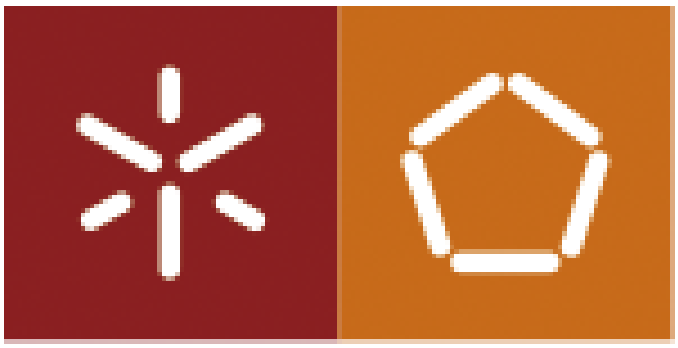


Licenciatura Engenharia Informática

Universidade do Minho

Laboratórios de Informatica III

Grupo 31:

- Manuel Fernandes (A93213)
- Tomás Machado (A104186)
- Francisco Maia (A108962)

ÍNDICE 1

Introdução:

Este segundo relatório visa expor o trabalho feito na segunda fase do projeto da unidade curricular de Laboratórios de Informatica III. Tendo em conta o novo enunciado dado, com novas queries e entidades, novos "datasets" de maior tamanho e as críticas feitas durante a defesa do projeto pelos docentes da cadeira, o grupo teve de se adaptar a este novo desafio, fazendo as alterações necessárias ao código. Neste relatório serão explicadas as resoluções destes novos problemas, as adaptações feitas a nível das estruturas de dados, e como foi feito o modo interativo.

Relembrando que o trabalho proposto tem como objetivo criar um sistema de streaming de música na linguagem C, de destacar a importância de conceitos como encapsulamento e modularidade.

Dados:

Para esta segunda fase, para além dos dados já usados anteriormente (Users, Artistas e Músicas) foram adicionadas duas novas entidades, assim como alterações nos campos anteriores:

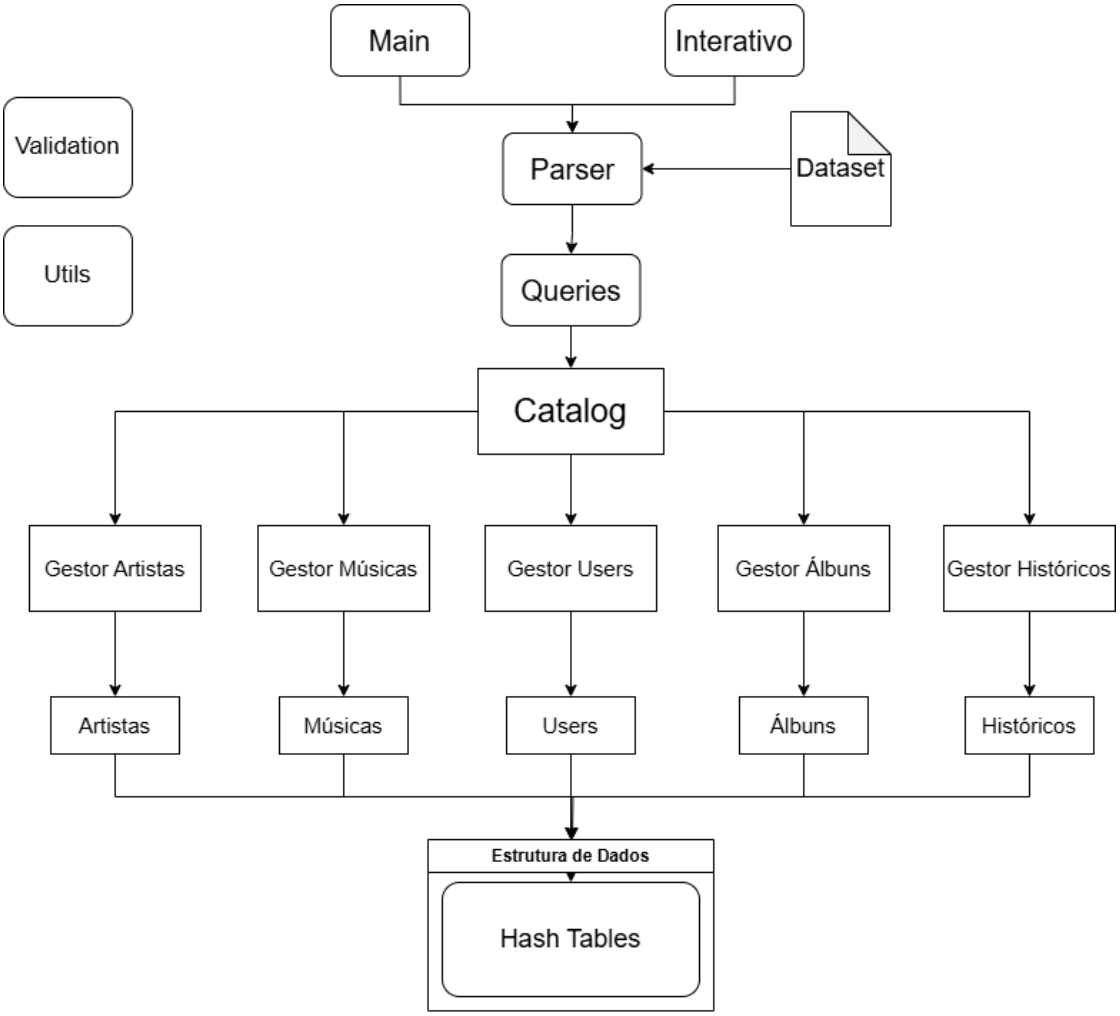
- **Users (Utilizadores)**, que tem um username, um email associado, o seu primeiro e último nome, a sua data de nascimento, o país onde a conta foi criada, o seu tipo de subscrição e lista de identificadores únicos das músicas gostadas pelo utilizador.
- **Artistas,** que tem um "id" único, o seu nome, alguns detalhes sobre o artista, o dinheiro gerado cada vez que uma música da sua autoria é reproduzida, a sua nacionalidade e se é um artista individual ou um grupo. No caso de ser um grupo, terá a lista dos membros.

- **Músicas**, que tem, tal como os artistas, um id único, o nome da música, a lista de ids dos autores da música, a duração, o género, o ano de lançamento e a letra, e agora contam com o identificador único do álbum ao qual a música pertence.
- **Álbuns,** que contem o id único do albúm, o título do álbum, a lista de ids dos artistas que lançaram o álbum, o ano de lançamento e a lista de Produtores.
- **Histórico,** tem um id único do registo, o id do utilizador a que o registo se refere, o id a música a que o registo se refere, data e hora em que a música foi ouvida pelo utilizador, duração da audição da música e plataforma em que a música foi reproduzida (computador ou dispositivo móvel).

Estruturas de dados:

Durante a defesa do projeto, os docentes chamaram a atenção do grupo devido ao uso de GPTRArrays e como estes podiam causar problemas a no futuro, não sendo então uma boa escolha a longo prazo. Isto pois, ao utilizar GPTRArrays com os datasets de maior tamanho, a execução perde bastante eficiência na busca dentro desses arrays. O grupo optou então por voltar a implementar as Hash Tables para todas as entidades.

Modulação:


Main
Interativo
Validation
Parser
Dataset
Utils
Queries
Catalog
Gestor Artistas
Gestor Músicas
Gestor Users
Gestor Álbuns
Gestor Históricos
Artistas
Músicas
Users
Álbuns
Históricos
Estrutura de Dados
Hash Tables

Para o nosso trabalho, foi utilizado o modelo usado acima apresentado. Em cada ficheiro .c de cada entidade, estão apenas funções do tipo "get", funções "create" e "destroy" que vão buscar informação necessária aos ficheiros de data que já foram validados pelo ***Parser*** (por exemplo, a função *get_user_first_name* na entidade ***Users***, irá buscar o primeiro nome de um determinado user).

O ficheiro ***Utils*** contém funções mais gerais, que podem ser aplicadas em diversos cenários (por exemplo a função *duration_to_seconds* que calcula o tempo de uma musica em segundos).

No ficheiro ***Validation***, estão funções que fazem validação de sintaxe e verificam a validade de certos campos, como email, subscrições, entre outros.

Decidimos criar um ***gestor*** para cada **entidade**, devido à complexidade de cada uma. Em termos de encapsulamento e isolamento, a separação em 5 gestores permite encapsular os dados e operações relacionadas a cada entidade, isolando funcionalidades e evitando que detalhes internos sejam acessados ou modificados fora do contexto adequado. Para garantir esse encapsulamento, temos o cuidado de não passar estruturas de dados para fora dos seus ficheiros ".c" respetivos e apenas passar copias dos valor, usando "*g_strdup*".

Esse **encapsulamento** reduz o risco de erros e facilita a manutenção. Com gestores distintos, qualquer mudança necessária numa entidade (como ajustes na estrutura de dados ou novos métodos de acesso) pode ser feita diretamente no gestor associado, sem impactar o restante do sistema. Isso torna a manutenção do código mais simples e organizado. A criação de gestores para cada entidade facilita também a reutilização de código e permite que futuras funcionalidades sejam adicionadas de forma modular. Por exemplo, novos métodos para manipulação de artistas podem ser incluídos diretamente no *Gestor_Artistas*, sem interferir nos outros gestores. Com esta modulação, garantimos uma clara separação de responsabilidades, o que melhora a legibilidade do código, pois cada gestor tem um papel bem definido. Outro bom exemplo da importância de uma boa modulação e encapsulamento foi o a mudança de GPTRArrays para hash tables, onde nem foi necessário alterar os ficheiros ".h" após a mudança de estruturas sugerida pela equipa docente.

Foi também criado um ficheiro ***Queries,*** que apenas serve como auxilar na implementação do modo interativo (será abordado mais à frente) e limita-se a imprimir os resultados das diferentes Queries pedidas. É também o local onde estão especificados os locais onde estão resolvidas cada uma das queries.

O ***Modo Interativo***, novidade desta segunda fase do projeto, consiste em deixar o utilizador interagir com o programa, escolhendo que queries fazer com que argumentos. No nosso caso, o programa dá as boas vindas ao utilizador e pede qual o caminho do dataset a processar ("*Indique o Caminho para o Dataset*\n"). Em caso de erro, o programa irá imprimir "Erro ao processar o ficheiro *x*.csv". Após isso, o programa irá perguntar quais as queries a executar *("Indique o número da query que pretende correr*\n*"*), e dará a opção de sair do modo interativo pressionando "0". Depois de indicado o número da query, o programa pede os argumentos a utilizar dependendo de que query o utilador pede, e se o argumento dado for correto, o programa irá devolver o resultado no formato desejado, caso contrário, a seguinte mensagem de erro irá aparecer (por exemplo): "Utilizador ou Artista não encontrado ou argumentos mal inseridos". O programa irá responder até o utilizador pressionar 0, o que fará com que saia do modo interativo.

Queries:

Houveram três novas queries na segunda fase do trabalho, e algumas alterações na query 1. Passamos então a explicar cada problema e a resolução do mesmo:

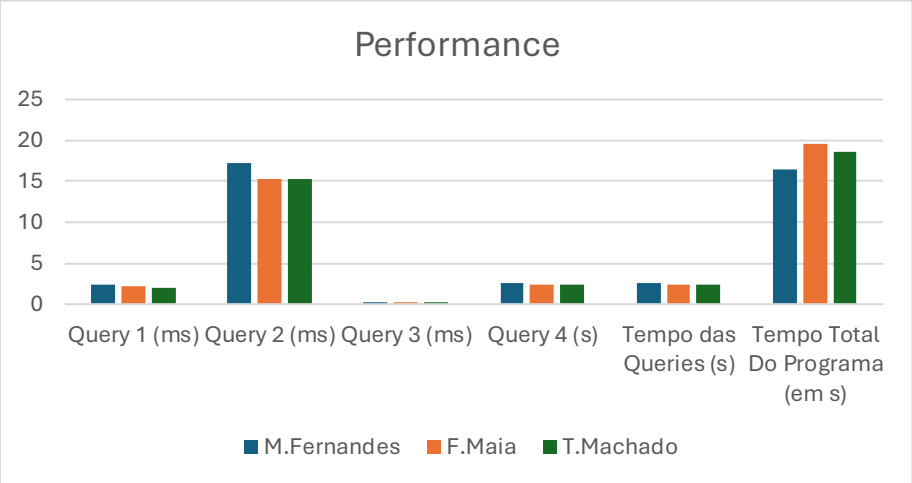
Na query 1, onde é pedido para "Listar o resumo de um utilizador ou artista, consoante o identificador recebido por argumento.", no caso de pedir o resumo de um artista, a função agora vai também devolver o número de albúns do artista e a sua receita, que é calculada a partir da soma da sua receita individual (nº de reproduções vezes o seu "rate per stream") e da sua receita de participações, caso esteja numa banda ou participe em musicas de outros artistas (nº de streams vezes o seu "rate per stream" a dividir pelo número de participantes).

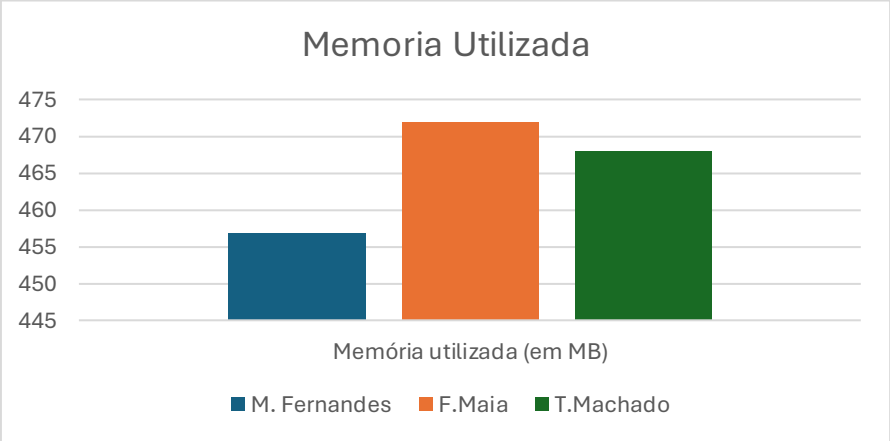
Query 4 : *"Qual é o artista que esteve no top 10 mais vezes?"*

Na query 4 é pedido para devolver o top 10 dos artistas que mais vezes constam nos 10 mais ouvidos a cada semana, podendo ser um intervalo de datas definido pelo utilizador, caso esse seja dado. Caso contrário, o programa considera a data 09/09/2024 como limite superior e a data mais antiga disponível no dataset como limite inferior. Importante notar que se considera que as semanas começam num domingo e acabam num sábado. Primeiramente, a função *calculate_week_position* calcula a posição de uma data em relação a uma semana específica, a partir da data de referência. Ela converte a data de entrada e a referência, calcula a diferença em dias e a converte para semanas. Isso permite organizar os registros de reprodução por semanas, mesmo quando o intervalo de datas ínclui semanas parciais. A função *update_artist_weeks_duration* é responsável por incrementar o contador do tempo total de reprodução para cada artista em semanas específicas. Com base no identificador da música e na data da reprodução, a função atualiza os tempos acumulados para o artista correspondente na semana correta. Esses dados são usados para calcular o top 10 semanal. Para determinar o top 10 de cada semana, a função *compare_artists_by_duration* compara dois artistas com base no tempo de reprodução registrado para uma semana específica. Caso dois artistas tenham o mesmo tempo de reprodução, a decisão de qual permanece é feita pelo menor ID do artista. A função *add_to_top10* utiliza essa lógica de comparação para gerenciar uma lista de até 10 artistas mais reproduzidos numa semana. Após adicionar um artista, a lista é ordenada, e, caso exceda 10 elementos, o último é removido . Após isso, o array de top 10 semanal é percorrido, e a cada vez que um artista aparace, aumenta-se o seu contador de vezes que esteve presente no top 10. Comparasse os contadores dos diferentes artistas, e determinasse por ordem decrescente o top (segue a mesma lógica de artistas com menor ID vem primeiro). Por fim, o resultado é devolvido no formato name;type;count_top_10.

Queries 5 e 6 não realizadas, a explicar na defesa!

Performance:





Processador:

- T.Machado: I7-1165G7 2.80 GHz

- M.Fernandes : AMD Ryzen 5 7535HS 3.30 GHz

- F.Maia: I7-1165G7 2.80 GHz

Conclusão:

Esta segunda fase do trabalho provou ser bem mais desafiante do que a primeira. O grupo mostrou algumas dificuldades nas resoluções das queries 5 e 6, pois apesar de haver um esboço daquilo que seria a resolução correta para os problemas, não foi possível concluir e refinar por completo o código. Na defesa do trabalho o grupo mostrará aquilo que tentou fazer e a lógica que tentamos tomar por detrás de cada problema. Quanto ao resto do trabalho: modo interativo, implementação das novas entidades, manter um bom encapsulamento e modulação, ter um código ausente de memory leaks e uma boa qualidade de código, o grupo conseguiu atingir grande parte dos objetivos propostos, demonstrando um bom domínio dos conceitos aprendidos, nomeadente modulação e encapsulamento, e entregou um projeto sólido e funcional nos outros aspetos. Como o grupo não terminou a query 5 e 6, em termos de performance não é possível fazer uma análise muito completa, de destacar que devido aos numerosos "loops" que ocorrem na resolução da query 4, o tempo de execução foi notávelmente maior, sendo então um ponto de melhoria que o grupo podia fazer.