FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Relatório

Bases de Dados

Saulo José Mendes, 2021235944

Catarina Silva, 2021216307

Diogo Barbosa, 2021234034

19 de maio de 2023

Índice

Índice **2**
Introdução **3**
Guia de Instalação **4**
Guia de Utilização **5**
Design da Base de Dados **8**
Timeline do Projeto **10**
Conclusão **10**

Introdução

Este relatório apresenta os detalhes e resultados do projeto desenvolvido no âmbito da disciplina Bases de Dados. Em concreto, o projeto consiste na criação de uma plataforma de streaming de música que permite aos consumidores ouvir e aos artistas publicar músicas. Ademais, a plataforma oferece aos utilizadores um mecanismo de interação social através de comentários, que podem ou não conter a avaliação sobre uma determinada música.

O projeto foi implementado com recurso à PostgresSQL para a gestão da base de dados, e o servidor foi desenvolvido utilizando o framework Flask em conjunto com a linguagem Python.

Guia de Instalação

Para o funcionamento íntegro da aplicação é preciso garantir que a base de dados é criada corretamente, e que o servidor está a correr através do código fornecido em Flask/Python.

Após criar a base de dados *slayfy*, deve-se correr os seguintes scripts:

1. "tabelas.txt" para criar as tabelas;
2. "update_top_ten_trigger.txt" para inicializar o *trigger* para a atualização das playlists top 10;
3. "examples.txt" que contém diversos exemplos que facilitam a utilização inicial da plataforma

O script "examples.txt" permite a adição de vários dados na base de dados que permitem verificar as funcionalidades especificadas no enunciado. Nomeadamente, temos a adição de:

- um administrador;
- três consumidores (não premium);
- três artistas;
- doze músicas;
- dois álbuns;
- uma publisher;
- quatro cartões pré-pagos;
- dois comentários (um comentário original e uma resposta);
- quatro playlists (três playlists Top10 criadas aquando da criação dos consumidores e uma playlist criada por um consumidor);
- vários registos referentes à *listening_history* dos três consumidores, e registos das subsequentes atualizações às playlists TOP10 (feitas através do *trigger* update_top_ten).

Por fim, deve-se correr o código fornecido "slayfy.py". Vale a pena sublinhar que os dados sobre o utilizador, palavra-passe, servidor e porto devem ser alterados consoante as informações de cada *host*. Para além disso, para executar os *requests* no Postman é preciso criar os campos Authentication (onde se pode inserir o token de autenticação) e Content-Type (que deverá ter o valor application/json).

Nota: também foi submetido o script "drop_all_tables.txt" que permite facilmente apagar todas as tabelas e o *trigger*.

Guia de Utilização

Um utilizador pode interagir com a base de dados de várias formas. Nomeadamente, deve estar atento às seguintes funcionalidades*:*

1. Registo de Utilizador

Tal como numa aplicação legítima, um utilizador terá a oportunidade de se registar na aplicação. Os exemplos ADD_USER e ADD_ARTIST da coleção fornecida com o presente relatório permitem visualizar esta opção.

Para criar o utilizador, são precisos os campos *username, password, mail* e *artist_permit.* O campo *artist_permit* é um *boolean* que diferencia artistas de consumidores:

- Se *artist_permit* for "False", então temos um utilizador a registar-se pela primeira vez como consumidor. Isto quer dizer que não é possível haver uma sessão iniciada nesse momento (logo, se o *header* Authentication estiver preenchido, irá dar erro). O consumidor tem inicialmente estado *premium* False e, neste momento, também é criada a playlist TOP 10 que será atualizada pelo sistema sempre que o consumidor ouvir uma música
- Se *artist_permit* for "True", então está a ser criado um novo artista. Conforme especificado no enunciado, apenas administradores podem criar artistas. Isto quer dizer que um administrador terá de estar autenticado. Além disso, para criar um artista são precisos os campos *artistic_name* e *genre.*

Nota 1: administradores apenas podem ser inseridos na base de dados manualmente.

Nota 2: o *id* dos users não tem de ser passado no *request* pois foi definido como *autoincrement.* O mesmo acontece com a maioria das entidades na base de dados.

2. Autenticação de Utilizador

Em vez de se registar, o utilizador pode fazer *log in* com credenciais pré-existentes. Se inserir credenciais corretas, então é-lhe fornecido um *token* de autenticação que será preciso para outras *features.* Utilizámos o JWT *token*, tal como recomendado no enunciado. São codificados neste *token* alguns dados que permitirão a *endpoints* subsequentes identificar imediatamente o utilizador, nomeadamente: username, id, artist_permit e admin_permit.

Claro, para simular a autenticação nos próximos *endpoints,* é preciso copiar o *token* para o *header* Authentication nos *requests* no Postman.

O request LOG IN na nossa coleção permite visualizar esta opção.

Além disso, antes de iniciar a sessão, se o utilizador for um consumidor, será verificado o estado das suas subscrições, ou seja, se tem alguma subscrição ainda em vigor, para atualizar o seu estado *premium.* Isto permitirá gerir todas as *features* premium que o consumidor possa desejar fazer após se autenticar.

3. Criação de Música

Músicas apenas podem ser criadas por artistas autenticados. O *request* ADD_SONG na nossa coleção é um exemplo desta *feature.*

Para criar uma música, é preciso adicionar no *request* o campo *title, release_date, genre, publisher* e *other_artists.* O campo *other_artists* é uma lista com os ids dos artistas que colaboraram na música. O artista "principal" da música é definido como o que estava autenticado aquando da execução do *request,* utilizando o seu *id* que foi obtido da descodificação do *token* de autenticação.

As associações entre artistas e as músicas em que colaboraram são registadas na tabela de ligação *artist_song.*

4. Criação de Álbum

Tal como músicas, apenas artistas autenticados podem criar álbuns. O *request* ADD_ALBUM na nossa coleção é um exemplo desta fonte.

Para criar um álbum, é preciso adicionar no *request* o campo *title, release_date, genre, publisher* e *songs.* O campo *songs* é uma lista que pode conter *ids* de músicas pré-existentes e dicionários com os dados necessários para criar uma nova música e inseri-la no álbum.

O artista apenas pode associar ao álbum músicas que criou ou nas quais colaborou. [1]

5. Procurar Música

Qualquer utilizador autenticado pode executar esta *feature.* É adicionado no URL uma *keyword* e é devolvida informação sobre todas as músicas com essa *keyword* no título. Esta informação consiste em: todos os artistas relacionados com esta (artista principal e colaboradores) e o álbum ao qual ela pertence (se ela pertencer a algum). O *request* SEARCH_SONG ilustra este endpoint. [2]

6. Detalhar Artista

Qualquer utilizador autenticado pode executar esta *feature.* É adicionado no URL o id do artista que queremos procurar. É devolvida informação sobre todas as músicas, álbuns e playlists a que ele está associado. O *request* DETAIL_ARTIST ilustra este endpoint.

7. Fazer Subscrição

Um consumidor autenticado pode executar uma subscrição para ser *premium.* É necessário colocar no *request* o campo *period* com o período de tempo desejado (1 para mensal, 3 para trimestral e 6 para semestral) e o campo *cards* com os cartões que o utilizador deseja utilizar para a compra.

---

[1] No nosso *design* da base dados, uma música pode estar associada a no máximo um álbum. Ao tentar associar uma música a um álbum quando esta já está inserida noutro álbum, é gerado um erro interno da base de dados.

[2] A procura é *case insensitive.*

Primeiro, é verificado se o utilizador já tem uma subscrição ativa. Se tiver, então o início da nova subscrição torna-se o final da subscrição ativa nesse momento. Senão, a subscrição começará no presente momento.

Depois disso, é verificado se todos os cartões que o utilizador deseja utilizar lhe pertencem ou não pertencem a ninguém. Se o utilizador quiser utilizar um cartão que pertence a outra pessoa, a operação toda é terminada. Se o utilizador quiser utilizar um cartão que não pertence a ninguém, então irá tornar-se dono do cartão e mais ninguém o poderá utilizar.

Por fim, tenta-se pagar a subscrição utilizando o valor que resta nos cartões. Apenas são considerados cartões não expirados, mas tentar utilizar um cartão expirado não resulta no *rollback* da transação. Também são feitas associações entre os cartões e a subscrição (para fazer registo do pagamento).

Caso se consiga pagar a totalidade da subscrição, a transação é *committed* e o consumidor passa a ser premium.

O *request* SUBSCRIBE_TO_PREMIUM ilustra este endpoint.

8. Criar playlist

Apenas consumidores podem criar playlists. Apenas consumidores premium podem criar playlists privadas. É necessário adicionar os campos *title, public* (boolean) e *songs,* que é uma lista com as músicas que se quer adicionar à playlist.. Também há um campo *description* que é opcional. O *request* CREATE_PLAYLIST permite visualizar este endpoint.

9. Ouvir música

Apenas consumidores podem ouvir músicas. Ao ouvir a música, é adicionado um registo na *listening_history*. Sempre que um consumidor ouve uma música, a playlist Top Ten é automaticamente atualizada pelo próprio sistema através de *triggers.* O *request* PLAY_SONG permite visualizar este endpoint.

10. Gerar Cartões Pré-Pagos

Função reservada apenas aos administradores, permite que se crie um ou mais cartões em simultâneo, mas deve-se ter em atenção que só se pode criar cartões de 10, 25 ou 50 euros. Cada cartão recebe um id único de 16 dígitos, e, posteriormente, pode ser atribuído a um utilizador. O *request* GENERATE_PREPAID_CARD permite visualizar este endpoint.

11. Fazer Comentários

Todos os consumidores têm a possibilidade de deixar um comentário sobre uma música. Para além disso, podem, de maneira opcional, avaliar a música numa escala de 1-5. Para esta função não há limites no número de comentários e pode-se comentar a mesma música mais de uma vez. Ademais, há também a possibilidade de interagir com um comentário existente, criando uma linha do género comentário/resposta. Os *requests* LEAVE_COMMENT e LEAVE_REPLY permitem ilustrar este endpoint.

12. Generate monthly report

Os consumidores também têm disponível uma ferramenta que lhes permite ver a história das músicas ouvidas nos últimos doze meses a contar de uma data desejada. Para este efeito, é-lhes mostrada a sua atividade a partir do dia 20 do 12º mês mais afastado até chegar ao mês desejado. O *request* GET_MONTHLY_REPORT permite ilustrar este endpoint.

Design da Base de Dados

Ao desenvolver o projeto, foi necessário fazer certas alterações ao design da base de dados definido para a entrega intermédia, nomeadamente:

- As chaves primárias das entidades foram definidas como *autoincrement;*
- O atributo *title* foi adicionado à entidade Playlist;
- O atributo *name* foi adicionado à entidade Publisher;
- A entidade Comment deixou de ser uma *weak* entity, para facilitar a implementação da *feature* de deixar comentários/respostas.
- O *check* "balance > 0" na entidade Card foi ajustado para "balance >= 0" para efetuar os pagamentos das subscrições dos consumers.
- A nossa conceção inicial implicava que os cartões pré-pagos, ao serem criados, eram imediatamente atribuídos a um consumidor. Esta relação foi alterada para que os cartões não tenham nenhum dono de início, e um consumidor apenas tome posse de um cartão ao usá-lo num pagamento. Isto implicou a criação de uma tabela de ligação *consumer-card.*
- A entidade Authentication foi criada para guardar os *tokens* de autenticação dos utilizadores.

Os diagramas concetual e físico, tal como o ficheiro *json* correspondente, são enviados em anexo para facilitar a sua visualização.

A base de dados final apresenta algumas particularidades que devem ser atentadas. Primeiro, os *tokens* de autenticação JWT não têm data de expiração. Estamos cientes que, numa situação de gestão de dados real, isto seria uma escolha bastante insegura, mas decidimos definir os *tokens* como estáticos para simplificar a *feature* de autenticação. Segundo, tanto os álbuns como as músicas têm um atributo *publisher,* e este pode ser diferente entre um dado álbum e as músicas inseridas nesse álbum. Terceiro, o *monthly report* é construído pré-definidamente a partir do dia 20 do mês e do ano inseridos pelo consumidor, às 12h. Por último, é importante sublinhar que, na nossa conceção da base de dados, e tal como já foi referido na explicação do endpoint ADD_ALBUM, uma música apenas pode pertencer no máximo a um álbum.

Gestão de Transações

Existem várias funcionalidades, como, por exemplo, a criação de um álbum ou a efetuação de uma subscrição, que exigem várias operações que têm de ser realizadas atomicamente, ou seja, ou acontecem todas ou não acontece nenhuma. Para gerir isto, utilizamos *conn.rollback* sempre que se verifica uma condição que implica abortar a transação, para "desfazer" as operações até a esse ponto, e fazemos *conn.commit* apenas quando a transação se completa até ao fim com sucesso, para oficializar todas as alterações.

Gestão de Concorrência

1. Registo de Utilizadores, Autenticação, Criação de uma Música, Playlist, Cartão ou Playlist

Estas transações podem criar certos problemas de concorrência. Imaginemos um exemplo em que iniciamos uma transação de criação de música e inserimos uma música com ID 3 na base de dados. Depois, antes de *commit* a transação, iremos criar várias associações entre a música e os artistas que colaboraram nela. Simultaneamente, outra transação começa para criar uma nova música e, como não "vê" as alterações *uncommitted* da primeira transação, não sabe que o ID 3 já foi usado e, portanto, cria uma segunda música com ID 3. No final, as duas transações são *committed* e temos duas músicas com a mesma chave primária. Este tipo de erro pode ocorrer em todas as operações mencionadas acima. Para evitar isto, no início das transações adquirimos um *shared lock* sobre as tabelas em questão, permitindo leituras mas não permitindo modificações nas tabelas até a transação ser *committed.*

Algumas tabelas de ligação não precisam de ser *locked.* Por exemplo, a tabela *artist_song* não precisa de ser *locked* pois só é alterada quando uma música é criada e, como a tabela *song* já foi *locked,* não há risco de alterações concorrentes na tabela *artist_song.*

Por outro lado, a tabela *song_playlist* pode ser alterada por dois tipos de transação diferentes: a criação de uma playlist e inserção de um registo na tabela *listening_history,* que despoleta um *trigger* que atualiza a playlist TOP10. Portanto, esta tabela também tem de ser *locked.*

2. Search Song, Detail Artist e Generate Monthly Report

São operações feitas através de um *query* SELECT, que adquire um *lock* implícito do tipo *shared lock* para permitir que várias transações leiam os dados em questão em simultâneo, mas nenhuma transação consiga alterar esses dados, proibindo assim automaticamente a corrupção destas operações. Logo, nestes endpoints não precisámos de gerir a concorrência explicitamente.

3. Fazer uma Subscrição

Tal como no caso de criação de outras entidades, como músicas e álbuns, é preciso fazer *lock* da tabela *subscription* para impedir criação de subscrições com a mesma chave primária. Também há outra situação que poderia gerar problemas de concorrência. Imaginemos que há dois consumidores a tentar utilizar o mesmo cartão para pagar uma subscrição. Como as transações não

têm conhecimento das alterações uma da outra, poderia acontecer que o cartão era usado pelos dois utilizadores, o que devia ser impossível. Contudo, como fazemos a operação INSERT na tabela *subscription* antes da etapa dos cartões, e uma transação inicial faz *lock* da tabela *subscription* até ser *committed,* transações subsequentes ficam suspensas até os pagamentos com os cartões da primeira transação se tornarem "oficiais", pelo que este conflito de concorrência também é resolvido.

4. Ouvir uma Música

Tal como foi referido acima, nesta operação é preciso fazer *lock* da tabela *song_playlist* para impedir que outras transações concorrentes criem registos na tabela com a mesma chave primária.

Timeline do Projeto

As tarefas do projeto foram divididas da seguinte forma:

- Catarina: registo de utilizadores, autenticação, criação de músicas, *feature* "get monthly report", subscrição.
- Diogo: *features* "search song" e "detail artist", criação de playlists, relatório, testes da base de dados.
- José: *trigger* update_top_ten, *feature* "play song", criação de álbuns, geração de cartões pré-pagos, *features* relacionadas com comentários.

Realizámos as tarefas em sessões conjuntas, tendo dedicado cerca de 29 horas no projeto.

Conclusão

Durante a realização do projeto, foi possível implementar com sucesso todas as funcionalidades essenciais da plataforma de streaming de música. Os utilizadores foram capazes de criar contas, sendo atribuídos os papéis de artista ou consumidor. Os artistas puderam publicar as suas músicas, enquanto os consumidores tiveram a oportunidade de explorar e ouvir uma ampla variedade de músicas disponíveis.

Através da combinação da linguagem PostgresSQL e do Flask/Python, foi possível garantir um desempenho adequado e uma experiência fluida na plataforma. As consultas à base de dados são adequadamente protegidas em termos de gestão de transações e concorrência.