Universidade do Minho
Escola de Engenharia

Laboratórios de Informática III

Projeto

Segunda Fase

Grupo 25

Tiago Silva Figueiredo, A106856
Duarte Escairo Brandão Reis Silva, A106936
Luís António Peixoto Soares, A106932

1 Índice

2 Introdução

No âmbito da unidade curricular de Laboratórios de Informática III, foi-nos proposto realizar um projeto na linguagem de programação C para consolidarmos as aprendizagens relativas à linguagem, bem como, desenvolvermos novas aptidões, dado que nos foram introduzidas novas técnicas e ferramentas utilizadas na engenharia de *software*, tais como, encapsulamento e modularização, estruturas de dados mais complexas, *Valgrind*, bibliotecas como o *glib 2*.0 e *debugers*.

Para este projeto foi ainda utilizada a ferramenta GitHub para versionamento do código.

Após concluirmos com sucesso a primeira fase do projeto, que consistiu em estruturar a aplicação de *streaming* de música, através da leitura e validação do *dataset* e a resolução de 3 *queries*, começamos a desenvolver a solução que nos permitisse satisfazer todos os requisitos da aplicação, tais como a resolução de todas as 6 *queries*, leitura e validação do novo *dataset*, e a criação de um modo interativo.

Com esta nova fase surgiram novos desafios, já que não era suficiente apenas uma correta implementação das *queries*, foi necessário conjugar também o desempenho geral do programa em termos de tempo de execução e memória utilizada.

Este relatório irá abordar as decisões tomadas e estratégias utilizadas para a realização desta segunda fase do projeto.

3 Sistema

3.1 Estruturas de Dados

Na primeira fase do projeto tivemos em consideração 3 tipos de entidades, *users, musics* e *artists*, sendo que as duas primeiras foram armazenadas em Hash Tables e a última numa lista ligada.

Nesta segunda fase foram introduzidas duas novas entidades, *albums* e *history*, e com esta adição e tendo em conta alguns aspetos discutidos na defesa da primeira fase, decidimos armazenar 4 das entidades em Hash Tables, sendo elas *users, musics, artists* e *albums*. No caso do *history*, decidimos que não seria armazenado por motivos que serão vistos mais à frente. Cada uma destas entidades é representada internamente no programa como uma estrutura, tal como foi visto na primeira fase.

As estruturas das 3 entidades iniciais mantiveram-se praticamente inalteradas, foi apenas necessário acrescentar alguns campos que serviram na sua maioria para a resolução das *queries*.

Quanto aos *albums* e *history*, as suas estruturas são semelhantes às previamente implementadas, contendo campos descritos com *strings (char*)*, *arrays de strings (char**)* e inteiros (int).

Para nos auxiliar na resolução das *queries*, criamos estruturas auxiliares que são preenchidas durante o *parsing* ou então antes do início da *query* onde vão ser necessárias.

3.2 Arquitetura do Sistema

3.2.1 Modularização

Visando obedecer à arquitetura proposta, este projeto foi dividido em vários módulos, estando presentes módulos de entidades, estruturas de dados, gestão do sistema, utilidades e módulos de *I/O*.

Nos módulos de entidades estão definidas todas as estruturas e funções referentes aos *artists, musics, users, albums* e *history* (separadamente), incluindo *getters* e *setters*, que servem, acima de tudo para respeitar o encapsulamento.

Nos módulos de estruturas de dados, estão presentes as estruturas utilizadas para armazenar as informações de todas as entidades (separadamente), algumas delas já referidas em 3.1, bem como as funções que lidam diretamente com essas mesmas estruturas, como funções *add, search* e *free*.

Nos módulos de gestão de sistema são definidas funções que permitem gerir todas as estruturas de dados de todas as entidades.

No módulo de utilidades estão definidas funções usadas principalmente na validação do *dataset*.

Nos módulos de *I/O* está definida a função *parser* que trabalha diretamente com os ficheiros CSV, e está encarregue de colocar as informações das entidades nos seus devidos locais.

3.2.2 Diagrama

A seguir segue-se um diagrama que tem como objetivo ilustrar a arquitetura do projeto.

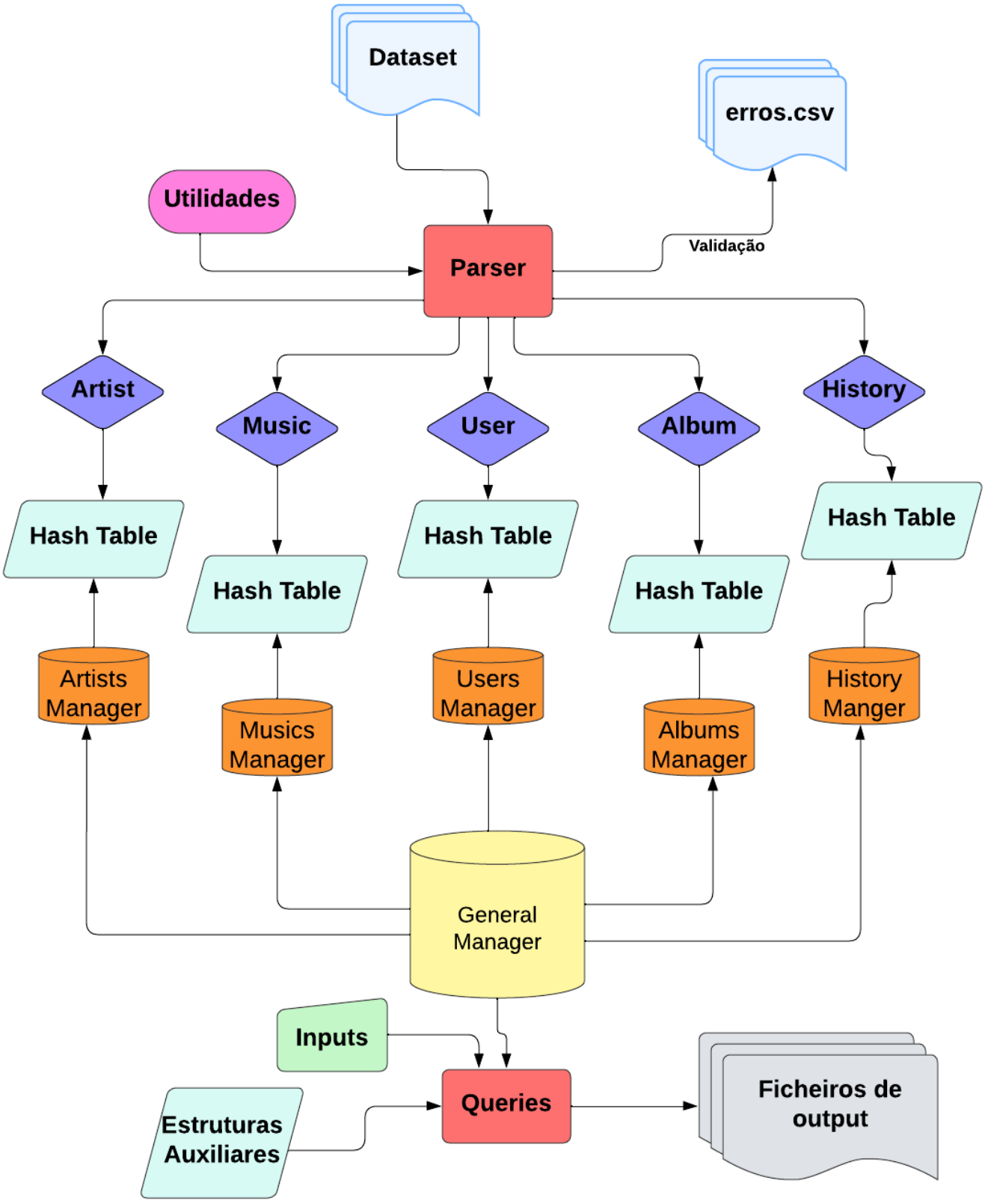

Figura 1: Diagrama da arquitetura do projeto

4 Discussão

4.1 Resolução das Queries

4.1.1 Query 1

Na *query* 1 foi mantida a estratégia implementada, apenas foram alterados os formatos de *output* possíveis, e acrescentada a possibilidade de termos um resumo, mas desta vez relativa a um artista.

4.1.2 Query 2

Na *query* 2 foi mantida a estratégia implementada, apenas foi alterado os formatos de *output* possíveis. Houve também uma alteração na estrutura auxiliar relativa a essa *query*, pois passou de uma lista ligada para uma Hash Table, o que levou a uma significativa redução de tempo de execução dessa mesma *query*.

4.1.3 Query 3

Na *query* 3 foi mantida a estratégia implementada, apenas foi alterado os formatos de *output* possíveis.

4.1.4 Query 4

A *query* 4 pede para indicar qual é o artista que esteve mais vezes no top 10, podendo ser também passado como argumento datas que restringem as semanas que devem ser tidas em consideração.

Para a resolução desta *query* usamos duas estruturas auxiliares, duas Hash Tables, uma delas que guarda todos os top 10 relativos a uma determinada semana, e outra que guarda os artistas que estiveram no top 10 entre o intervalo de datas, juntamente com o número de vezes que estiveram no top 10, sendo esta tabela preenchida a partir da primeira.

Com a segunda tabela preenchida basta apenas descobrir qual o artista que esteve mais vezes no top 10.

4.1.5 Query 5

Na *query* 5 é pedido para listar N utilizadores que tenham um gosto semelhante a um utilizador passado como argumento.

Para a resolução desta *query* utilizamos o recomendador disponibilizado pela equipa docente. Esse recomendador tinha uma série de argumentos, mas os mais relevantes eram uma matriz de inteiros que representava os gostos de cada utilizador, e um array de id's de utilizadores, devidamente alinhado com a matriz de gostos.

Estes 2 parâmetros são preenchidos antes da respetiva *query* e são passados como argumento.

O recomendador devolve a lista de utilizadores com gostos semelhantes, sendo apenas necessário colocar essa informação no ficheiro de *output*.

4.1.6 Query 6

A *query* 6 pede um resumo anual para um dado utilizador passado como argumento. Pode também ser pedido para listar os N artistas mais ouvidos nesse ano.

Na resolução desta *query* foi usada uma estrutura auxiliar, contudo cada *input* implicou a criação de outras estruturas específicas.

Para cada utilizador passado como *input* são criadas 4 Hash Tables que têm como objetivo calcular o número de músicas mais ouvidas, o artista mais ouvido, o género de música mais ouvido e o álbum favorito.

Outros valores necessários para a resposta à *query* não necessitam de estruturas auxiliares e são calculados dentro da própria *query.*

Caso o argumento opcional esteja presente no *input* basta-nos aceder a uma das Hash Tables criadas especificamente, já que, independentemente desses valores serem necessários, eles são já previamente calculados sem prejudicar o desempenho da *query.*

Com todas as informações necessárias apenas temos de as preparar para serem colocadas no ficheiro de *output*.

4.2 Análise de Desempenho

A seguir é apresentamos uma tabela que contém o tempo e memória média obtidas depois de 5 execuções do programa de testes em cada uma das máquinas dos elementos do grupo.

Os resultados de cada máquina são influenciados devido aos componentes de cada uma delas, daí estarem expressas na tabela as diferenças de *hardware* entre cada uma das máquinas.

Máquina	1	2	3
S.O.	Pop!_OS	Ubuntu	Ubuntu
CPU	AMD Ryzen 7 5800H	Intel i5-13500H Processor 4.7 GHz	Intel® Core™ i7-7700HQ (Octa-core)
RAM	16GB	16GB	8GB
Tempo para o dataset pequeno			
Query 1	0.001745s	0.000429s	0.001836s
Query 2	0.067123s	0.069378s	0.091685s
Query 3	0.001187s	0.000660s	0.002769s
Query 4	0.406406s	0.303192s	0.837250s
Query 5	2.883776s	2.422850s	4.554590s
Query 6	0.000447s	0.006304s	0.001019s
Total:	19.674s	17.863s	26.539s
Memória (MB)	0.885	0.887	0.886
Tempo para o dataset grande			
Query 1	0.003985s	0.001733s	0.002341s
Query 2	0.852429s	0.823092s	0.841362s
Query 3	0.002609s	0.001631s	0.002159s
Query 4	2.305925s	2.660522s	2.450189s
Query 5	42.244109s	37.964147s	39.761201s
Query 6	0.005520s	0.005430s	0.005501s
Total:	189.834s	168.609s	172.564s
Memória (MB)	5479	5477	5481

4.3 Encapsulamento

Com o objetivo de respeitar o encapsulamento, tivemos desde o primeiro momento cuidados relativos a essa matéria. Desde o início criamos sempre estruturas opacas estando nos ficheiros **.h** apenas os *typedef* das respetivas estruturas.

Para assegurarmos que os dados ficavam inalterados, recorremos à duplicação da memória dos campos sempre que for necessário o acesso aos mesmos. Através de *getters e setters* conseguimos respeitar o encapsulamento em todas as estruturas.

Foram definidas funções que devolviam cópias das entidades sempre que era necessário fazer uma procura (*search*) na estrutura onde a entidade estava armazenada.

4.4 Programa de Testes

Como auxílio no desenvolvimento do projeto, foi criado, já na primeira fase, e devidamente atualizado na segunda, um programa de testes que verifica se os *outputs* da nossa resolução das *queries* está de acordo com os *outputs* esperados.

Este programa de testes revelou-se extremamente útil, já que nos dá o *feedback* do tempo de execução e a memória máxima utilizada, duas componentes essenciais para um bom desenvolvimento desta segunda fase.

4.5 Programa Interativo

O programa interativo implementado nesta segunda fase do projeto dá a liberdade para o utilizador do programa escolher qual *query* quer executar, sendo uma experiência mais agradável para o utilizador.

Este modo foi pensado para ser intuitivo para quem o utiliza, sendo bem claro em cada passo o que o utilizador necessita de fornecer ao programa para o normal funcionamento do mesmo.

Como durante a leitura do *dataset* é feita a validação do mesmo, é garantido ao utilizador que quando um *input* para uma qualquer *query* seja inválido, ele será informado do erro, tornando esta uma interação mais fiável.

5 Conclusão

5.1 Dificuldades sentidas

Tal como é esperado, esta segunda fase trouxe as suas adversidades.

Os dois aspetos mais desafiantes foram manter o encapsulamento entre os módulos e o desempenho do programa.

Inicialmente o programa apresentava muitas deficiências em questões de encapsulamento, já que tínhamos vários acessos indevidos a estruturas onde estão guardados dados que não podem ser alterados externamente.

Contudo após adotarmos medidas que visam respeitar o encapsulamento (já descritas em 4.3) conseguimos manter o programa dentro do que é espectável a nível de encapsulamento.

Quando ao desempenho, as dificuldades viram-se principalmente em manter o tempo de execução abaixo de 10 minutos, e a memória utilizada abaixo dos 5000 MB, isto relativo ao *dataset* grande.

Esta dificuldade foi ultrapassada quando percebemos que o armazenamento das informações do *history* aumentava drasticamente a nossa memória utilizada, daí termos optado por não armazenar o histórico por completo, mas sim as estruturas auxiliares que dele dependiam.

Para uma melhor gestão da memória optamos também por inicializar as estruturas auxiliares à resolução das *queries* no momento em que elas vão começar a ser executadas, e libertamos a memória alocada para as mesmas no final da execução completa de cada *query*.

5.2 Aprendizagens

Com a colusão desta segunda fase, e do projeto em geral, sentimos uma grande evolução no conhecimento da linguagem C, e outras ferramentas utilizadas ao longo de todo o projeto.

Saímos com uma visão diferente relativamente ao que é trabalhar num projeto de maior dimensão que implica não apenas uma correta implementação das soluções para os problemas que vão surgindo, mas tem em conta também o desempenho computacional, como peça chave para uma boa fluidez do projeto em si.