LI3 - Relatório da Fase I - Grupo 32

João Delgado - A106836 Simão Mendes - A106928
Pedro Pereira - A107327

9 de novembro de 2024

Resumo

Este relatório descreve o desenvolvimento da Fase I do projeto LI3, cujo objetivo foi implementar uma aplicação para o *parsing* de ficheiros CSV relacionados com os componentes de uma plataforma de música e a execução de *queries* sobre esses ficheiros. Nesta fase, os dados extraídos foram convertidos em estruturas de dados adequadas, permitindo a execução de *queries* específicas no *dataset*, bem como a integração de informações para responder a *queries* mais complexas. Os resultados obtidos estabelecem uma base sólida para as próximas fases do projeto, possibilitando uma análise aprofundada e um processamento eficiente dos dados fornecidos.

Conteúdo

Capítulo 1

Introdução

O presente relatório tem como objetivo descrever o trabalho desenvolvido na Fase I do projeto de LI3, que consiste na criação de uma aplicação para o *parsing* de ficheiros CSV relacionados a componentes de uma plataforma de música. Este projeto visa não apenas a extração e manipulação de dados, mas também a exploração de funcionalidades que permitem uma melhor interação com a informação musical.

Os principais objetivos deste trabalho prático incluem a implementação de um sistema que permita a **leitura eficiente** de dados a partir de ficheiros CSV, a realização de *queries* para obter informações relevantes e a apresentação de **resultados** de forma clara e concisa. A aplicação foi projetada para ser modular, facilitando a manutenção e a expansão futura do sistema, além de ter em conta a usabilidade e a **eficiência** no processamento dos dados.

Dentre as características mais relevantes do programa, destacam-se a capacidade de realizar *queries* complexas que inter-relacionam diferentes conjuntos de dados e a implementação de técnicas de **encapsulamento** que garantem uma **arquitetura limpa e organizada**. Este relatório apresenta, de forma detalhada, as escolhas de design feitas pelo grupo, os resultados obtidos durante o desenvolvimento e as reflexões acerca do processo de implementação.

Capítulo 2

Sistema

2.1 Arquitetura

A seguir, é apresentado um diagrama que ilustra, de forma simplificada, os principais módulos funcionais do programa. Este diagrama demonstra as interações entre os módulos e descreve o fluxo de dados desde o input até ao output, abrangendo ambos os componentes desenvolvidos nesta fase do projeto.

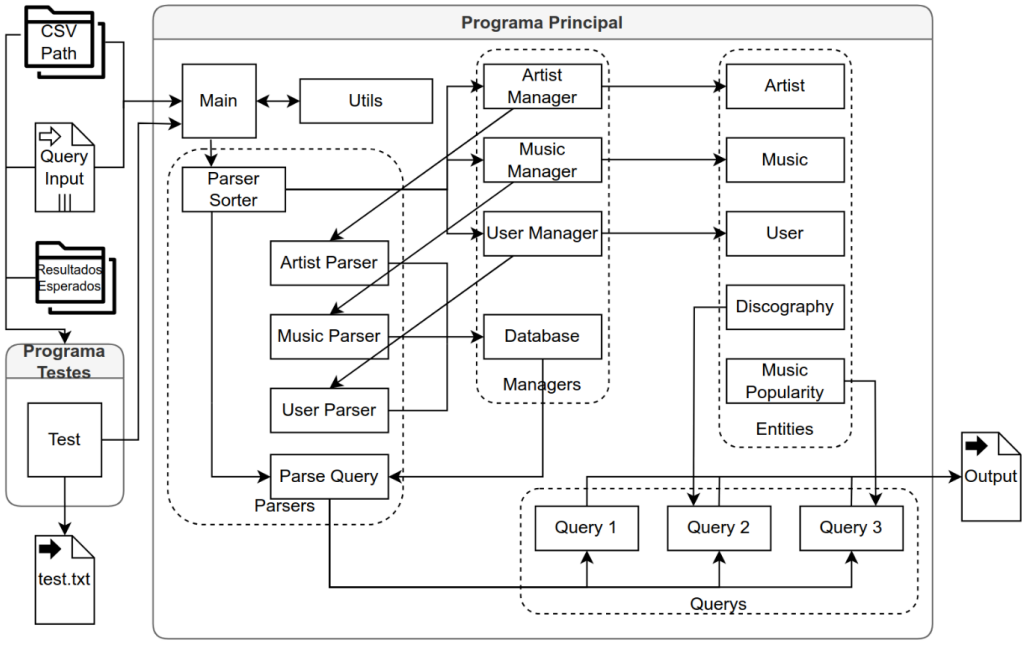

Figura 1: Arquitetura geral do projeto

2.1.1 Programa-Principal

O programa principal está estruturado de modo a que os parâmetros de entrada sejam os seguintes:

```
./programa-principal  <dataset-path> <queries-input-path>
```

Após a introdução dos parâmetros, o programa inicia a execução de acordo com o diagrama apresentado anteriormente. Com a análise do diagrama podemos concluir que o sistema está estruturado em várias camadas:

1. **Gestores:** Módulos responsáveis pela gestão das entidades. Solicitam o *parse* de dados da respetiva entidade e são responsáveis por toda a criação e libertação das estruturas de dados que armazenam os conjuntos das mesmas. Nestes, existem as tabelas *hash* dos três dados principais: **artistas**, **músicas**, **utilizadores**. Além disso, constroem a **database**, que consiste essencialmente num conjunto das três tabelas *hash* correspondentes às estruturas anteriores.

2. **Parsers:** Incluem componentes dedicados ao *parsing* de dados. Estes módulos são compostos por três ficheiros responsáveis pela análise das principais estruturas: **artistas** (*Artist Parser*), **músicas** (*Music Parser*) e **utilizadores** (*User Parser*). O *Parse Query* é encarregado de ler o ficheiro de entrada e de executar cada uma das *queries* de forma isolada, utilizando os dados processados pelos *parsers* mencionados anteriormente. Por fim, o *Parser Sorter* interliga todos os *managers*, que por sua vez estão conectados aos *parsers*, e solicita ao *database manager* a criação da estrutura de dados denominada **database**.

3. **Entidades:** As estruturas de dados constituem o núcleo do projeto, sendo responsáveis pelo armazenamento, em formato pré-definido, dos campos extraídos pelos módulos de *parsing* dos ficheiros CSV de entrada. As três principais estruturas são **artista** (*artist*), **música** (*music*) e **utilizador** (*user*), as quais agregam os dados que consideramos essenciais para a execução desta fase inicial do projeto. É relevante notar que, por motivos de gestão de memória, optámos por não armazenar determinados dados, como as letras das músicas e as descrições dos artistas, uma vez que estes não são necessários para a realização das *queries*, por enquanto. As três entidades mencionadas podem ser consultadas no final desta subsecção. Além destas estruturas principais, utilizamos algumas estruturas auxiliares para fins de execução das *queries*, sendo elas a **discografia** (*discography*) e a **popularidade da música** (*music popularity*). Discutiremos estes aspetos com mais detalhe na subsecção dedicada às *queries*.

4. **Queries:** Após o *parsing*, cada *query* é executada de forma semi-isolada, em módulos independentes. Em geral, estas aproveitam estruturas já definidas na biblioteca *glib* para facilitar a sua execução. Na secção seguinte, abordaremos individualmente cada uma delas.

5. **Utilidades:** Definimos alguns módulos utilitários, que possuem responsabilidade desde a validação sintática e lógica até módulos com funções que auxiliam na conversão de dados para a biblioteca *glib*. Incluímos também funções destinadas à manipulação de *strings*, com o objetivo de facilitar o *parse* de dados e a execução de *queries*.

6. **Output:** Módulos responsáveis pela geração de *output* para linhas que não sejam válidas conforme os requisitos predefinidos da validação sintática e lógica, bem como para a saída de cada *query*, de forma individual. Cada *query* possui um módulo de saída específico. Os *parsers* partilham um módulo de saída geral para registar os ficheiros de erro.

Entidades Principais		
Artista	**Música**	**Utilizador**
Identificador Nome Membros (grupo) País Tipo	Identificador Título Artista(s) Duração Género	Identificador Email Nome Sobrenome Data nascimento País Tipo subscrição Músicas favoritas

Database
Artistas Músicas Utilizadores

Figura 2: Estruturas de dados principais

Os módulos foram cuidadosamente organizados de modo a permitir o isolamento dos dados específicos de cada um. Esta abordagem facilita a manutenção e a extensibilidade do sistema, uma vez que cada módulo pode ser desenvolvido, testado e modificado independentemente. Existem vários módulos que são implementados em função da respetiva estrutura de dados, sendo de certa forma semelhantes, como é o caso do ecossistema *gestor-parser-entidade*, que pode ser observado algumas vezes ao longo do projeto.

2.1.2 Programa-Testes

O programa de testes, funciona de forma muito similar ao principal, utilizando da maioria dos módulos do mesmo, no entanto os seus parâmetros de entrada, são ligeiramente diferentes, sendo estes os seguintes:

```
./programa-testes <dataset-path> <queries-input-path> <output-folder-path>
```

Além disso, redefinimos algumas das funções originais com o propósito de registar o tempo de execução de cada *query*. O *output* gerado pelo programa de testes apresenta os seguintes campos: a **execução** de cada *query*, incluindo discrepâncias ou a execução correta; o **tempo médio de execução** para cada tipo de *query*; a **quantidade de *queries* corretas** em cada categoria; o **tempo total de execução** do programa; e **a memória total utilizada**.

2.2 Queries

Nesta secção, iremos detalhar a implementação de cada uma das três *queries* propostas nesta fase do trabalho. O nosso objetivo foi **maximizar** a **eficiência** e **minimizar** o custo de **memória**, o que acreditamos ter alcançado. As três *queries* em questão são as seguintes:

1. **Listar o resumo de um utilizador, consoante o identificador recebido por argumento:** Esta *query* é extremamente simples. A sua execução consiste apenas em procurar o identificador do utilizador na tabela *hash* de utilizadores e obter os seus campos, que são então impressos num ficheiro de *output* na ordem solicitada.

2. **Listar os Top N artistas com maior discografia:** Para esta *query*, foi criada uma estrutura de dados auxiliar, **discografia** (*discography*), que armazena a duração total das músicas e o identificador do artista. A *query* apenas percorre a tabela *hash* de músicas uma única vez, permitindo que execuções subsequentes utilizem os resultados anterior. Também foi considerado o parâmetro adicional de **país**, que restringe a seleção dos artistas. Finalmente, os dados foram organizados numa lista ligada com uma função da *GLib*, e apenas os N primeiros artistas foram apresentados.

3. **Listar géneros de música mais populares numa faixa etária:** Esta *query* revelou-se relativamente simples após a realização da segunda, uma vez que apenas era necessário aplicar os mesmos princípios. Assim como na anterior, criámos uma estrutura de dados denominada **popularidade de música** (*music popularity*), na qual armazenámos

cada género e os respetivos "favoritos". Utilizámos, desta vez, um vetor, no qual o índice representava a idade do utilizador e o valor era uma lista ligada da estrutura **popularidade de música**. Novamente, conseguimos apenas percorrer a tabela *hash* de utilizadores apenas uma vez. Por fim, numa lista ligada, juntámos todos os elementos do vetor entre as idades solicitadas e, utilizando uma função da própria *GLib*, ordenámos o mesmo conforme os parâmetros necessários. Como sempre, imprimimos nos ficheiros de *output* o resultado esperado.

De seguida, apresentamos a distribuição das estruturas de dados auxiliares utilizadas pelas duas últimas *queries*:

Discografia	Popularidade de Música
Identificador Soma duração músicas	Género musical Número de Gostos

Figura 3: Estruturas de dados *queries*

2.3 Makefile e Documentação

O **Makefile** criado para este projeto é eficiente e de fácil utilização, com algumas *flags* importantes para otimização, como a **-O3** para uma compilação mais rápida e eficiente, e a **-march=native**, que permite ao compilador ajustar o código especificamente para a arquitetura da máquina onde está a ser compilado. Além disso, integramos as *flags* necessárias para a biblioteca **GLib**, o que garante que o programa utiliza adequadamente as suas funcionalidades.

Para manter uma estrutura organizada, os **arquivos objeto** (.o) para os dois modos do programa são armazenados em pastas próprias dentro da pasta resultados. Assim, podemos compilar o programa com o comando **make**, e remover todos os arquivos de compilação e de execução com **make clean**.

Adicionalmente, para a documentação do projeto, utilizamos o gerador **Doxygen**, que facilita a criação de uma documentação detalhada e completa. Com **make doc**, o Doxygen gera uma documentação que descreve todo o código-fonte, incluindo detalhes como parâmetros de funções, valores de retorno, e descrições básicas de cada função individualmente. Esta documentação é essencial para uma compreensão clara e rápida do funcionamento do código, servindo de referência para manutenção e futuras expansões do projeto. Se necessário, a documentação pode ser removida com o comando **make docclean**.

Capítulo 3

Discussão

3.1 Tempo de Execução

De forma sintética utilizamos duas grandes estruturas de dados para a organização do projeto:

- Tabelas de *Hash* (*GHashTables*)
- Listas ligadas simples (*GSLists*)

As tabelas de ***hash*** desempenharam a função de armazenar todas as estruturas principais utilizadas ao longo do projeto, proporcionando benefícios significativos, como o acesso em tempo constante **O(1)** às estruturas de dados, o que favoreceu operações como inserções e buscas. A limitação desta estrutura de dados seria a possibilidade de colisões, contudo, desconhecemos em detalhe a forma como a *GLib* gere esses casos.

Adicionalmente, recorremos a **listas ligadas simples**, concebidas com o objetivo de mitigar o impacto das *queries* que processam grandes volumes de dados, como acontece nas *queries* dois e três. Estas listas foram também empregues para armazenar dados em formato de lista dentro de outras estruturas, como as músicas favoritas dos utilizadores e os artistas associados a cada música. Como sempre, esta estrutura de dados apresenta algumas desvantagens, como um pequeno atraso na velocidade geral do programa devido à adição de nós, uma vez que cada nó precisa de ser criado individualmente.

Gostaríamos de destacar um problema que enfrentámos anteriormente. Primeiramente, estávamos a percorrer as tabelas de *hash* uma vez para cada *query* (exceto a um), o que resultava numa significativa degradação do desempenho do programa. Felizmente, conseguimos reestruturar o funcionamento de todas as *queries* de forma a que a operação fosse realizada apenas uma vez,

na primeira ocorrência de cada tipo de *query*, permitindo que os resultados obtidos fossem reutilizados nas ocorrências subsequentes.

3.2 Memória

Em termos de utilização de memória, acreditamos ter cumprido os objetivos do trabalho, uma vez que utilizámos estruturas que visavam minimizar o consumo de memória. Isso incluiu armazenar apenas os dados absolutamente necessários e gerir cuidadosamente a libertação de memória. Praticamente toda a memória utilizada resulta do *parsing* dos dados, sendo que a alocação de memória pelas *queries* é praticamente insignificante em termos de consumo simultâneo de memória.

Ainda assim, consideramos que outras abordagens poderiam ser exploradas para evitar potenciais problemas, como o uso de estruturas de dados mais "leves" do que as listas ligadas.

3.3 Análise de Performance

Como o texto isolado não reflete de forma precisa o desempenho de um programa, apresentamos a seguir uma tabela que mostra a execução do mesmo em diferentes dispositivos, incluindo algumas métricas que consideramos relevantes. As médias aqui apresentadas consideram dez testes, cinco dos *datasets* com erros e cinco dos sem erros.

Dispositivo	CPU	Memória	OS	Compilador
Zenbook 15 OLED (UM3504)	AMD Ryzen™ 7 7735U	16 GB LPDDR5 6400 MT/s	Manjaro 24.1.2	GCC 14.2.1
Vivobook 16 (F1605, 12th Gen Intel)	Intel® Core™ i7-1255U	16 GB DDR4 3200 MT/s	Ubuntu 24.04.1 LTS	GCC 13.2.0
Raspberry PI 5 8GB	ARM Broadcom BCM2712	8 GB LPDDR4X 4267 MT/s	Raspberry Pi OS 12	GCC 12.2.0

Dispositivo	Texec Q1 (s)	Texec Q2 (s)	Texec Q3 (s)	Total Memory (MB)	Texec Total (s)
Zenbook 15 OLED (UM3504)	0,00002	0,00269	0,09259	591,48	5,33
Vivobook 16 (F1605, 12th Gen Intel)	0,00001	0,00214	0,07512	591,79	3,91
Raspberry PI 5 8GB	0,00002	0,00463	0,15588	502,45	9,64

Figura 4: *Benchmark* do programa em alguns dispositivos do grupo

Capítulo 4

Conclusão

De modo geral, acreditamos que esta fase do projeto foi concluída com sucesso, apesar dos desafios e dos problemas específicos que enfrentámos, conforme discutido no capítulo anterior. Pensamos que o nosso código estará **bem estruturado**, **modular** e alinhar-se-á com os princípios fundamentais de desenvolvimento, como o **encapsulamento**, a clareza e a organização, tornando-o não só funcional, mas também de **fácil manutenção e compreensão** para futuras fases do projeto.

No decorrer do projeto, o nosso grupo não só aprimorou as suas habilidades técnicas, como também desenvolveu uma abordagem prática para resolver problemas e contornar dificuldades, o que será certamente útil na continuação deste trabalho.

Com uma visão clara e objetiva para a segunda fase, sentimos que estamos bem preparados para enfrentar os novos desafios e exigências que surgirem. Estamos confiantes de que, com o que já alcançámos, conseguiremos cumprir todos os objetivos desta unidade curricular de Laboratórios de Informática III.