chave primária**, devendo, consequentemente, ser único e obrigatório(não nulo). É verificado se a primeira letra da localidade do concerto está em maiúscula.
  2. Data - atributo do tipo *Date* que representa a data do concerto. Este atributo está incluído na **chave primária**, devendo, consequentemente, ser único e obrigatório(não nulo). É verificado se a data do concerto não é futura à data presente.
  3. NomeConcerto - atributo do tipo *VarChar* que representa o nome do concerto. Caso não seja nulo, é verificado se a primeira letra do nome do concerto está em maiúscula.
  4. Lotação - atributo do tipo inteiro que representa a lotação do concerto. Caso não seja nulo, é verificado se a lotação é maior que 0.
  5. PrecoBilhete - atributo do tipo *Double* que representa o preço de um bilhete normal do concerto. Caso não seja nulo, é verificado se o preço do bilhete é maior ou igual a 0.
- **Editora** que representa um conjunto de editoras na base de dados, possuindo informações sobre:
  1. EditoraNome - atributo do tipo *VarChar* que representa o nome da editora. Este atributo está incluído na **chave primária**, devendo, consequentemente, ser único e obrigatório(não nulo).
  2. DataFundação - atributo do tipo *TimeStamp* que representa a data de fundação da editora. Este atributo está incluído na **chave primária**, devendo, consequentemente, ser único e obrigatório(não nulo). É verificado se a data de fundação da editora não é futura à data presente.

3. Descrição - atributo do tipo *Text* que representa a informações variadas de uma determinada editora.

- **PeríodoEditora** que representa um conjunto de períodos em que um autor esteve associado a uma editora na base de dados. Esta entidade é uma entidade fraca pois não têm, naturalmente, atributos que possam constituir chaves candidatas de forma a que a **chave primária** desta entidade depende, portanto, das **chaves primárias** das entidades identificadoras **Editora** e **Autor**. Esta entidade possui informações sobre:
  1. DataInício - atributo do tipo *Date* que representa o a data inicial em que um autor se vinculou a uma editora. Este atributo está incluído na **chave parcial**, devendo, consequentemente, ser único e obrigatório (não nulo). É verificado se a data inicial de vinculação do autor a uma editora não é futuro à data presente.
  2. DataFim - atributo do tipo *Date* que representa o a data final em que um autor se desvinculou de uma editora. Este atributo está incluído na **chave parcial**, devendo, consequentemente, ser único e obrigatório (não nulo).
- **Género** que representa um conjunto de géneros musicais na base de dados, possuindo informações sobre:
  1. Tipo - atributo do tipo *VarChar* que representa o nome/tipo do género musical, utilizado como **chave primária** devendo, consequentemente, ser único e obrigatório (não nulo).
  2. Descrição - atributo do tipo *Text* que descreve o género musical.

4 Modelo de dados relacional

O modelo de dados relacional ou diagrama físico, a seguir apresentado, foi gerado pela ferramenta indicada.

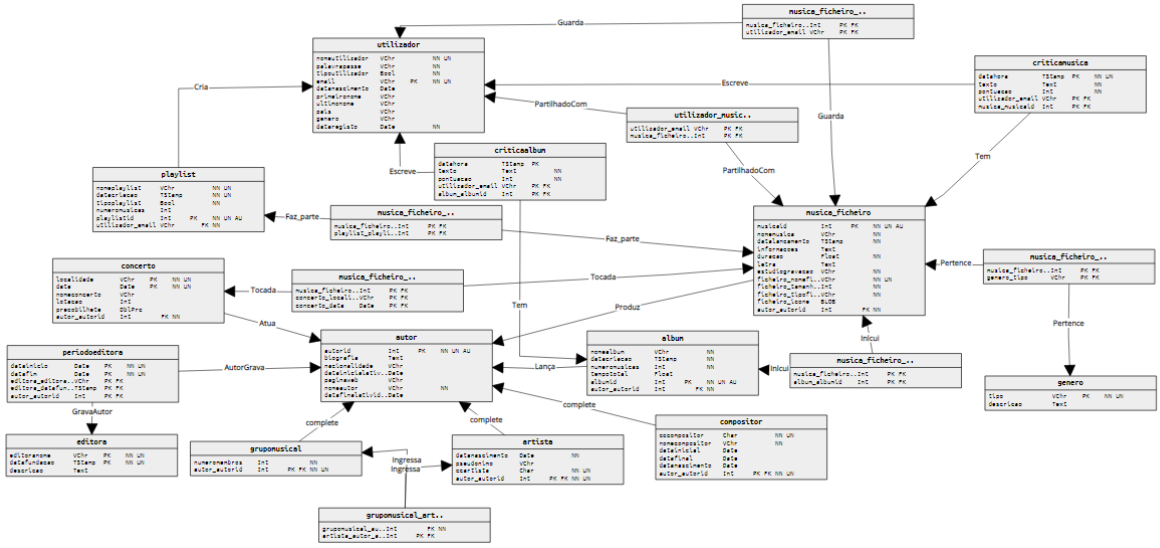

Figura 2: Diagrama Físico

Neste diagrama verifica-se a criação das tabelas que irão constar na base de dados, bem como as atribuições das chaves estrangeiras entre tabelas de forma a manter a integridade dos dados. Este tipo de chaves tem como objetivo garantir que os atributos associados a essas chaves estrangeiras (*Foreign Key* ou FK) numa tabela se encontra na tabela de referência como chave primária (*Primary Key* ou PK).

Durante a criação deste diagrama verificou-se as regras de tradução do diagrama ER no diagrama físico correspondente nas quais se destacam:

1. Relacionamento binário entre entidade de grau 1:1 e participação obrigatória: é necessária apenas uma tabela, com a chave primária igual a uma das chaves primária das entidades correspondentes.

2. Relacionamento binária entre entidades de grau 1:1 e participação não obrigatória de apenas uma das entidades: são necessárias duas tabelas, uma para cada entidade, em que a chave primária de cada entidade serve de chave primária da tabela correspondente e a chave primária da entidade com participação não obrigatória tem de ser usada como chave estrangeira na tabela correspondente à entidade cuja participação é obrigatória.

3. Relacionamento binária entre entidades de grau 1:1 e participação não obrigatória de ambas as entidades: são necessárias três tabelas, uma para cada entidade e a terceira para o relacionamento, em que a chave primária de cada entidade serve como chave primária de chave primária da tabela correspondente, a tabela do relacionamento terá como atributos (chaves estrangeiras) as chaves primárias das duas entidades e qualquer dos atributos pode ser a chave primária da tabela.

4. Relacionamento binário entre entidades de grau 1:N e participação obrigatória do lado N: são necessárias duas tabelas, uma para cada entidade em que a chave primária de cada entidade serve de chave primária da tabela correspondente e a chave primária da entidade do lado 1 tem de ser usada como chave estrangeira na tabela correspondente à entidade do lado N.

5. Relacionamento binário entre entidades de grau 1:N e participação não obrigatória do lado N: são necessárias três tabelas, uma para cada entidade e uma terceira para o relacionamento, em que a chave primária de cada entidade serve de chave primária da tabela correspondente e a tabela relativa ao relacionamento terá de ter entre os seus atributos as chaves primárias de cada entidade como chaves estrangeiras e como chave primária a chave primária da entidade do lado N.

6. Relacionamento binária entre entidades de grau M:N: são necessárias três tabelas, uma para cada entidade e uma terceira para o relacionamento, em que a chave primária de cada entidade serve de chave primária na tabelas correspondente e a tabela relativa ao relacionamento terá de ter entre os seus atributos as chaves primárias de cada uma das entidades (como chaves estrangeiras, estabelecendo a ligação com cada uma das tabelas correspondentes) e como chave primária a composição das chaves primárias das entidades que se relacionam.

7. Nos relacionamentos que envolvem entidades fracas é criada uma tabela para cada entidade fraca (EF) que inclui todos os atributos da EF, contendo como chave estrangeira o(s) atributo(s) que é(são) chave primária da entidade identificadora e como chave primária a combinação da chave parcial da EF com o(s) atributo(s) que é(são) chave primária da entidade identificadora.

8. Os relacionamentos que envolvem heranças/hierarquias e de acordo com a nomenclatura da ferramenta utilizada podem ser:

    - **Complete** em que são criadas uma tabela para a super-entidade com todos os atributos específicos da super-entidade, outra tabela para cada sub-entidade com os atributos específicos de cada sub-entidade e onde a chave primária da super-entidade aparece nas tabelas das sub-entidades, onde é simultaneamente chave estrangeira e chave primária. Esta foi a abordagem utilizada na relação entre as entidades Autor, Grupo Musical, Artista e Compositor.

- **Concrete** em que é criada uma tabela para cada sub-entidade e não é criada nenhuma tabela para a super-entidade. Neste caso, cada tabela contem todos os atributos da super-entidade e os atributos específicos da sub-entidade, sendo que a chave primária de cada tabela é a chave primária da sub-entidade correspondente.
- **Single** em que é criada uma única tabela para a super-entidade e para todas as sub-entidades, em que a chave primária desta tabela é a chave primária da super-entidade. Esta foi a abordagem utilizada entre as entidades Utilizador e Editor.

5 Diagramas Caso de Uso

Após a identificação dos requisitos funcionais, construímos os diagramas UML.

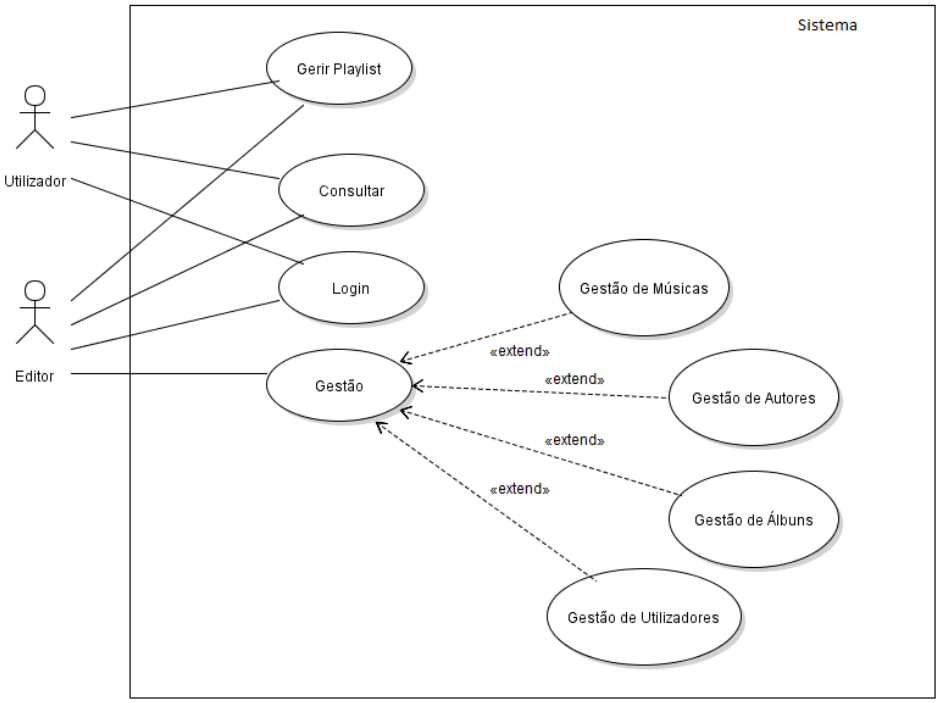

Figura 3: Diagrama UML de casos de uso geral (kite level)

Um diagrama de casos de uso UML descreve funcionalidades iniciadas pelo(s) actor(es) em termos de interações entre o sistema e o meio onde este está localizado. Os limites do sistema são representados como uma caixa, e fora desta estão os actores e sistemas externos. No diagrama da figura 3, observamos uma representação das funcionalidades do sistema no nível de abstração mais elevado.

No caso de uso consultar, o utilizador e editor, os dois atores presentes, quererão ver as músicas, os álbuns e os autores que existem na base de dados, bem como todos os dados relativos a cada um deles.

No caso de uso gerir *playlist*, o utilizador (editor ou não) poderá querer criar uma nova *playlist*, associando-lhe diferentes músicas, modificar uma *playlist*, alterando-lhe diversas informações ou modificando as músicas associados a esta, ou remover por completo uma *playlist* existente.

No caso de uso *login*, o utilizador ou o editor poderão querer iniciar sessão na aplicação, caso tenham uma conta associada. A este caso de uso, está associada a ação de registo, que terá de ser realizada previamente ao início de sessão, caso o utilizador pretenda utilizar a aplicação e não tenha conta associada.

No caso de uso gestão, o editor poderá querer gerir as músicas, os autores, os álbuns, os utilizadores, entre outras entidades, existentes na base de dados.

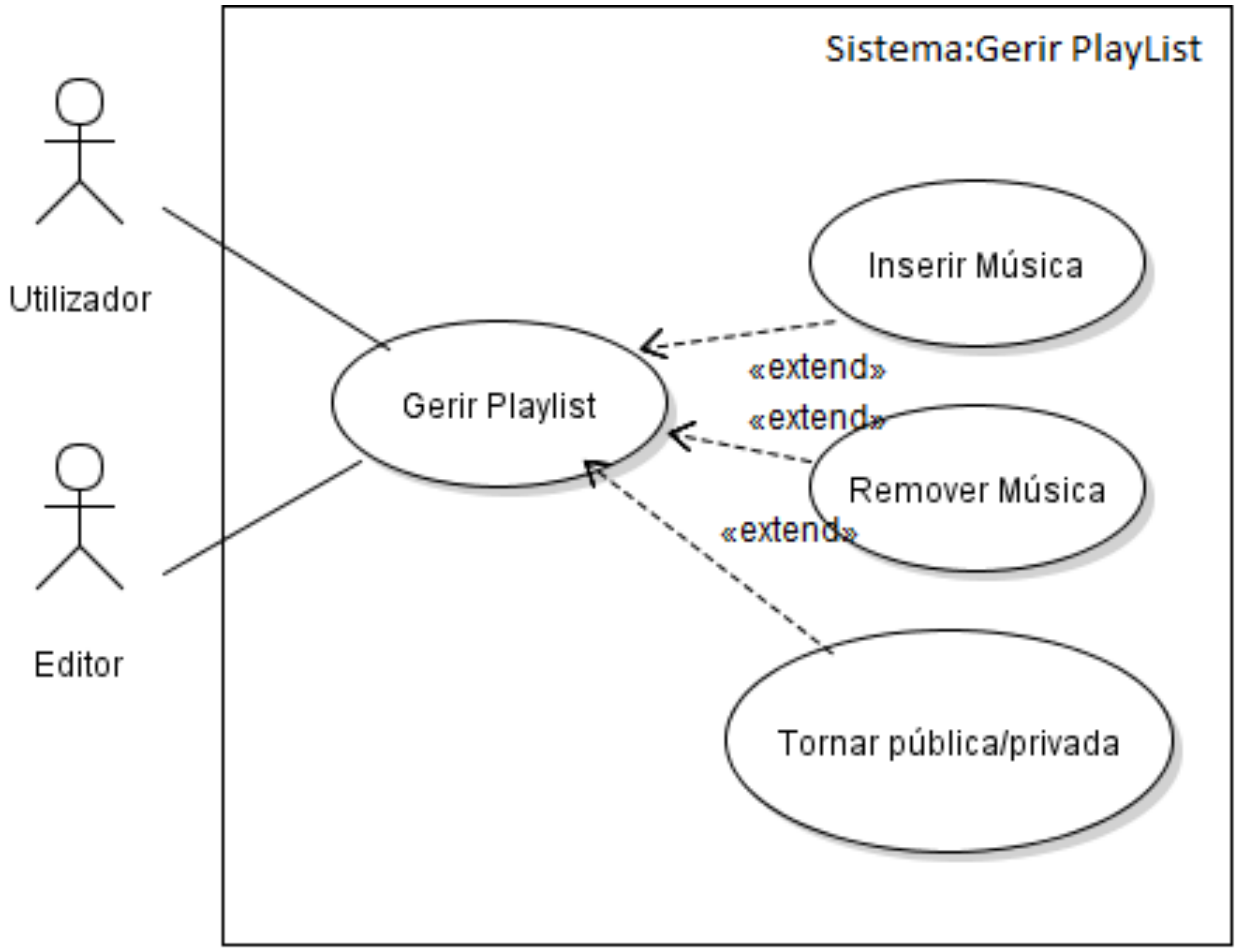

Figura 4: Diagrama UML de casos de uso PlayList (sea level)

No diagrama da figura 4, observamos as ações relacionadas com a gestão de *playlists* que um utilizador (editor ou não) poderá realizar.

No caso de uso inserir músicas, o utilizador poderá adicionar um determinado número de músicas, existentes na base de dados, a uma nova *playlist* criada por ele.

O utilizador poderá, também, querer modificar uma *playlist* já existente.

O utilizador poderá, também, querer remover uma *playlist* já existente.

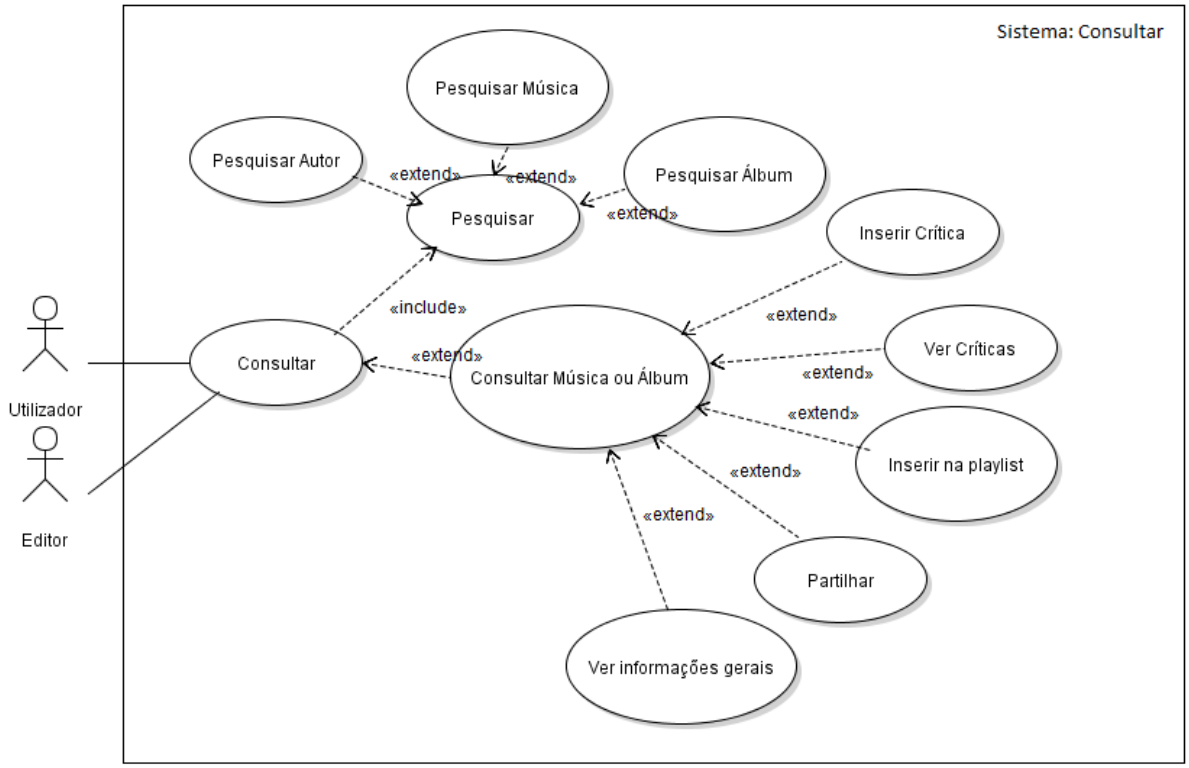

Figura 5: Diagrama UML de casos de uso Consulta (sea level)

No diagrama da figura 5, observamos as ações relacionadas com a consulta e pesquisa de músicas, autores ou álbuns e em particular, as ações que derivam da consulta de uma música ou álbum por parte de um utilizador (editor ou não).

No caso de uso consultar, o utilizador ou editor, poderá querer consultar uma música, autor ou álbum existentes na base de dados. Este caso de uso implica uma pesquisa prévia da entidade que se pretende consultar.

No caso de uso particular de consultar uma música ou um álbum e após esta ação, o utilizador ou editor, poderá querer inserir uma crítica na música ou álbum consultado(a). A inserção de uma crítica à música ou ao álbum corresponde a adicionar uma justificação textual e uma pontuação (de 0 a 100).

Desta forma, o utilizador ou editor, poderá também querer apenas ver as críticas que já foram inseridas, por outros utilizadores, na música ou álbum consultado(a).

No caso de uso particular de consultar uma música, o utilizador ou editor, poderá optar por inserir a música consultada a uma *playlist* já existente na base de dados. Para tal, este necessita de fornecer o identificador (ID) da respetiva *playlist*.

No caso de uso partilhar, o utilizador ou editor, poderá querer partilhar um determinado número de músicas existentes na base de dados com um outro utilizador do sistema que possui uma conta associada. Para tal, este necessita de fornecer o número de músicas e o seu respetivo identificador (ID), bem como o nome do utilizador com quem pretende partilhar as respetivas músicas.

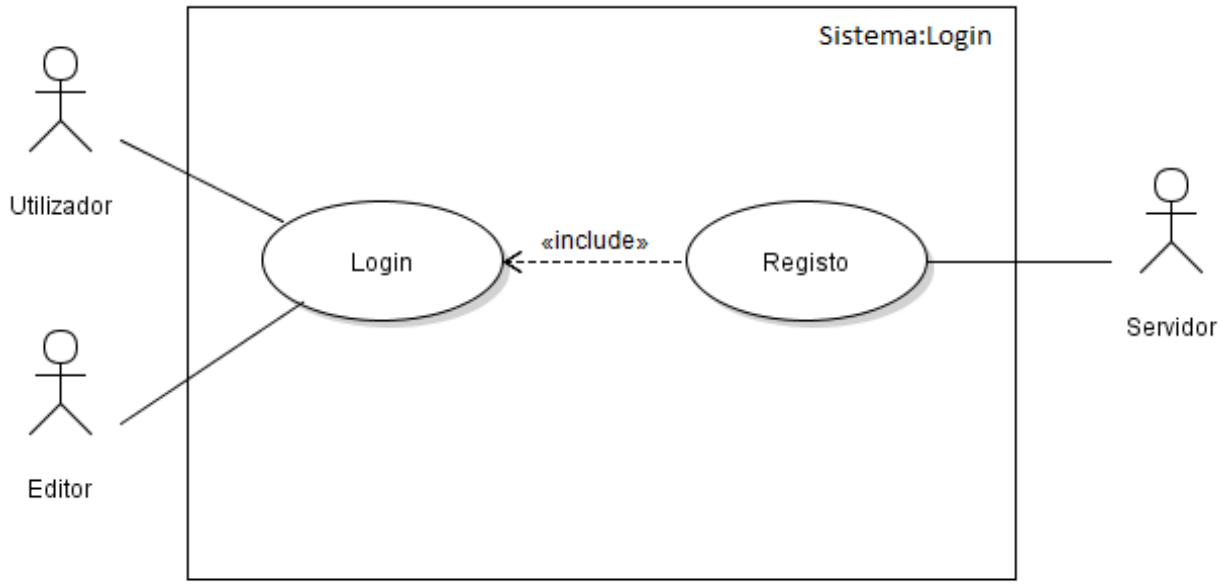

Figura 6: Diagrama UML de casos de uso Login (sea level)

No diagrama da figura 6, observamos as ações relacionadas com o *login* e com o registo por parte de um utilizador (editor ou não).

No caso de uso *login*, o utilizador terá de introduzir o seu nome de utilizador (*username*) e a sua palavra passe (*password*) e, caso a combinação dos dois parâmetros existir na base de dados, é-lhe garantido o acesso à aplicação. Caso contrário, o utilizador não inicia a sessão na aplicação, pois não lhe está associado uma conta.

O utilizador poderá, também, querer registar-se na aplicação, para posteriormente efetuar a ação de *login*. Para tal, é-lhe pedido uma série de parâmetros que ficarão associados à sua conta.

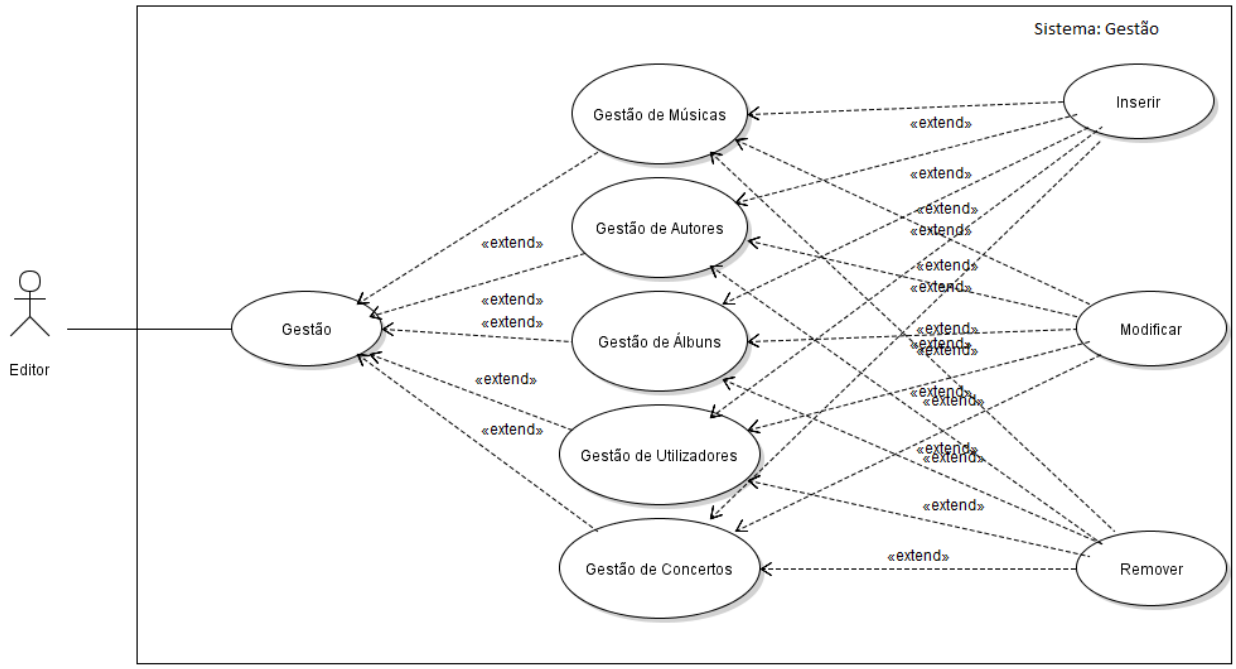

Figura 7: Diagrama UML de casos de uso Gestão (sea level)

No diagrama da figura 7, observamos as ações relacionadas com a gestão de músicas, álbuns, autores, utilizadores e concertos por parte um editor (utilizador com privilégios).

No caso de uso gestão de músicas, o editor poderá querer inserir músicas à base de dados, modificar parâmetros das músicas já existentes na base de dados ou remover músicas que existem atualmente.

No caso de uso gestão de autores, o editor poderá querer inserir autores à base de dados, especificando se o autor é artista, grupo musical ou compositor, modificar parâmetros de alguns dos tipos de autor já existentes na base de dados ou remover um dos tipos de autor que existem atualmente.

No caso de uso gestão de álbuns, o editor poderá querer inserir álbuns à base de dados, especificando as músicas que pretende adicionar ao álbum a criar, modificar parâmetros dos álbuns já existentes na base de dados ou remover álbuns que existem atualmente.

No caso de uso gestão de utilizadores, o editor poderá querer adicionar utilizadores, garantindo-lhes o acesso à aplicação da base de dados, especificando o tipo de utilizador, modificar informações ou privilégios associados à conta de utilizadores já existentes na base de dados ou remover contas de utilizadores, negando-lhes o acesso.

No caso de uso gestão de concertos, o editor poderá querer inserir concertos à base de dados, modificar parâmetros dos concertos já existentes na base de dados ou remover concertos que existem atualmente.

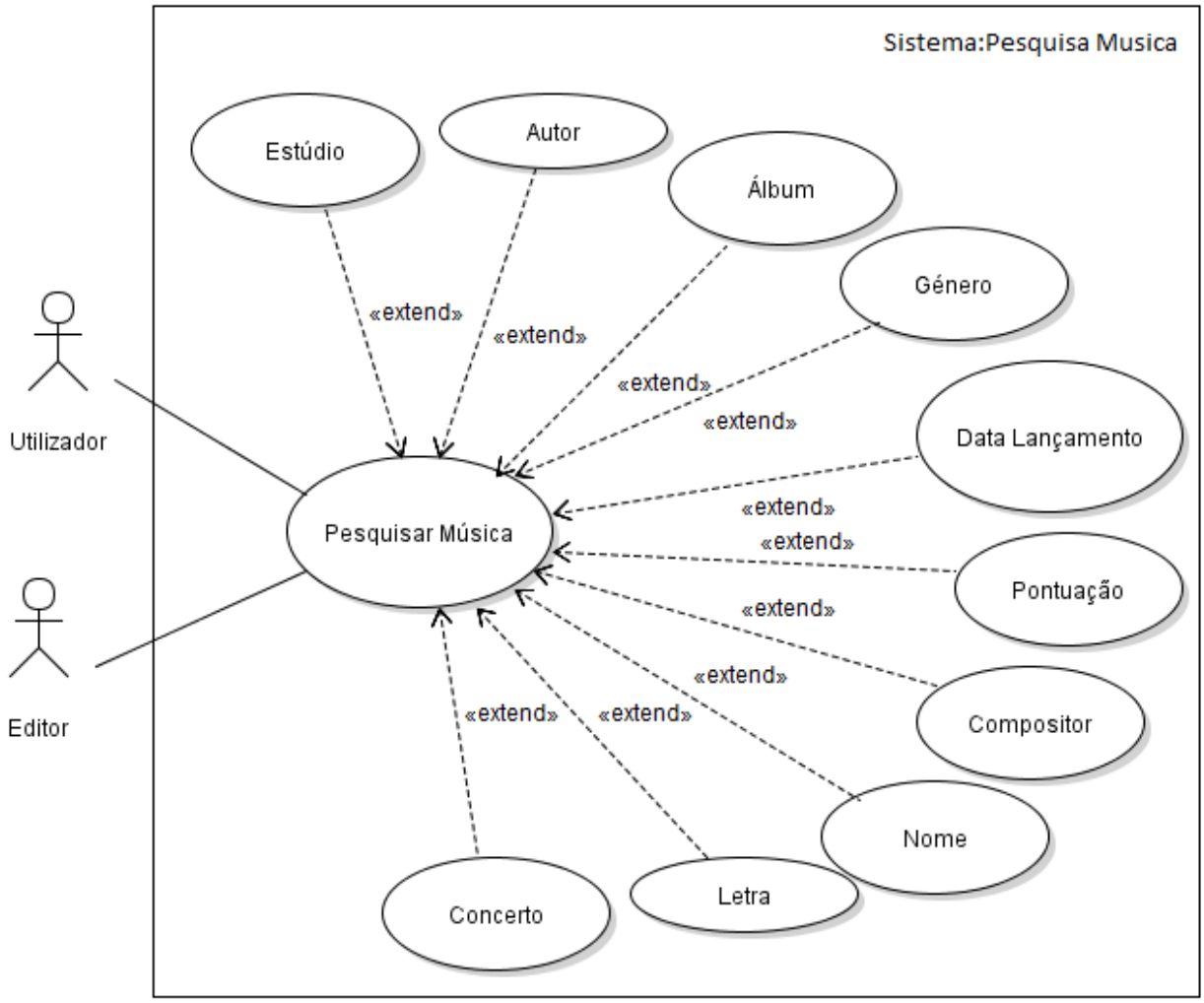

Figura 8: Diagrama UML de casos de uso Pesquisar Musica (underwater level)

No diagrama da figura 8, observamos as ações relacionadas com a pesquisa de músicas por parte de um utilizador (editor ou não).

No caso de uso pesquisar música, o utilizador poderá querer pesquisar uma música, existente na base dados, de várias maneiras. Para tal, o utilizador poderá escolher de que forma pretende pesquisar, sendo os resultados da pesquisa associados ao parâmetro inserido pelo utilizador. Desta forma, o utilizador poderá pesquisar a música pelo seu nome, género, data de lançamento, estúdio de gravação, autor, álbum, pontuação, compositor, letra ou concerto.

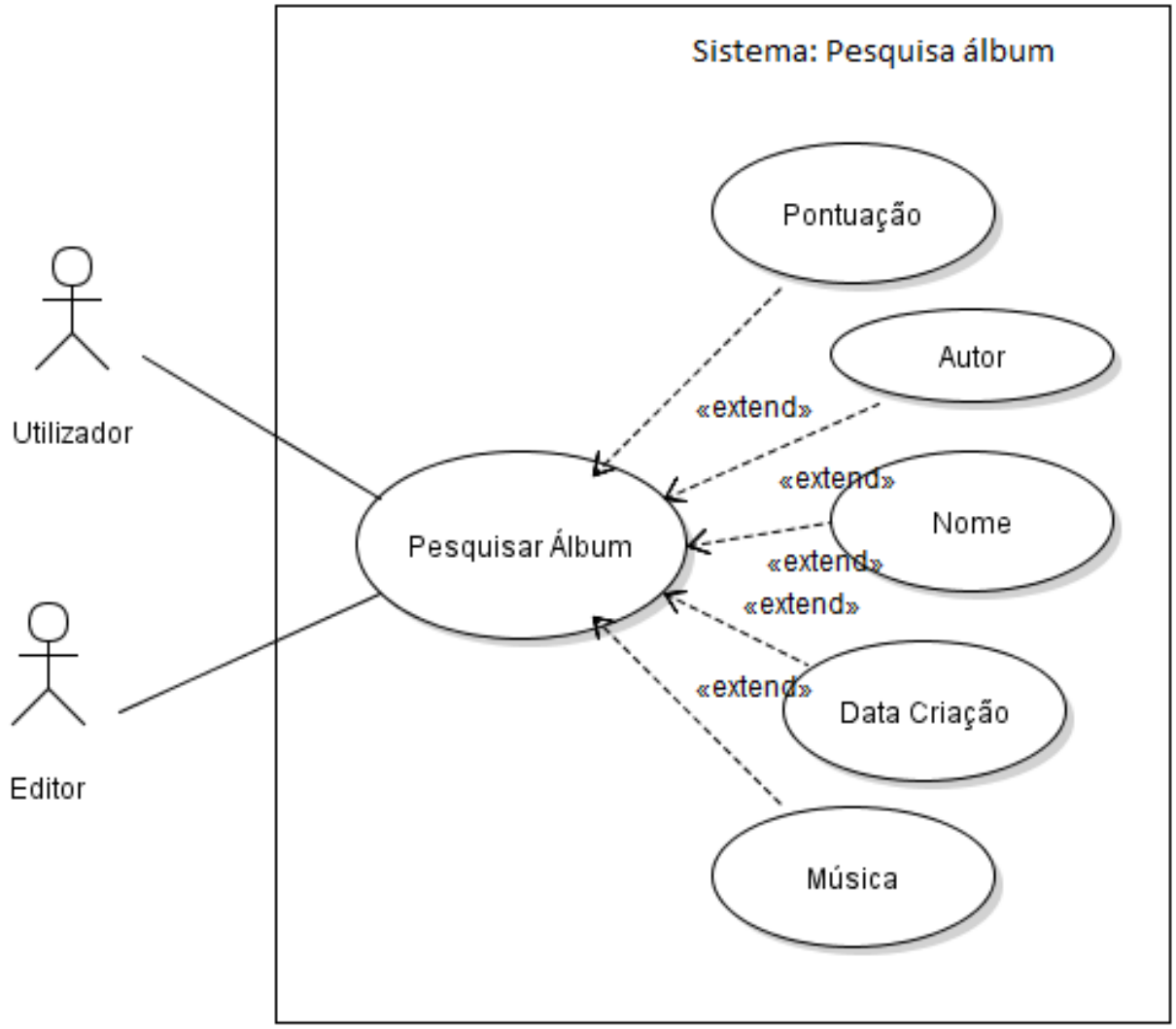

Figura 9: Diagrama UML de casos de uso Pesquisar Álbum (underwater level)

No diagrama da figura 9, observamos as ações relacionadas com a pesquisa de álbuns por parte de um utilizador (editor ou não).

No caso de uso pesquisar álbum, o utilizador poderá querer pesquisar um álbum, existente na base dados, de várias maneiras. Para tal, o utilizador poderá escolher de que forma pretende pesquisar, sendo os resultados da pesquisa associados ao parâmetro inserido pelo utilizador. Desta forma, o utilizador poderá pesquisar o álbum pelo seu nome, data de criação, pontuação, autor, mas também através de um nome de uma música pertencente ao álbum de pesquisa.

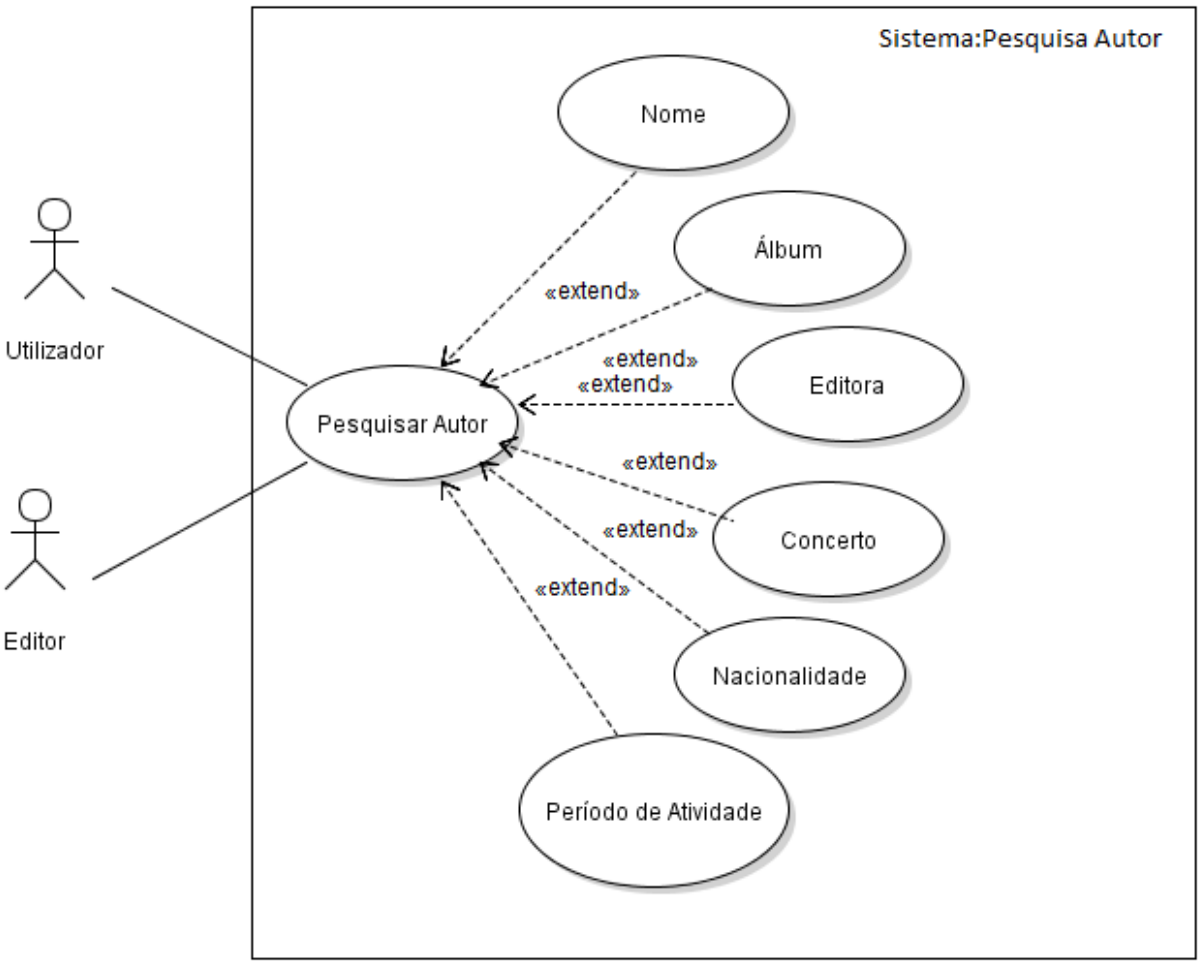

Figura 10: Diagrama UML de casos de uso Pesquisar Autor (underwater level)

No diagrama da figura 10, observamos as ações relacionadas com a pesquisa de autores por parte de um utilizador (editor ou não).

No caso de uso pesquisar autor, o utilizador poderá querer pesquisar um autor (artista, grupo musical ou compositor), existente na base dados, de várias maneiras. Para tal, o utilizador poderá escolher de que forma pretende pesquisar, sendo os resultados da pesquisa associados ao parâmetro inserido pelo utilizador. Desta forma, o utilizador poderá pesquisar o autor pelo seu nome, editora, nacionalidade, período de atividade, mas também através de um nome de uma música produzida pelo autor da pesquisa ou ainda através de um concerto em que o autor da pesquisa atuou.

6 Descrição do código SQL

Neste ponto falaremos brevemente do código SQL que implementámos na API, nomeadamente nas funções de procura, inserção, modificação e remoção de músicas, uma vez que estas funcionalidades são as que nós achamos ser as mais importantes para o funcionamento da aplicação, e porque outras funções acabam por ser semelhantes.

Procura de música:

```
SELECT m.musicaid,m.nomemusica, a.nomeautor
FROM musica_ficheiro as m,autor as a
WHERE m.autor_autorid = a.autorid
AND m.nomemusica like %s
ORDER by m.musicaid
```

Nesta função selecionamos o ID da música, o nome da música e o nome do autor dessa mesma música, para isso é necessário primeiro juntar as tabelas musica_ficheiro com a tabela autor de modo a que se possa saber a que autores pertence cada música. Seguidamente faz-se uma restrição que servirá como procura, neste caso em particular, o utilizador introduz um nome e o código SQL compara esse nome com todos os nomes de musicas existentes na tabela, por fim é feito um ordenamento pelo ID.

Inserção de músicas:

```
INSERT INTO musica_ficheiro(nomemusica,datalancamento,informacoes,duracao,letra,
estudiogravacao,ficheiro_nomeficheiro,ficheiro_tamanhoficheiro,ficheiro_tipoficheiro,
autor_autorid)
VALUES (initcap(%s),%s,%s,%s,%s,initcap(%s),%s,%s,%s,%s)

INSERT INTO musica_ficheiro_utilizador
VALUES((SELECT max(musicaid) FROM musica_ficheiro),(SELECT email FROM utilizador
WHERE nomeutilizador = %s));

INSERT INTO musica_ficheiro_genero
VALUES((SELECT max(musicaid) FROM musica_ficheiro), %s);
```

Nesta função o editor poderá inserir músicas na base de dados, nomeadamente na tabela musica_ficheiro, onde vão estar guardadas todas as informações relativas à música. Não é preciso introduzir a chave primária *musicaid*, uma vez que esse atributo é *serial*, ou seja, vai-se incrementar por 1 sempre que se insere algo na tabela. Para além disto será também necessário associar o utilizador que inseriu a música e o género musical nas tabela musica_ficheiro_utilizador e musica_ficheiro_genero respetivamente.

Modificação de músicas:

```
UPDATE musica_ficheiro
SET nomemusica = initcap(%s),
datalancamento = %s,
informacoes = %s,
duracao = %s,
letra = %s,
estudiogravacao = initcap(%s),
ficheiro_nomeficheiro = %s,
ficheiro_tamanhoficheiro = %s,
ficheiro_tipoficheiro = %s,
autor_autorid = %s
WHERE musicaid = %s"""

UPDATE musica_ficheiro_genero
```

```
SET genero_tipo = %s
WHERE musica_ficheiro_musicaid = %s"""
```

Nesta função o editor poderá modificar toda a informação associada a uma determinada música na base de dados, armazenada na tabela musica_ficheiro, sendo utilizado como referência o ID da música que é introduzido por ele. Para tal, o editor terá de fornecer todos os parâmetros que atualizam a música referenciada pelo ID, nomeadamente o nome da música, a sua data de lançamento, duração, letra, estúdio de gravação, o nome do ficheiro de áudio da música, o tamanho e o tipo do ficheiro, bem como o ID do novo autor da música. Como ainda se deu a possibilidade de alterar o género musical da música referenciada, então, caso o género musical seja válido (exista), é necessário atualizar a informação na tabela musica_ficheiro_genero que indica qual o novo género da música.

Remoção de músicas:

```
DELETE FROM musica_ficheiro
WHERE musicaid = %s
```

Nesta função apagamos a linha da tabela musica_ficheiro, cujo o ID é introduzido pelo editor.

7 Manual de Instalação

1. Instalar o sistema de gestão de base de dados (SGBD) PostgreSQL (https://www.openscg.com/bigsql/postgresql/installers.jsp).
2. Adicionar uma conceção do SGBD PostgreSQL ao servidor local com os seguintes parâmetros:
   - Name: localhost
   - Host: localhost
   - Username: postgres
   - Password: postgres
3. Criar uma nova base de dados (BD) manualmente, clicando na opção "New Database"no ícone "Databases", ou correndo o seguinte comando SQL:

   ```
   CREATE DATABASE dropmusic
   WITH OWNER = postgres
   ENCODING = 'UTF8'
   TABLESPACE = pg_default
   LC_COLLATE = 'Portuguese_Portugal.1252'
   LC_CTYPE = 'Portuguese_Portugal.1252'
   CONNECTION LIMIT = -1;
   ```

4. Instalar Python verão 3.6.7 (https://www.python.org/ftp/python/3.6.7/python-3.6.7.exe).
5. Instalar o adaptador do PostreSQL para suporte da linguagem de programação Python: Psycopg ((http://www.stickpeople.com/projects/python/win-psycopg/).
6. Correr os *scripts drop_tables.py* para eliminar, eventualmente, as tabelas existentes na base de dados *dropmusic*, *create_tables.py* para criar as tabelas na base de dados respetiva e *alter_tables.py* para adicionar as restrições aos atributos das tabelas e as relações entre estas.

8 Manual de Utilizador

1. Iniciar a aplicação no terminal
2. O utilizador deverá usar os números do teclado para selecionar a opção desejada.
3. O utilizador deverá introduzir dados quando o programa solicitar.
4. Fazer o Registo/Login

    - Consultar
        - Pesquisar musica
        - Pesquisar álbum
        - Pesquisar autor
    - Gerir PlayLists
        - Criar playlist
        - Modificar playlist
        - Remover playlist
        - Procurar playlist
    - Partilhar
        - Ver músicas partilhadas com o utilizador
        - Partilhar música(s) com outro utilizador
    - Gerir músicas (Só para editores)
        - Inserir música
        - Modificar música
        - Remover música
    - Gerir álbuns (Só para editores)
        - Inserir Álbum
        - Modificar Álbum
        - Remover Álbum
    - Gerir autores (Só para editores)
        - Inserir um artista
        - Inserir um grupo musical
        - Inserir um compositor
        - Modificar um artista
        - Modificar um grupo musical
        - Modificar um compositor
        - Remover Autor
    - Gerir utilizadores (Só para editores)
        - Inserir utilizador
        - Modificar utilizador
        - Remover utilizador
    - Gerir concertos (Só para editores)
        - Inserir concerto

        - Modificar concerto
        - Remover concerto
    - Gerir editoras (Só para editores)
        - Inserir editora
        - Modificar editora
        - Remover editora
5. Logout/Sair
6. Fechar o terminal

Referências

[1] Pedro Furtado, "Projecto de Bases de Dados", Disciplina de Base de Dados, Departamento de Engenharia Eletrotécnica e de Computadores, Faculdade de Ciências e Tecnologia da Universidade de Coimbra, 2018.

[2] Alexandre Pinto e Nuno Laranjeiro, "Online Database Architect(v3)"Internet: http://onda.dei.uc.pt/v3/, 1 de Fevereiro, 2018 [Novembro, 2018].

[3] *Quick Reference, SAP PowerDesigner Documentation Collection*, pp. 18-20, 23 de Março, 2018.

[4] The PostgreSQL Global Development Group, "PostgreSQL Documentation"Internet:https://www.postgresql.org/docs/, 2018

[5] Cay Horstmann e Alexandre Pellegrin,"Violet UML Editor"Internet:http://alexdp.free.fr/violetumleditor/page.php,[Novembro, 2018]