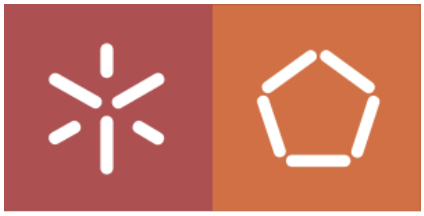


Universidade do Minho
Escola de Engenharia

Laboratórios de Informática III

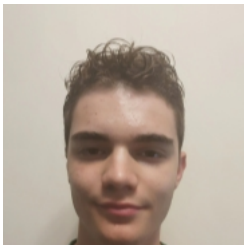
Trabalho prático – Fase 1

Grupo 60

Alexandre de Oliveira Monsanto, a104358

Maria Inês da Rocha Pinto Gracias Fernandes, a104522

Vasco João Timóteo Gonçalves, a104527



a104358

a104522

a104527

Novembro, 2024

Índice

Introdução

O presente relatório apresentará informações relativas à 1ª Fase do Trabalho Prático da Unidade Curricular Laboratórios de Informática III, pertencente ao 2º Ano da Licenciatura em Engenharia Informática da Universidade do Minho, realizada no ano letivo 2024/2025.

O projeto final referido consiste na implementação de uma base de dados em memória, utilizando-se ficheiros *.csv* fornecidos pelos professores, constituídos por dados relativos a um sistema de *streaming* de música (*artists*, *musics*, *users*), sendo o objetivo principal armazená-los e implementar métodos de pesquisa e associações entre estes.

São aplicados neste trabalho conceitos fundamentais como modularidade, encapsulamento, reutilização e abstração de dados.

Na 1ª fase, agora concluída, era necessário implementar o *parsing* e validação dos dados de entrada e o modo *batch*, bem como a realização das 3 *queries* propostas pela equipa docente.

Sistema

A aplicação, através do número de argumentos recebidos, seleciona o seu modo de funcionamento. Nesta fase foi apenas implementado o modo *batch*, sendo apenas este analisado nesta secção.

No início da aplicação, dentro dos catálogos do módulo *database*, são imediatamente criadas todas as estruturas necessárias para armazenar os dados que, posteriormente, serão preenchidas pelos próximos passos.

Através do primeiro argumento (caminho para os ficheiros *.csv* do *dataset*) é populada a base de dados que contém os catálogos de cada entidade. A leitura de dados do ficheiro é realizada através de um *parser* genérico que recebe apontadores para funções que fazem o tratamento, validação e inserção dos dados nos catálogos. Caso a validação dos dados falhe, o *parser* escreve num ficheiro passado como argumento.

Seguidamente, é realizado o *parsing* dos comandos que obtém o índice da *query* a ser chamada, o modo de *output* dos dados e os argumentos da *query*. As *queries* utilizam funções de listagem e obtenção de dados disponibilizadas pelos catálogos.

Através de um módulo de *writing* genérico com modos de *output* tipo *csv* e tabular (*flag* F), com funções que recebem um número variável de argumentos, respetiva formatação e nome dos campos, as *queries* vão escrevendo as "respostas" num *buffer* de ficheiro, transferido para o disco no final da execução do comando.

Após a execução do programa, é feita uma limpeza da memória, eliminando todas as estruturas previamente criadas pelos módulos e, consequentemente, eliminados os dados do *dataset*, evitando *memory leaks*.

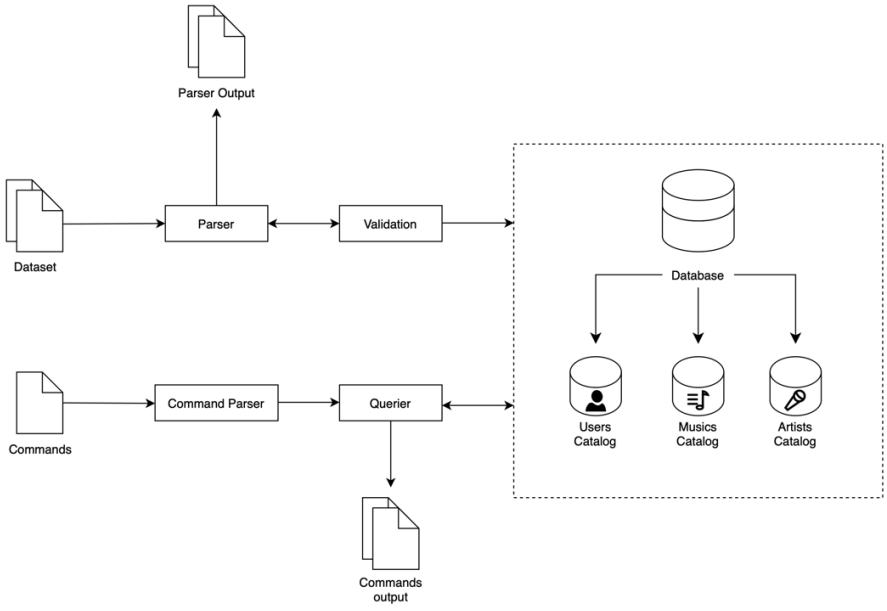


Figura 1 - Arquitetura para a aplicação a desenvolver

Discussão

1. Análise de desempenho

O desempenho do projeto foi avaliado com base nas *queries* realizadas, testando-se o tempo de execução e o consumo de memória nas diferentes máquinas dos membros da equipa. Embora as diferenças de desempenho possam ser observadas devido ao *hardware* variado, os resultados são consistentes com o esperado, dado o uso de algoritmos eficientes, como as *HashTables*.

2. Modularização e encapsulamento

A modularização foi aplicada ao longo do projeto, dividindo-o em várias unidades de código bem definidas e independentes, como módulos para *users*, *musics*, *artists*, *queries* e *utils*. Cada módulo tem uma responsabilidade clara, facilitando a manutenção e a expansão do código. A modularização também ajuda na reutilização do código e na redução da duplicidade.

O encapsulamento foi usado para proteger os dados dentro de cada estrutura, garantindo que as alterações só possam ser feitas por funções específicas, minimizando os efeitos colaterais e a possibilidade de erros. Por exemplo, a criação de novos usuários é feita através de funções como *create_and_store_user*, que asseguram a integridade dos dados.

3. Estruturas de dados

As escolhas de estruturas de dados foram fundamentais para o desempenho e eficiência do sistema. As HashTables foram escolhidas para armazenar os *users*, visto que permitem uma procura, inserção e remoção eficientes. O uso de *GPtrArray* da biblioteca *GLib* para armazenar os artistas, por outro lado, facilita a manipulação dinâmica de listas com capacidade de redimensionamento automático.

Essas escolhas permitiram que o sistema lidasse eficientemente com grandes volumes de dados e consultas, melhorando a capacidade de resposta do sistema. Para a organização de dados como os ids de músicas ou artistas, as listas de *strings* foram adequadas, com validação e *parsing* de dados implementados de forma robusta.

4. *Queries*

4.1. *Query* 1

A *Query* 1 permite listar o resumo de um utilizador com base no seu identificador único. Utilizando o ID fornecido, o programa pesquisa nas estruturas de dados, armazenadas em memória, através do catálogo de utilizadores. A implementação é facilitada pelo uso de uma função que vai buscar diretamente o utilizador pelo ID, garantindo uma pesquisa eficiente e precisa. A *query* devolve os dados do email, primeiro e último nome, idade e país do utilizador, formatados de acordo com o enunciado. Caso o ID não seja encontrado, o *output* devolve uma linha vazia.

4.2. *Query* 2

A *Query* 2 permite listar os artistas com a maior discografia, definida pela soma das durações das suas músicas. A *query* aceita um argumento opcional de filtro por país, que, se existir, considera apenas os artistas desse país. A implementação utiliza um *array* de apontadores (*GPtrArray*) para armazenar e processar os artistas, sendo que o tamanho da discografia é calculado individualmente para cada artista.

Cada entrada é formatada com o nome do artista, tipo (individual ou grupo), duração da discografia e país, respeitando o limite imposto pelo argumento 'N'. A ordenação considera a duração total da discografia e, em caso de empate, o ID do artista (por ordem crescente). Para casos em que o filtro de país exclui alguns artistas, a *query* ajusta automaticamente o número de entradas para completar o top N.

5. Resultados obtidos

Os resultados obtidos nos testes são coerentes com as expectativas, dado que os algoritmos e estruturas de dados utilizadas (*HashTables*, *arrays* dinâmicos e listas de *strings*) são conhecidos pela sua eficiência em situações de grande volume de dados. A modularização e encapsulamento também desempenham um papel importante na redução da complexidade do código e na facilidade de manutenção, sem afetar o desempenho geral do sistema. Em suma, as escolhas técnicas feitas garantiram um equilíbrio entre eficiência e clareza do código.

Conclusão

O trabalho desenvolvido nesta primeira fase satisfez, de forma geral, os requisitos pedidos. Este projeto foi desenvolvido com o objetivo de ser o mais eficiente e rápido possível, tentando sempre cumprir as regras de encapsulamento e modularidade.

O grande desafio deste projeto foi a programação à larga escala, com base numa grande quantidade de dados disponíveis, sendo imperativo trabalhar com estruturas de dados e algoritmos que se desenvolvessem tanto em complexidade como em eficiência para que pudéssemos concretizar os objetivos. Isto obrigou-nos a explorar e a conhecer uma nova API (GLib) e, consequentemente, ter em consideração os métodos mais eficientes para cada caso. A escolha de estruturas de dados como *HashTables* garantiu-nos que as consultas possam ser feitas de forma rápida e eficaz.

Algumas dificuldades que fomos enfrentando ao longo da realização do projeto foram a resolução de *memory leaks*, nem sempre óbvias, e a contínua adaptação de estruturas de dados ao longo da criação das *queries*.

O sistema cumpriu os objetivos propostos, mas poderá ser aprimorado numa fase futura.