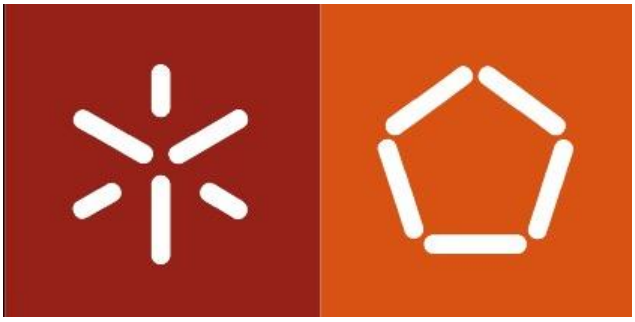

Universidade do Minho
Licenciatura em Engenharia informática

Projeto Laboratórios de informática 2º Fase

Departamento de informática

Universidade do Minho - janeiro de 2024

Desenvolvido pelo grupo 45:

Afonso Pedreira (A104537)

Dário Guimarães (A104344)

Hugo Rauber (A104534)

Índice:

1. Introdução

O presente relatório refere-se à segunda fase do desenvolvimento do projeto proposto na unidade curricular de Laboratórios de informática III do ano 2023/2024, que visa a criação de um programa de gestão e consulta de dados relacionados a utilizadores, voos e reservas.

A segunda fase concentrou-se em acabar as queries não finalizadas na primeira fase, polir certos aspetos como tempos de execução, encapsular o projeto, e implementar uma interface para o modo interactive.

Tal como na primeira fase, e recém mencionado, o projeto segue uma arquitetura modular e encapsulada e segue as diretrizes dadas no enunciado do projeto.

Durante esta segunda fase foram encontrados alguns desafios, tais como a gestão de memória, tempos de execução e a implementação do modo interactive.

Este relatório está dividido em capítulos. Alguns tópicos são retomas da primeira fase, e, como se mantiveram inalterados, foram novamente mencionados aqui. Assim, o relatório aborda os **problemas iniciais na segunda fase**, os **testes e workflow de trabalho**, **funcionalidades implementadas**, **encapsulamento**, **interface**, **métricas de desempenho,** e uma **conclusão**.

2.Testes e Workflow de trabalho

Tal como na primeira fase, para evitar certos problemas, tomámos certas decisões para diminuir as consequências.

2.1. Testes Unitários (1ª fase)

Com o objetivo de desde cedo encontrar os erros que poderiam escalar, implementamos testes unitários às queries, através dos *datasets* fornecidos. Isto é para cada query corremos os seus exemplos fornecidos e verificamos se o output coincide com o esperado. Tudo isto pode ser acedido através do comando *make test*.

A mesma abordagem foi tomada para os memory leaks, que facilmente, através do comando *make check-memory*, podem ser prevenidos, e para formatar o código, com *clang-format*.

2.2. Testes de Desempenho e Gerais

Para facilitar a utilização do programa por parte dos usuários que desejam verificar a sua correta implementação. Ao executar o programa, há a opção de fornecer um *dataset,* as queries a serem testadas e os ficheiros esperados. Assim, ao executar, será devolvido um ficheiro com tempos de execução, memória utilizada e as comparações feitas (resultados obtidos com os esperados).

Assim, é dada uma análise abrangente do desempenho geral do programa.

2.3. GitHub Actions e Review

Para além do referido na primeira fase sobre as GitHub Actions juntamente com os comandos make, a necessidade de haver revisões por parte de algum colega e o código ter de passar por testes. Ainda, criou-se um repositório adjunto para ter total acesso a certos parâmetros. Decidiu-se assim que, para esta segunda fase, todos os *commits* tinham de passar primeiro pelo repositório auxiliar, serem testados e aprovados por todos os membros, e só depois enviados para o principal, impossibilitando, assim, *merges* com a *main* que de outra forma poderiam ter sido aprovados mesmo estando com erros.

2.4. Documentação (1ª fase)

Por fim, escrevemos uma documentação para todas as funções que estão expostas nos ficheiros de *headers* para que o trabalho em grupo fosse facilitado, estando assim sempre por dentro de como usar os outros módulos e que parâmetros estão eles à espera, bem como o que vão retornar. A documentação pode ser exportada, através do *Doxygen*, com o comando *make doxygen*.

3. Funcionalidades Implementadas

Aqui, mostram-se, as Funcionalidades Implementadas. Como algumas já foram mostradas na primeira fase, aqui vão aparecer as 10 queries de forma resumida.

Cada query é encarregada de criar um ficheiro de saída, chamar uma funcção para escrever nele e de o fechar.

3.1. (Q1) Listar informações de um utilizador, voo ou reserva

A função query_1 recebe um identificador único (id) e um conjunto de catálogos (catalogs). Com base no tipo de identificador, verifica se é um número, uma reserva ou um voo, e chama a função correspondente (write_flight_data, write_reservation_data, write_user_data). Cada função de escrita formata as informações relevantes.

3.2. (Q2) Listar voos ou reservas de um utilizador

A função query_2 recebe um identificador de usuário (id) e um argumento opcional (optional) que especifica se deve listar voos, reservas ou ambos. Com base nessas entradas, a função recupera as informações relevantes do usuário do catálogo (catalogs). Em seguida, itera sobre as listas encadeadas de voos e reservas do usuário, comparando as datas de início para ordenar a saída por data.

Durante a iteração, as informações de voos e reservas são formatadas e escritas, com a opção de incluir o tipo ("flight" ou "reservation") se necessário.

As listas de voos e reservas são liberadas da memória. O resultado final é uma lista ordenada de voos, reservas ou ambos, dependendo do argumento opcional.

3.3. (Q3) Apresentar classificação média de um hotel

A função query_3 recebe um identificador de hotel (id) e um conjunto de catálogos (catalogs). A partir desse identificador, obtém o hotel correspondente e recupera a sua avaliação. A avaliação é então convertida para uma string.

3.4. (Q4) Listar reservas de um hotel

A função query_4 recebe um identificador de hotel (id) e um conjunto de catálogos (catalogs). A partir desse identificador, obtém o hotel correspondente e recupera a lista de reservas associadas a esse hotel. A função itera sobre a lista de reservas, formatando as informações relevantes, como ID da reserva, datas de início e término, ID do usuário, classificação e preço total.

3.5. (Q5) Listar voos de um aeroporto entre datas

A função query_5 recebe o nome de um aeroporto (name), uma data de início (begin_date) e uma data de término (end_date), além de um conjunto de catálogos (catalogs). A partir do nome do aeroporto, obtém a lista de voos associados a esse aeroporto. A função itera sobre a lista de voos, verificando se o horário de partida está dentro do intervalo especificado.

Para cada voo que atende aos critérios, as informações relevantes, como ID do voo, data de partida, destino, companhia aérea e modelo da aeronave.

3.6. (Q6) Listar o top N aeroportos com mais passageiros, para um dado ano

A função query_6 recebe um ano (year), um valor de N (n), e um conjunto de catálogos (catalogs). Converte o ano para um formato inteiro. Em seguida, obtém uma lista dos top N aeroportos com base no número de passageiros para o ano especificado.

A função itera sobre a lista, obtendo o ID e o número de passageiros de cada aeroporto.

3.7. (Q7) Listar o top N aeroportos com a maior mediana de atrasos

A função query_7 recebe um valor de N (top_n_airports) e um conjunto de catálogos (catalogs). Obtém uma lista dos top N aeroportos com base na mediana do atraso de voos.

A função itera sobre a lista, obtendo o ID do aeroporto e sua mediana de atraso.

3.8. (Q8) Apresentar receita total de um hotel entre datas

A função query_8 recebe um identificador de hotel (id), uma data de início (begin_date) e uma data de término (end_date), além de um conjunto de catálogos (catalogs). A partir do identificador do hotel, obtém o hotel correspondente e calcula a receita total para o período especificado.

A receita total é convertida para uma string.

3.9. (Q9) Listar utilizadores por prefixo de nome

A função query_9 recebe um prefixo (prefix) e um conjunto de catálogos (catalogs). Cria um arquivo de saída e obtém uma lista de utilizadores cujos nomes começam com o prefixo especificado.

A função itera sobre a lista de utilizadores, obtendo o ID e o nome de cada utilizador.

3.10. (Q10) Apresentar várias métricas gerais da aplicação

4. Encapsulamento e Funcionamento da aplicação

Na segunda fase deste projeto, a estrutura foi organizada de forma encapsulada para tornar tudo mais fácil de entender e utilizar. Têm-se assim, duas subpastas importantes:

- **Catálogos:** Trata-se da organização e gerenciamento das entidades. Usam-se várias ferramentas inteligentes, como árvores de busca e catálogos personalizados, para facilitar a busca nas informações.
- **Entidades:** Aqui, mantêm-se as informações brutas das entidades. É onde se guardam os dados.
-

Esta abordagem teve pontos positivos:

- **Abstração:** A variedade de catálogos esconde muita complexidade, permitindo que haja uma maior concentração noutras áreas de forma mais simples.
- **Organização:** Com esta estrutura de pastas e a lógica dos catálogos, tudo fica no seu lugar. É fácil encontrar o que se precisa, mesmo quando o projeto cresce.
- **Reutilização de Código:** A consistência nos catálogos significa que se podem reutilizar pedaços de código em várias partes do projeto.
- **Manutenção:** Permite fazer mudanças internas mais facilmente sem mexer em certas partes externas e causar problemas.

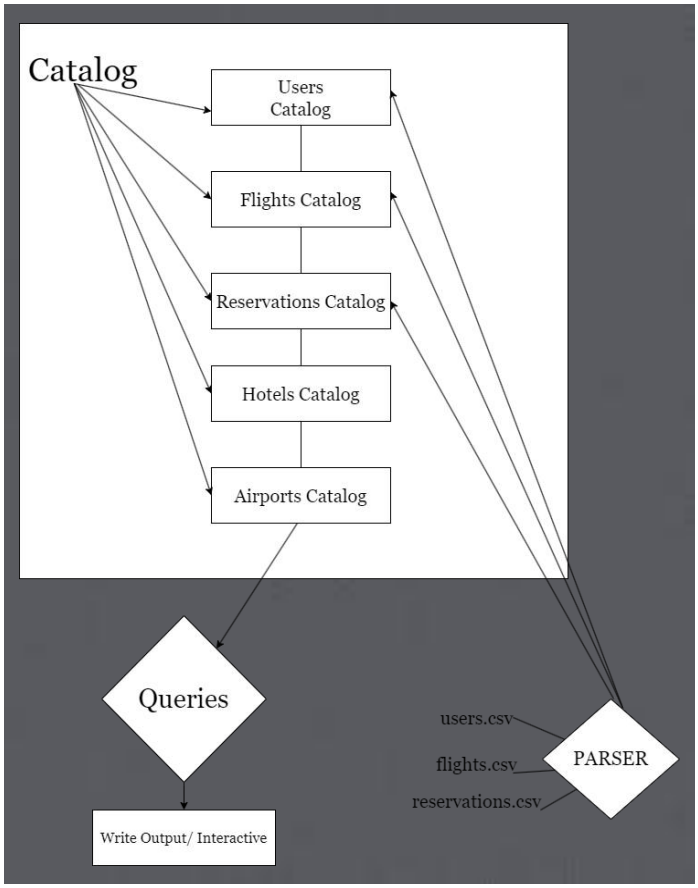

Fig 1.- Esquema do funcionamento da aplicação

5. Modo interactive (Trivial Booking)

O modo interactive permite ao usuário interagir com um menu e poder obter resultados das queries de uma forma mais simples.

Embora não tenha um modo de resultados tão complexo, permite receber uma linha de input (sem acentos) e devolver no ecrã, os resultados em forma de paginação, assim como um output em ficheiro. Deu-se como nome à aplicação “Trivial Booking”.

O interactive funciona com diferentes estados. É chamada a função interactive quando não são passados argumentos na execução do programa. É desenhado o menu principal e chamada a função ´events_menu´ a qual deteta as teclas pressionadas e a intenção do user de mudar de menu, mudando assim o ´State´ para o menu atual, chamando novas funções ´draw_(menu atual)´ e ´events_(menu atual)´. À medida que o menu muda, novas funções são chamadas pelo interactive. Podem-se ver a seguir, imagens sobre as iterações pelo menu:

Fig.2

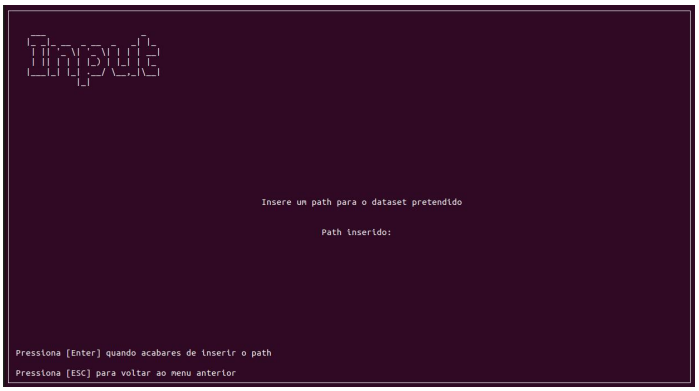

Fig. 3

Fig. 4

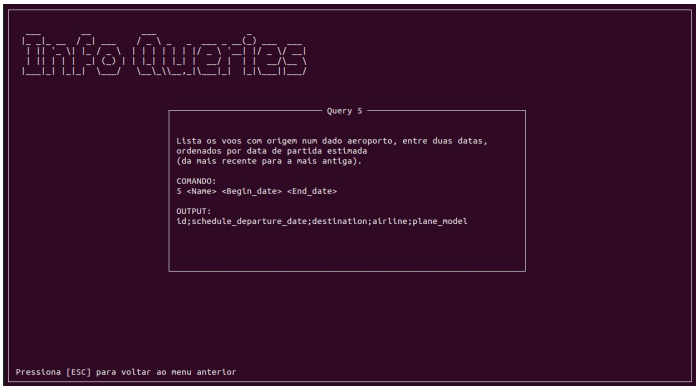

Fig. 5

Fig. 6

Fig. 7

6. Métricas de desempenho

Nesta segunda fase, realizaram-se melhorias significativas na aplicação, resultando numa redução notável no consumo de memória, de 5GB para 2,6GB, maximizando a eficiência dos recursos. Isto foi possível através da introdução de encapsulamento no projeto e da implementação de estruturas de dados auxiliares estrategicamente concebidas.

Assim, testaram-se 3 equipamentos diferentes, com recurso aos testes **(2.2)** para determinar tempos de execução.

Pode-se ver na tabela seguinte, o resultado dos testes.

	Afonso	Dário	Hugo
CPU	Ryzen 5 5500U 2,1GHz	Ryzen 5 3600 3,6Ghz	Ryzen 5 3500U 2,1Ghz
Cores & Threads	6&12	6&12	4&8
RAM	8GB DDR4 3200MHZ	8GB DDR4 3200MHZ	8GB DDR4 3200MHZ
Sistema Operativo	Manjaro Linux	Arch Linux	Ubuntu
Máximo de memória usada	2,64GB/20KB	2,64GB/20KB	2,64GB/20KB
Large Dataset Tempo Exec	130,04s	127,32s	138,58s
Normal Dataset Tempo Exec	0,57s	0,53s	0,62s

O programa foi executado 5 vezes, tendo sido depois efetuada a média dos tempos obtidos.

7. Conclusão

Nesta etapa final do projeto, reforçamos e solidificamos ainda mais a estrutura do código, alcançando resultados superiores com a incorporação do encapsulamento. Esta adição fortaleceu a robustez do projeto, contribuindo para a manutenção da coesão e aprimorando a sua escalabilidade. A introdução da interface também proporcionou uma experiência mais simplificada na obtenção de resultados aos usuários.

Durante esta fase, como mencionado anteriormente, também enfrentamos alguns desafios. O grupo manteve-se mais coeso e colaborativo e, embora ainda se tenham notado divergências no conhecimento técnico sobre a linguagem C, nesta fase, o trabalho foi melhor distribuído. Assim, houve uma dedicação da parte de todos os elementos, o que contribuiu para um bom e satisfatório resultado.