Relatório 2ª Fase | Laboratórios de Informática III

Grupo 78 | 2024/2025

Afonso Carpinteiro (A106909)

Bruno Carvalho (A106809)

Índice

1. Introdução

Este relatório tem como objetivo apresentar informações relativas à 2ª Fase do Projeto da Unidade Curricular "Laboratórios de Informática", integrada no 2º ano da Licenciatura em Engenharia Informática, realizada no ano letivo 2024/2025, na Universidade do Minho.

Concluída a 1ª Fase em que foi implementado o *parsing* e a validação de dados, o programa-principal e o programa-testes e três *queries*, completámos agora os requisitos previstos na 2ª Fase: novas coleções de dados, outras três *queries* a implementar e a criação do programa-interativo.

2. Desenvolvimento

Para o projeto final, conseguimos implementar todos os modos descritos no enunciado (principal, testes e interativo), bem como todas as queries. Além disso, acrescentemos também documentação *Doxygen* em todos os ficheiros .h. No que toca a modularidade e encapsulamento, também acreditamos estar tudo tratado corretamente.

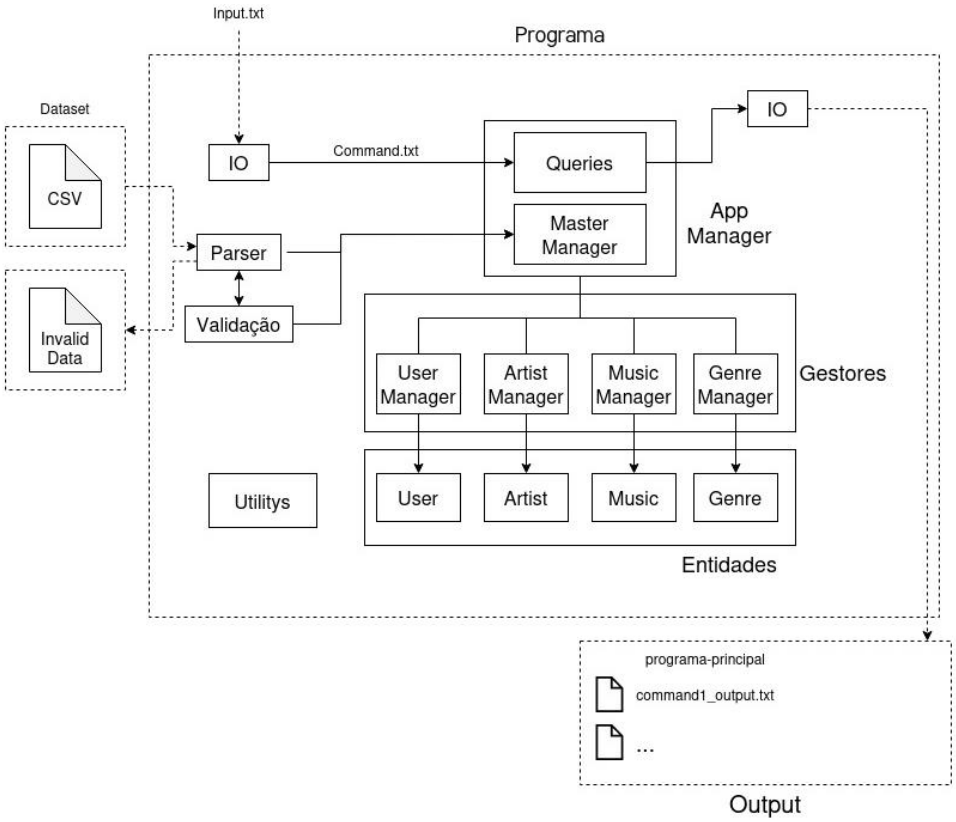
2.1. Mudanças em relação à 1ª Fase

Algumas mudanças tiveram de ser feitas em relação à 1ª fase, desde logo seguindo algumas recomendações deixadas pelos docentes na defesa da 1ª Fase do projeto.

Uma das preocupações dos docentes tinha a ver com encapsulamento. A *Query* 2 utilizava um *array* com as *keys* de uma *HashTable*. Havia a preocupação de que as *keys* pudessem ser alteradas e, consequentemente, apagar um elemento de uma coleção de dados. Desta forma, esta *query* já não aplica este método de funcionamento e usa agora um *array* dinâmico com os *id's* dos artistas, mas que não interfere com a *HashTable* dos mesmos já que, quando cada elemento desta coleção é validado, é guardado no *array* dinâmico e na *HashTable*.

2.2. Arquitetura Aplicada

A arquitetura, naturalmente, também foi alterada, não só pela introdução de novas entidades e respetivos gestores, mas também por questões de melhoramento da modularidade e encapsulamento. Na figura seguinte é apresentado um gráfico da arquitetura aplicada neste projeto.



2.2.1 Estruturas de dados aplicadas

Para a realização das novas queries (e também alteração de queries da 1ª Fase), foi necessária a adição (e modificação) de várias estruturas de dados, de forma a obter melhores resultados no que toca ao tempo de execução do programa. Abaixo, são apresentadas as diferentes estruturas de dados utilizadas em cada entidade.

Músicas - Tabela de *Hash*

Utilizadores – Tabela de *Hash*

Álbuns - Tabela de *Hash*

Históricos - Tabela de *Hash* & *Array* Dinâmico

Géneros - *Array* Dinâmico

Artistas - Tabela de *Hash* & *Array* Dinâmico

2.3. *Queries*

2.3.1. *Query* 1

Inclui-se a *query* 1 neste relatório pois foi ligeiramente alterada em relação à 1ª Fase. Esta pode agora, no input, receber um identificador (*id*) de um artista, além do *id* de um utilizador. Não houve muitas dificuldades a realizar esta *query*, exceto no que tocou ao cálculo da receita de alguns artistas. Tivemos algum tempo a tentar perceber o que poderia ser até que, eventualmente, esta situação se resolveu.

2.3.2. *Query* 4

Apresenta o artista que esteve mais vezes no top 10 num determinado intervalo de datas. No input, pode ou não receber as datas de início e de fim a considerar para esta query. Antes de falarmos sobre a query4 em si, apercebemo-nos que seria necessário percorrer todos os históricos para fazermos as três novas queries. Logo, depois de darmos load ao master_manager, fazemos um processamento dos históricos onde passamos por cada um deles, extraímos a informação necessária para as queries e guardamos nas estruturas apropriadas. O nosso gerenciador de históricos possui os históricos organizados num array que é posteriormente ordenado por data (de mais recente para mais antigo). Neste processamento, guardam-se os artistas da música que foi ouvida e na

própria estrutura do artista é guardado o tempo que o utilizador ouviu o artista. Neste processamento, quando se chega a uma data que passa da semana atual, o array com os id's dos artistas ouvidos é organizado de ordem decrescente do tempo de audição e, depois, será adicionada essa semana à lista ligada presente nos top10 artistas de forma ordenada (mais recente à mais antiga) enquanto se "reseta" o tempo dos artistas presente no array. Depois, retiram-se 7 dias ao dia atual, aumenta-se a semana em 1 e "reseta-se" o array com os id's dos artistas. Na própria função da query 4, basta percorrer todos os artistas somando as semanas que estão dentro do intervalo se este for dado, senão soma-se o número total de semanas de cada artista e vai-se comparando com o maior n de top10 até o momento. Ao terminar, basta dar return aos elementos dos artistas com maior número de top10.

2.3.3. Query 5

Como na query anterior, também aqui é necessário explicar certos passos do processamento de históricos. No processamento de históricos, usando o *id* da música recebida, encontra-se o género da mesma e uma visualização deste género é adicionada a um *array* de inteiros no utilizador, que é criado na hora utilizando o número de géneros existentes, se este ainda não tiver sido criado. A ordem do índice do *array* nos utilizadores é igual à ordem no *array* estático de géneros no gerenciador de género. Depois de adicionar as todas as visualizações, passamos para a função da query 5, onde estes valores do *array* são copiados para uma *matrizClassificacao* onde as linhas representam um utilizador e as colunas um género. Depois só é necessário enviar este *array* e outros argumentos para a função recomendador fornecida pelos docentes e, para terminar, basta escrever no ficheiro de output os *id's* devolvidos por esta função.

2.3.4. Query 6

Apresenta um resumo anual para um utilizador. No input, recebe um *id* de um utilizador, o ano a que se refere o resumo e pode ainda receber um argumento extra (N). Se assim for, apresenta os N artistas mais ouvidos (duração ouvida). Depois de decidirmos como esta query deveria ser feita, não foi muito difícil de a concluir (tirando da equação alguns *SegFaults*). O maior problema que tivemos foi o tempo que esta *query* estava a demorar a executar. Por este motivo, teve que

se alterar a forma como ela processava os históricos de um utilizador. Inicialmente, recebia um *array* com os *id's* de todos os históricos da Tabela de *Hash* dos mesmos. Atualmente, cada utilizador tem um *array* de *id's* de históricos na sua estrutura. Desta forma, na query, basta aceder-se a este *array*, imensamente mais pequeno, e diminui-se o tempo de execução.

2.4. Análise de Desempenho

O programa-testes foi executado cinco (5) vezes tendo sido registados, na tabela infra, os valores médios de cada query e de execução total, assim como o pico de memória utilizada. Os dois dispositivos utilizados foram os seguintes:

• Dispositivo 1 (Lenovo Ideapad 320-15ISK)

• Intel® Core™ i3-6006U @ 2.00GHz | 12,0 GiB RAM DDR4 2133MHz (Ubuntu 22.04.4 LTS)

• Dispositivo 2 (Asus Vivobook 15 Model1502Y)

• AMD Ryzen 7 7730U @ 2.00GHz | 16 GiB RAM DDR4 3200 MHz (Ubuntu 22.04.4 LTS)

	Dataset pequeno		*Dataset* grande	
	Dispositivo 1	Dispositivo 2	Dispositivo 1	Dispositivo 2
Query 1	0,0017106s	0,0008415s	0,0069138s	0,0112408s
Query 2	0,0239086s	0,0029690s	0,0978248s	0,0355294s
Query 3	0,0383200s	0,0001485s	0,0053880s	0,0015676s
Query 4	0,0375752s	0,0114630s	0,3168870s	0,1015734s
Query 5	3,0294988s	1,6268725s	17,709630s	11,7989588s
Query 6	0,0005742s	0,0002690s	0,0148046s	0,0033138
Tempo de Execução Total	31,4226288s	16,05s	212,637s	106,992s
Pico de Memória utilizada	563.744KB	563.232KB	4.239.814KB	4.239.200KB

Tabela 1: Tempo de execução médio das 6 queries, execução total e máximo de memória utilizada

2.5. Otimizações e limitações

Acreditamos que em termos de tempo de execução, obtivemos resultados relativamente bons. No que toca a memória utilizada, não fomos tão felizes, mas também por falta de tempo para podermos melhorar este aspeto. Entendemos

que existiriam várias formas de diminuir este pico de memória como, por exemplo, substituir os tipos de alguns parâmetros em algumas estruturas por tipos que ocupem menos memória, mas que sirvam na mesma o objetivo pretendido.

2.6. Dificuldades sentidas

Algumas das dificuldades foram, ao longo deste relatório, já reportadas. No entanto, a maior dificuldade foi mesmo a memória que o nosso programa ocupa. Logo no início desta 2ª Fase, quando nos foram fornecidos os *dataset* grandes e ainda só tínhamos as primeiras três queries feitas, nos apercebemos de que estávamos perante um problema de memória. Fizemos de tudo para diminuí-la e conseguimos mesmo reduzir em cerca de 1GB comparativamente ao pior momento que tivemos em termos de memória.

3. Conclusão

Finda esta 2ª fase do projeto, podemos dizer que esta unidade curricular nos colocou à prova perante diversos desafios. Acreditamos que os conceitos essenciais desta cadeira, a modularidade e o encapsulamento, ficaram bem entendidos. Além disso, tivemos a oportunidade de descobrir coisas novas acerca da linguagem de programação *C*, assim como as suas variadas bibliotecas, mas principalmente, a *GLib*.

Embora não tenhamos atingido os melhores resultados possíveis, acabamos este projeto sentindo-nos realizados, pois sabemos que sempre demos o nosso todo, ainda que nem sempre tenha sido fácil concretizar certas tarefas.