



Projeto LI3 24/25

Relatório 1ª Fase

Gabriel Dantas, a107291

José Fernandes, a106937

José Martins, a104443

Simão Oliveira, a107322

Grupo 15

Engenharia Informática
Universidade do Minho
2024/2025

Índice

Índice.....	1
Introdução.....	2
Objetivo.....	3
Sistema.....	4
Arquitetura da Aplicação.....	4
Funcionamento da Aplicação.....	5
Discussão.....	7
Modularização.....	7
Estruturas de dados.....	7
Eficiência.....	9
Conclusão.....	11
Aspetos a melhorar.....	11

Introdução

O presente relatório adjunta o projeto que se realiza no âmbito da unidade curricular de Laboratórios de Informática III do 2º ano de Engenharia Informática, na Universidade do Minho.

O projeto foi desenvolvido em grupo, com colaboração via *GitHub*, elaborado na linguagem de programação C.

A gama de conhecimentos explorados no decorrer da composição deste, são:

- Modularidade
- Encapsulamento
- Estruturas dinâmicas de Dados
- Validação funcional
- Medição de desempenho

A estrutura básica do repositório do projeto mostra-se abaixo:

```
grupo 15/  
  | README.md  
  | trabalho-prático/  
    | Makefile  
    | include/  
      | ...  
    | src/  
      | ...  
    | resultados/  
      | ...  
    | resultados-esperados/  
    | programa-principal (após dar make)  
    | programa-testes    (após dar make)  
    | relatório-fase1.pdf  
    | relatório-fase2.pdf (apenas na 2ª fase de entrega)
```

Objetivo

Tem-se por objetivo o desenvolvimento de um projeto final, escrito em C, que consista na implementação de uma base de dados em memória a partir de ficheiros .csv, fornecidos pelos docentes, que sumarizam informações relativas a um sistema de streaming de música. Estes dados serão armazenados e implementar-se-ão métodos de pesquisa, o mais eficientes possíveis, para atender às *queries* impostas pela chamada do programa.

O projeto final instrui aos discentes conceitos de modularidade, encapsulamento, abstração e reutilização de código, usabilidade e eficácia de estruturas dinâmicas de dados, validação do código funcional e capacidade de medir o desempenho do software. Com esta proposta criar-se-á uma base sólida de conhecimentos e habilidades no que diz respeito à engenharia de software.

Nesta 1ª fase agora completa, foi necessário desenvolver a etapa de *parsing* e a validação dos dados de entrada, desenvolver os modos executáveis principal e de testes e realizar 3 *queries* propostas pelos docentes.

Sistema

Arquitetura da Aplicação

A arquitetura do projeto, representada no Diagrama 1, foi pensada tendo em conta uma organização modular, visando sempre a abstração e opacificação dos módulos criados, bem como o fácil gerenciamento e transmissão de informação entre os mesmos.

Utilizou-se uma estrutura denominada *Database* como um gestor de nível superior para interagir com os gestores das entidades (considerando-se as entidades os tipos de dados criados diretamente da informação fornecida nos .csv), estes chamados de *Entity Catalog*.

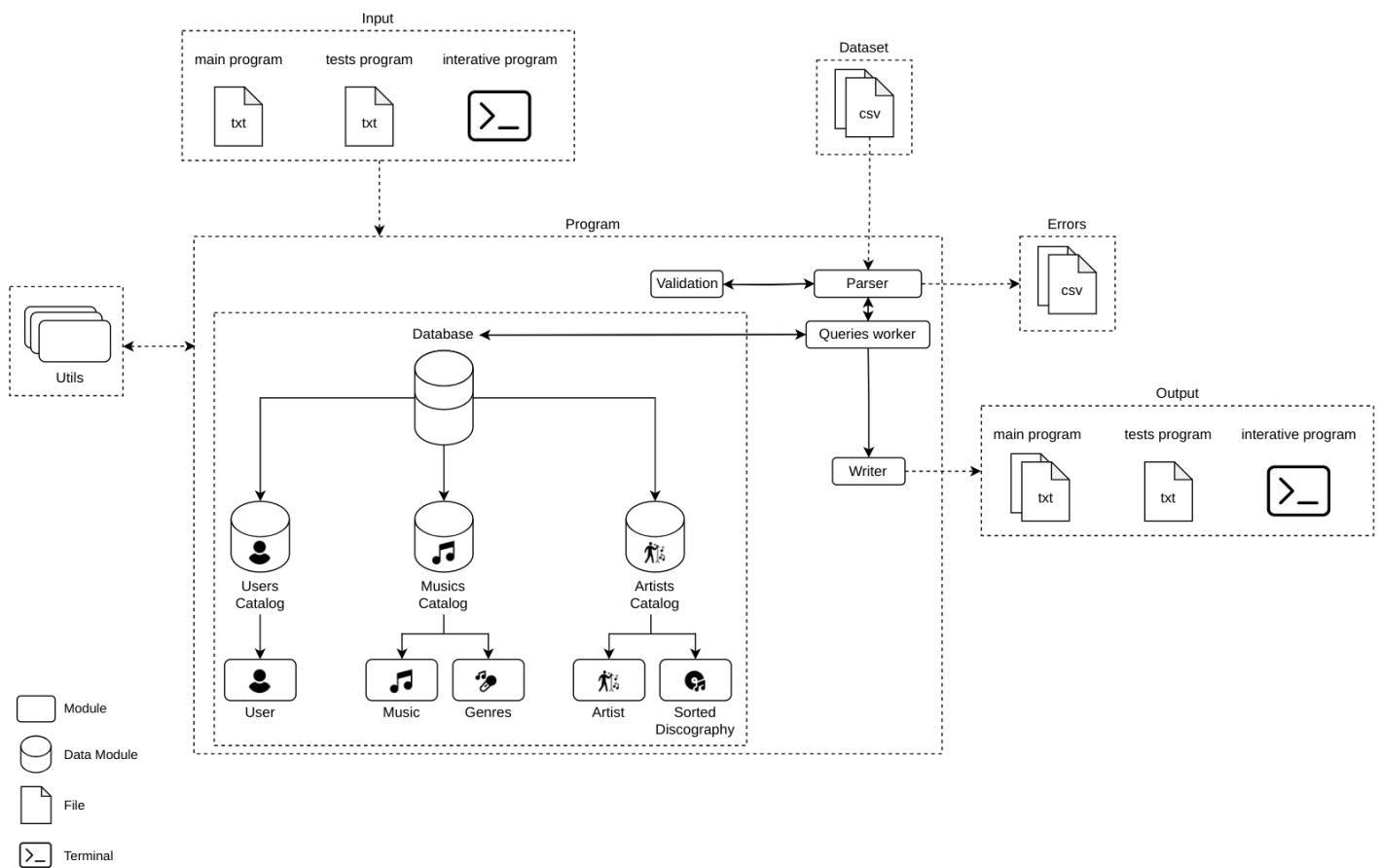


Diagrama 1. Estrutura da Arquitetura

Funcionamento da Aplicação

A aplicação inicia-se com a chamada, via terminal, através de um número diferente de argumentos, dependendo do modo em que esta se deseja utilizar (programa-principal ou programa-testes, excetua-se neste 1º relatório a análise do programa-iterativo, visto o seu desenvolvimento pertencer à 2ª fase do projeto).


De início são criados a *DataBase* e os catálogos. Os catálogos são os gestores das entidades do sistema, sendo campos do *Database*, um gestor superior que interage com estes catálogos havendo requerimento de informação devolvida por estes. Os catálogos são implementados através de *hashtables*, com intuito de eficiência na procura direta dos dados, utilizando-se os *id* 's (ou propriedade mais semelhante a um identificador unitário) como chave e a estrutura a ela relativa como valor.

De seguida os arquivos do *Dataset* são lidos (através do 1º argumento de chamada do programa) por um parser genérico e é feito o seu tratamento. As informações inválidas são colocadas em arquivos de erros, já os restantes populam a *Database*, ou seja, são convertidos em estruturas de dados de entidades e adicionados aos respectivos catálogos.

São ainda criadas duas estruturas de dados auxiliares, já tencionando a eficiência à resposta das *queries*, uma denominada *genres* no catálogo das músicas e outra intitulada *sorted discography array* no catálogo dos artistas. Estas estruturas serão elucidadas na seção de discussão do projeto.

Este tratamento inicial permite que a resposta às *queries*, posteriormente, geridas pelo *queries worker*, seja mais eficiente, visto os dados já estarem organizados. O *queries worker* requer a informação ao *DataBase* e este, por sua vez, interage com os catálogos para conseguir fornecer a resposta desejada usando funções de listagem e obtenção de dados disponibilizadas pelos catálogos.

Referem-se ainda brevemente os *utils*, um conjunto de módulos de pequeno porte que são usados pelos restantes módulos. Estes são módulos simples, que abstraem código de modo a poder ser reutilizado e tratam informação de mais baixo nível, como, por exemplo, strings de datas e durações de músicas, de modo a poderem



ser usados por outros módulos, neste caso concreto, pelo módulo de validação de dados.

O *Writer* é um módulo de escrita genérico que tem a função de criar e escrever nos ficheiros de output após o sistema obter a resposta a cada *query*. As *queries* escrevem as respostas num *buffer*, sendo este fornecido ao *Writer* logo de seguida.

No final da execução é feita uma limpeza da memória, onde as estruturas criadas são eliminadas, impedindo que sucedam *memory leaks* no momento terminal da execução do programa.

Discussão

Esta seção tem por objetivo debater os métodos de modularização e encapsulamento utilizados, bem como a escolha das estruturas de dados para realizar eficazmente as *queries*. Note-se que esta seção terá uma forte incidência nas técnicas de modularização neste 1º relatório, visto que o encapsulamento e o debate aprofundado das decisões tomadas na escolha das estruturas de dados são alvo de avaliação apenas no prosseguimento do projeto.

Modularização

Para modularizar o código foram usados diversos métodos estudados. Distinguem-se a declaração e a implementação das funções e estruturas de dados, separadamente, através dos cabeçalhos e módulos de implementação, respetivamente. As estruturas de dados foram implementadas em escopo local e tornadas opacas através da sua declaração por *typedef*, desta forma os seus campos apenas podem ser acedidos pelo conjunto de funções com tal intuito, os *getters* e os *setters*.


Os módulos que necessitam de funções externas apenas num escopo local tiveram as suas importações feitas somente nos arquivos de implementação.

A separação dos módulos foi pensada tentando sempre manter os módulos com um objetivo conciso e simples de perceber, porém criando submódulos quando havia a sensação da complexidade dos tais começar a tornar-se elevada, criando problemas de organização e leitura do código.

Estruturas de dados

Para responder às *queries* utilizaram-se estruturas de dados mais específicas.

A *query* 1, dado buscar eficiência na procura das informações de um utilizador através do seu *id*, é bastante veloz utilizando o próprio catálogo dos utilizadores, visto este ter sido implementado, como já referido, por uma *hashtable* onde o *id* do utilizador corresponde à chave deste.



A *query* 2, deseja saber quais são os N (número arbitrário passado como argumento na chamada da *query*) artistas com um maior valor de discografia (soma dos tempos de duração de todas as músicas correspondentes a esse artista). Para executar esta *query* em um espaço de tempo curto, fez-se um pré-tratamento, criando uma estrutura de dados nova chamada *Sorted Discography Array*.

Esta estrutura é uma lista ligada que guarda, em cada nodo, o *id* de um artista e informação relevante para responder à *query* (nome, tipo de artista, tempo total de discografia e país) que é ordenada decrescentemente pelo parâmetro duração da discografia total do artista.

Durante o *parsing*, quando se percorrem os *tokens* relativos às músicas para criar o catálogo das músicas, simultaneamente se opera com fim a esta estrutura. Para cada nova música, converte-se a sua duração em um inteiro que contabiliza os segundos e esta é somada a um campo do catálogo dos artistas, chamado *discography*, no respetivo artista, procurado pelo *id*.

Após este processo, itera-se sobre o catálogo dos artistas e é criada a estrutura com as informações relevantes para a *query*, e adicionada, de forma ordenada, à lista ligada.

Assim, a pesquisa nada mais é do que devolver diretamente as informações dos N primeiros artistas desta lista. Caso seja passado um argumento que restringe a procura para artistas de um país específico, basta fazer esta procura de forma seletiva, comparando o país passado com o país do artista e fazendo uma contagem até chegar a N artistas, ou terminar de percorrer a lista ligada.

A *query* 3 procura descobrir quais os géneros de música mais populares com base numa faixa etária. Para tentar minimizar o tempo das operações sempre que esta *query* é chamada, resolveu-se optar por uma *hashtable*, que foi chamada de *genre*, onde a chave corresponde ao nome dos géneros e o valor contém uma lista ligada.

Nesta lista ligada, cada nodo guarda a informação, em forma de uma estrutura simples com 2 inteiros, da idade e dos likes acumulados respetivos ao género da chave nesta respetiva idade. Esta lista ligada é ordenada decrescentemente pela idade.

A *hashtable genre* é criada percorrendo utilizador por utilizador, para cada utilizador é percorrida a lista das músicas marcadas com gosto e, para cada música

desta lista é procurado pelo seu *id*, no catálogo das músicas, o seu gênero. Assim, se o gênero da música ainda não existir na *hashtable*, é adicionado e cria-se uma lista ligada nova com o nodo da idade do utilizador que gostou da música e apenas 1 *like*. Caso já exista, faz-se o gerenciamento dos nodos da lista ligada. Caso a idade atual já pertença à lista ligada, incrementam-se os *likes* de todas as idades anteriores e da atual. Caso não exista, incrementam-se os *likes* de todas as idades anteriores e cria-se um novo nodo para a atual.

Dado esta estrutura, para saber os gêneros com mais *likes* por faixa etária, terá que se percorrer a *hashtable genre* e, para cada gênero, subtrair ao número de likes da idade superior do intervalo o da idade inferior.

Eficiência

Em termos de eficiência dos processos de criação e busca das estruturas acima referidas com objetivo de responder às *queries*, pensamos que tenhamos encontrado soluções interessantes, com procuras rápidas o suficiente para satisfazer as necessidades desta 1ª fase, tanto de um ponto de vista intuitivo, como de um ponto de vista comparativo com os restantes grupos.

Entretanto, tencionamos experimentar novas soluções que possam melhorar o desempenho do nosso programa. Pensámos, posteriormente à implementação da *query* 3, tentar uma abordagem utilizando árvores binárias de procura organizadas por idades. Na seguinte fase do projeto analisaremos esta perspetiva, dado o tempo de inicialização da estrutura criada especificamente para esta *query* ser relativamente elevado, como se pode logo de seguida.

Apresenta-se na Tabela 1 o resumo da performance do programa, calculado através dos resultados dados pelo programa de testes (utilizando o datasets com erros), nos dispositivos de cada um dos integrantes do grupo, com as seguintes especificações:

- Dispositivo 1 (ASUS Vivobook S 15 S5506MA_S5506MA)
 - Especificações: Intel® Core™ Ultra 7 155H × 22, 16.0 GB RAM LPDDR5X, Ubuntu 24.04.
- Dispositivo 2 (Yoga 7i 2-in-1 Gen 9)
 - Especificações: Intel® Core™ Ultra 7 155U 16.0 GB RAM LPDDR5X-7.467MHz Ubuntu 24.04

- Dispositivo 3 (ASUS Zenbook 14X UM3402YA_UM3402YA)
 - Especificações: AMD® Ryzen 7 5825u with radeon graphics × 16, 16.0 GB RAM, Pop!_OS 22.04 LTS
- Dispositivo 4 (HP Victus by HP Laptop 16-d0xxx)
 - Especificações: 11th Gen Intel® Core™ i5-11400H @ 2.70GHz × 12, 20,0 GB RAM, Ubuntu 22.04.4 LTS

	Dispositivo 1	Dispositivo 2	Dispositivo 3	Dispositivo 4
Tempo de Inicialização DB (s)	4.505	5.119	5.833	5.102
Tempo de Inicialização Q2 (ms)	0.343	0.365	0.449	0.457
Tempo de Inicialização Q3 (s)	3.017	3.296	3.399	3.303
Query 1 (ms)	0.006	0.022	0.027	0.011
Query 2 (ms)	0.018	0.029	0.049	0.029
Query 3 (ms)	0.006	0.014	0.028	0.015
Tempo de Total Execução (s)	4.704	5.365	6.317	5.422
Tempo de Free (s)	0.270	0.243	0.483	0.321
Pico de Memória Utilizado (KB)	627968	625407	625024	625408

Tabela 1. Tempo de Inicialização da DataBase, em especial das estruturas criadas especificamente para executar as *queries* 2 e 3, tempo de execução médio das 3 *queries*, tempo total de execução, tempo de libertar a DataBase e máximo de memória utilizado. Valores obtidos da média de 10 execuções.

Conclusão

Ao longo do projeto, dedicamo-nos a entender de forma humilde e curiosa os novos conceitos lecionados durante as aulas, destacando-se a modularidade e o encapsulamento, e aprendemos a utilizar as ferramentas necessárias e atenuantes para o desenvolvimento do mesmo, tais como o *Valgrind* e o *GDB*.

O trabalho em grupo foi maioritariamente positivo, havendo uma base de entreajuda e discussão de ideias com finalidade de aprimorar a organização e “beleza” do nosso código, a sua eficiência e manter a coerência e organização do projeto. Derivado do empenho mútuo do grupo construímos uma base sólida para prosseguir para a seguinte etapa do projeto e encontramos-nos motivados para tal.

Dado isto podemos declarar orgulhosamente que se encontram resolvidas de forma correta e com soluções eficientes todas as tarefas obrigatórias propostas pela equipa docente.

Aspetos a melhorar

Embora tenhamos alcançado ótimos resultados durante esta primeira fase de desenvolvimento do projeto, reconhecemos espaço para melhoria. Desejamos, durante a segunda fase, investigar estruturas de dados que possam tornar o nosso programa mais eficiente, tanto em questão de tempo de execução como em memória utilizada, e aprimorar o nosso entendimento quanto às noções de modularidade e encapsulamento, visto serem conteúdos programáticos complexos com um grau de aplicabilidade árduos.

Encontramo-nos abertos a sugestões dos docentes de modo a melhorar o sistema e arquitetura construídos. Ambicionamos um patamar de classificação elevado, pelo que estamos ao dispor de aceitar ajuda externa e reavaliar e reestruturar alguns fatores do nosso código de modo a aprimorar a eficiência e organização do mesmo.