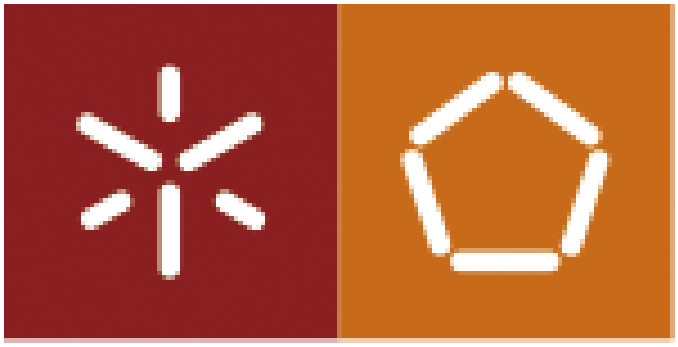


Licenciatura Engenharia Informática

Universidade do Minho

Laboratórios de Informatica III

Grupo 31:

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

ÍNDICE 1

Introdução:

Neste ano letivo de 2024/2025, no âmbito da Unidade Curricular de Laboratórios de Informatica III, foi nos proposto fazer um sistema de streaming de música na linguagem C, no qual teríamos de responder a um conjunto de perguntas (queries), aprimorar as nosssas competências essenciais à implementação de programas de forma estruturada em C, trabalhar com um conjunto de dados relativo a músicas, artistas, utilizadores,... com especial atenção aos conceitos de Modularidade e Encapsulamento, que foram extremamente importantes durante a realizacão do código. Na 1ª fase do trabalho, o grupo teve de responder às queries 1, 2 e 3.

Dados:

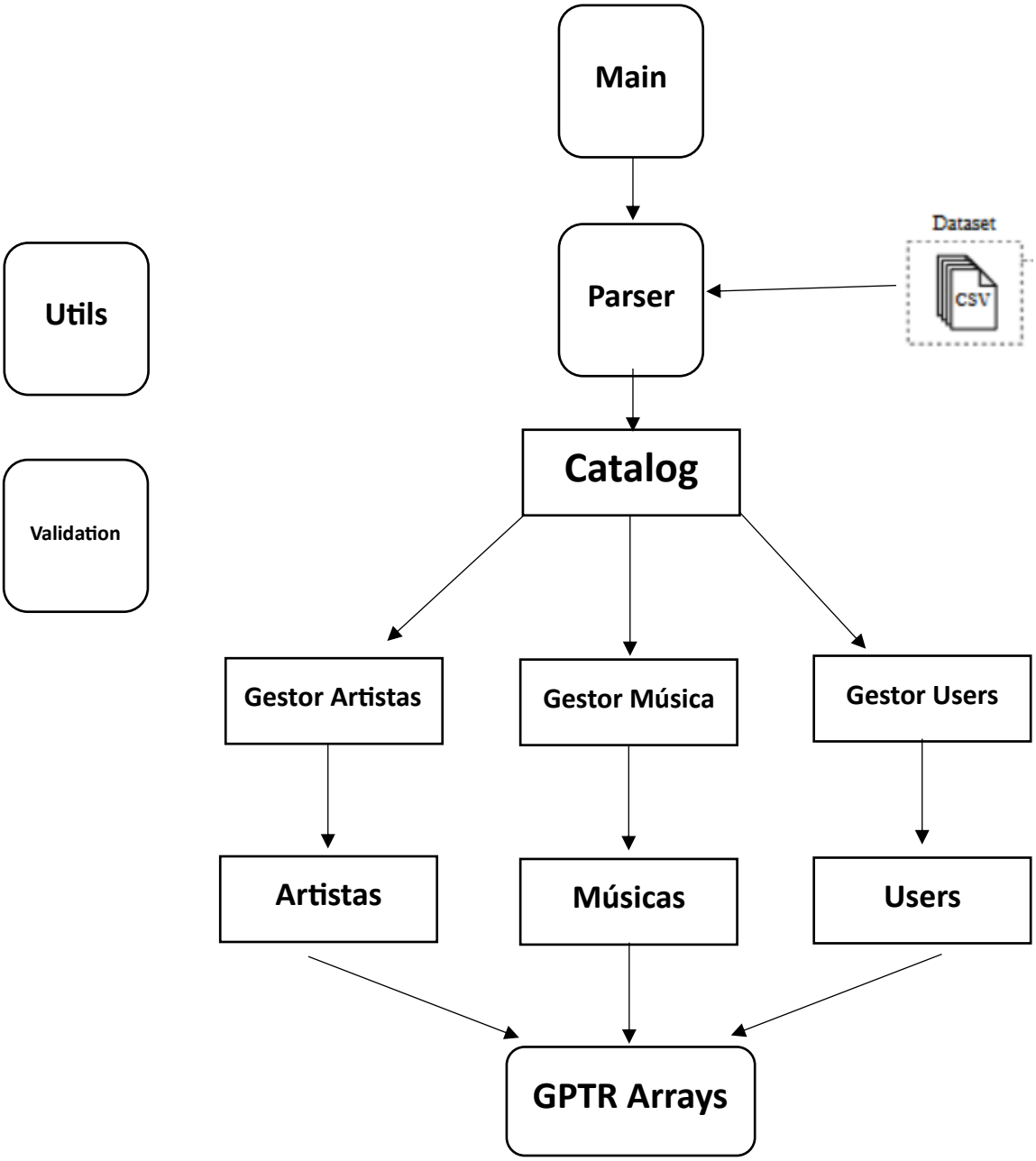
Nesta primeira fase do trabalho, trabalhos com três tipos de dados diferentes, cada um com os seus campos destintos;

- **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.

Estruturas de dados:

Inicialmente, o grupo implementou Hash Tables para armazenar os três tipos de dados. Apesar de a utilização de Hash Tables ser bastante eficaz quando bem implementadas, tendo uma inserção e busca de dados rápida, no nosso caso acontecia um grande número de colisões das “keys”, o que tornou a execução do programa mais lenta e mais consumidora de memória. Decidimos então optar por GPTR Arrays para o armazenamento dos dados. Nos GPTR Arrays, o id vai corresponder à sua posição no array. Por exemplo, uma música com o id 123 num array a, irá conrresponder à posição 123 do array: a [123]. Este método de armazenamento permite uma pesquisa muito eficiente, e, no caso de ser necessário a ajustar o tamanho do array para novos dados, a redimensão do mesmo é feita dinamicamente. Esta mudança de estruturas teve um agradável impacto na performance do código, tanto no tempo de execução como na memória utilizada, passando de mais ou menos 16 segundos de tempo e 450 MB de memória para 9.2 segundos e 317 MB de memória (segundo o programa de testes fornecido).

Modulação:


Main
Parser
Dataset
CSV
Utils
Validation
Catalog
Gestor Artistas
Gestor Música
Gestor Users
Artistas
Músicas
Users
GPTR Arrays

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 3 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.

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 organizada. 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_Artist*, sem interferir nos outros gestores. Com esta modulação, garantimos uma clara separação de responsabilidades, isso melhora a legibilidade do código, pois cada gestor tem um papel bem definido.

Queries:

Passamos agora a apresentar as três questões que tivemos de responder na 1ª fase do trabalho, explicando de forma sucinta como foi resolvida cada problema:

Query 1: *"Listar o resumo de um utilizador, consoante o identificador recebido por argumento"*

A query 1 tem uma resolução bastante rudimentar. Apenas foi necessário converter para um inteiro usando a função *user_id_int* e, após a verificação se o id do user existe, usamos as funções *get* para ir buscar os parâmetros email, primeiro nome, ultimo nome, idade e país do respetivo user ao dataset de Users que foi previamente validado pelo parser, e dar *print* de maneira a respeitar o output pedido.

Query 2: *"Quais são os top N artistas com maior discografia?"*

Na query 2, começamos por criar um GArray, onde guardamos todos os id's dos artistas validados pelo Parser. Em seguida iteramos sobre o GPTR array que contem todos os dados sobre cada artista. Existe a possibilidade de ser dado um país como parâmetro, para isso o código verifica se o mesmo acontece, e em caso afirmativo, filtra o array para conter apenas artistas desse país. Após isso, usamos a função *g_array_sort_with_data* para ordenar o array pela duração da discografia, comparada pela função *compare_artist_discography*, por ordem decrescente. Em caso de empate, o artista com menor id fica na posição cimeira.

Query 3: *"Quais são os géneros de música mais populares numa determinada faixa etária?"*

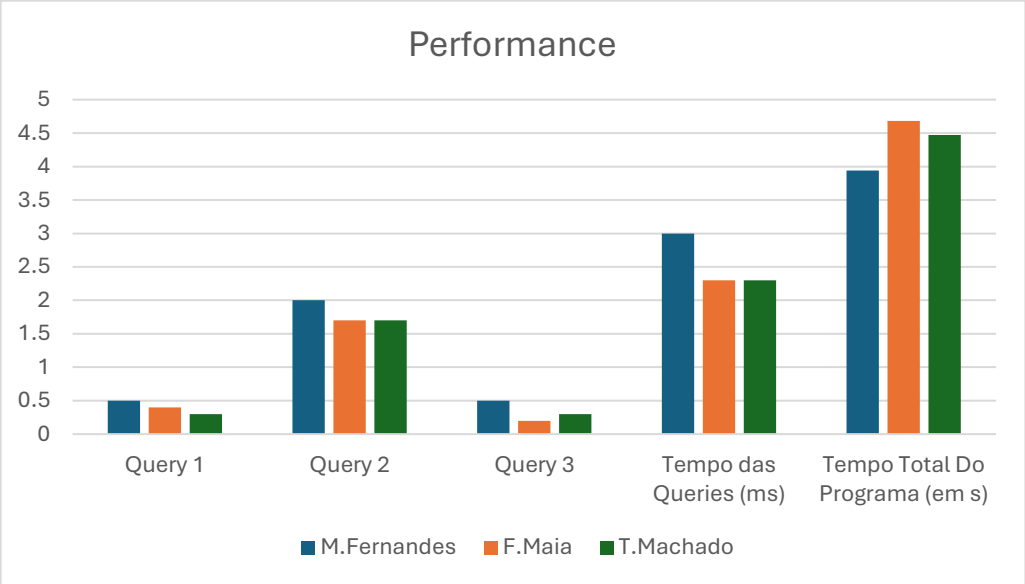
Para a query 3, começou-se por criar uma matriz 100x10 no Gestor de Users, que é inicializada quando criamos o gestor, onde temos 100 linhas/arrays para cada idade válida dos users e 10 colunas, uma para cada género. Cada posição da matriz representa a quantidade total de likes para um certo género numa determinada idade (por exemplo a posição 18x5 da matriz seria o número de likes que músicas do género hip hop tem contando apenas com users com 18 anos de idade). Depois, dependendo do intervalo pedido somamos o número de likes entre as idades para todos os géneros

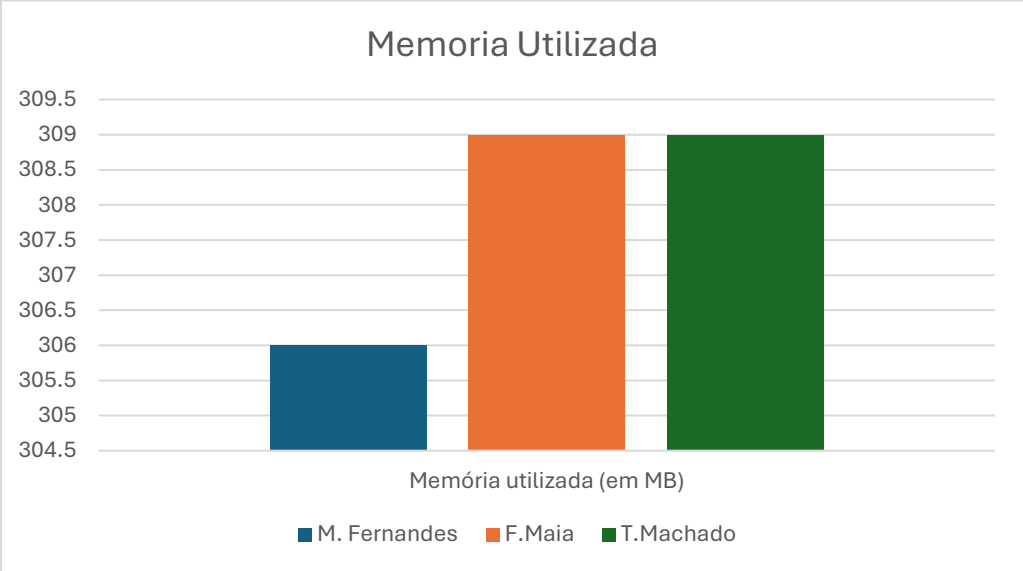
musicais e adiciona ao array *aggregated_genre_counts* na posição correspondente. Após agregar as contagens de likes de todos os géneros dentro da faixa etária, o código cria o array *genre_counts,* onde cada posição contem o género e a soma de likes, para facilitar a ordenação. Usamos o quick sort para ordenar o array de forma decrescente (em caso de empate, a função compara os gêneros alfabeticamente usando *strcmp).* Por fim, o código verifica se todas as contagens são zero, o que indicaria que não há likes para a faixa etária, se for o caso, escreve uma linha em branco no arquivo de saída, caso contrário, ele percorre *genre_counts* e escreve apenas os gêneros com contagens maiores que zero, no formato "gênero;contagem".

Performance:

Em termos de performance das queries, a query 2 leva mais tempo a ser executada comparativamente às queries 1 e 3. Enquanto que para a 1 e 3 temos um tempo de **O(1)**, para a query 2 levara **O(N)**, sendo N o número de artistas.

Correndo o programa de testes na máquina de cada um dos elementos do grupo, obtivemos os seguintes resultado





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:

A primeira fase do trabalho foi bastante bem sucedida. Comparativamente aos restantes grupos, temos um tempo de execução mais que aceitável, tendo apenas dois ou três grupos com um melhor tempo final. Com isto podemos concluir que a nossa otimização está no bom caminho. Reparamos também que tivemos uma utilização de memória elevada, o que será um aspeto a melhorar para a segunda fase do trabalho. Em relação aos novos conceitos aprendidos, nomeadamente a modularidade e encapsulamento, o grupo percebeu bem a necessidade destes duas novas aprendizagens, e penso que a aplicamos bem no projeto e serão importantes em projetos futuros. Um excelente exemplo das vantagens de utilização de modularidade, foi a mudança de Hash Tables para GTPR Arrays, pois as alterações feitas implicaram que apenas precisamos de alterar o modulo específico e sem necessidade de alterações nos “header files”.