Trabalho prático - LI3

Relatório da 1ª fase

Ana Sá Oliveira, a104437
Inês Silva Marques, a104263
José Rafael De Oliveira Vilas Boas, a76350

LEI - 2023/2024

Índice

Arquitetura do projeto

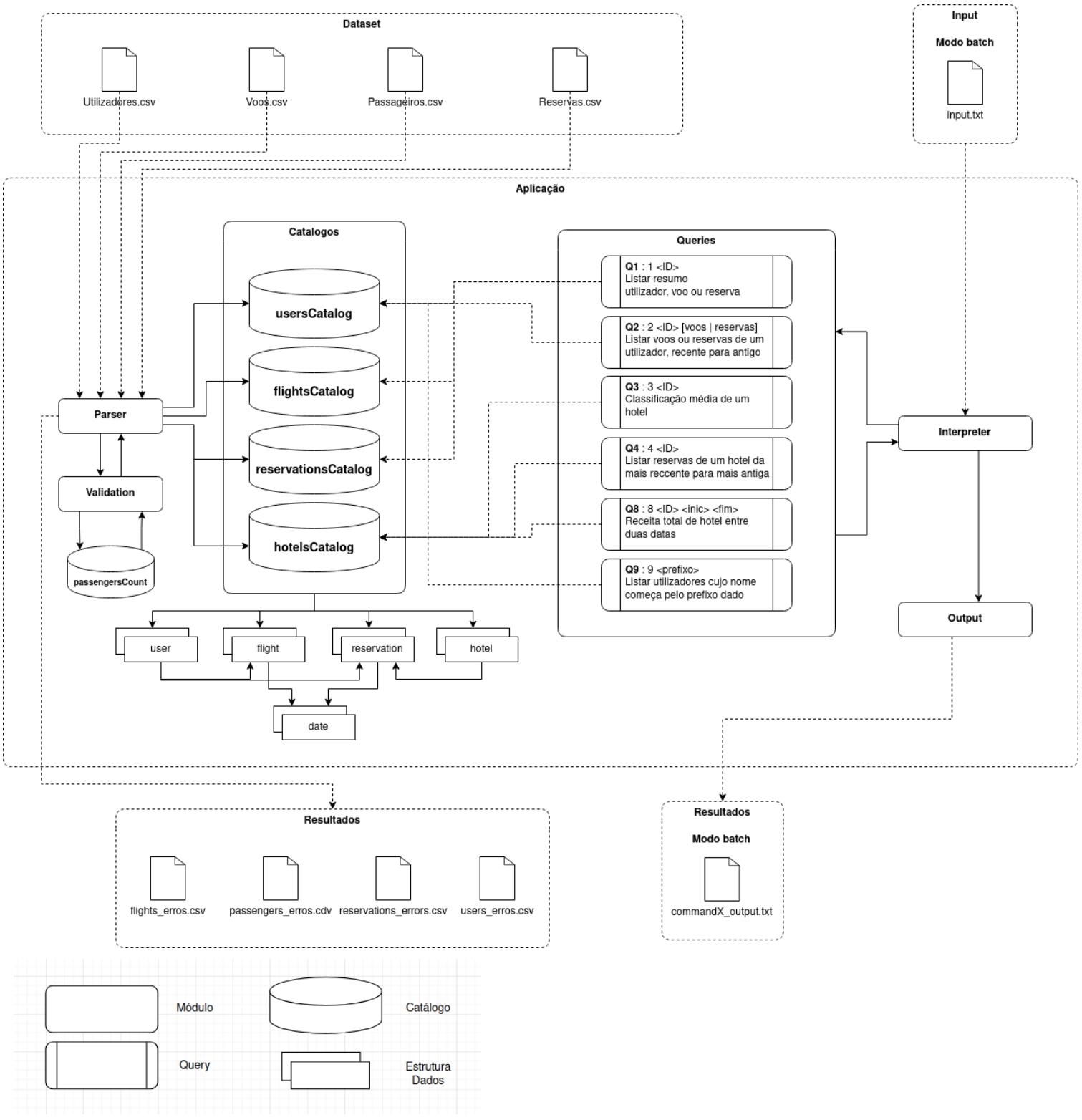

Figura 1 - esquema da arquitetura do programa

Parser

O parser é o módulo no qual é realizada a leitura dos ficheiros de entrada CSV e é efetuado um parsing genérico. Para cada tipo de ficheiro (users.csv, reservations.csv, flights.csv e passengers.csv), descobrimos o caminho para o mesmo, verificamos se o ficheiro existe e, se existir, abrimos o ficheiro. De seguida, lemos linha por linha o conteúdo dos ficheiros. Cada linha será repartida em várias partes. Cada parte representa um aspecto sobre uma determinada identidade (por exemplo o email de um utilizador). Se cada parte da linha for válida, ou seja, se a linha for válida (segundo as funções do dataset validator) guardamos essa informação nas estruturas de dados, ou seja, nos catálogos de cada identidade. Se a linha for inválida, iremos colocá-la nos ficheiros de erros.

Validação dos dados

O dataset validator é o módulo no qual é realizada a validação dos dados dos ficheiros CSV. Neste módulo, podemos saber se um utilizador, uma reserva, um voo ou um passageiro é válido. Para saber se uma identidade é válida analisamos as várias partes da linha (strings) que representam aspectos sobre a identidade. Para isso usamos várias funções auxiliares. Por exemplo, valid_email verifica se um determinado email é válido. Para verificarmos se um utilizador é válido apenas precisamos que as várias partes da linha do utilizador sejam válidas. No entanto, para as outras identidades não basta isso.

Nas reservas também precisamos de saber se o utilizador existe no catálogo de utilizadores válidos. Nos voos precisamos de saber se o número de lugares de um voo é superior ou igual ao número de passageiros. Neste caso, teremos que ler o ficheiro dos passageiros para contar o número de passageiros válidos (com utilizador válido) por voo. Depois já podemos validar os voos. E por fim, podemos validar os passageiros, que apenas precisam de saber se o utilizador existe no catálogo de utilizadores válidos e se o voo existe no catálogo de voos válidos.

Tipos e estruturas de dados

Os tipos e estruturas de dados são responsáveis por representar e armazenar dados. Neste projeto utilizamos duas estruturas de dados principais: hashtables e arrays. Fizemos 4 hashtables: uma para os utilizadores, outra para as reservas, outra para os voos e outra para os hoteis. Cada hashtable é formada por estruturas específicas. As estruturas dos utilizadores tem vários parâmetros sobre o utilizador e tem uma lista de voos e reservas desse mesmo utilizador. A estrutura dos voos tem vários parâmetros sobre o voo e apresenta uma lista dos passageiros desse voo. A estrutura das reservas tem vários parâmetros sobre a reserva. Já a estrutura dos hoteis apresenta vários parâmetros sobre o hotel e apresenta uma lista das reservas do hotel.

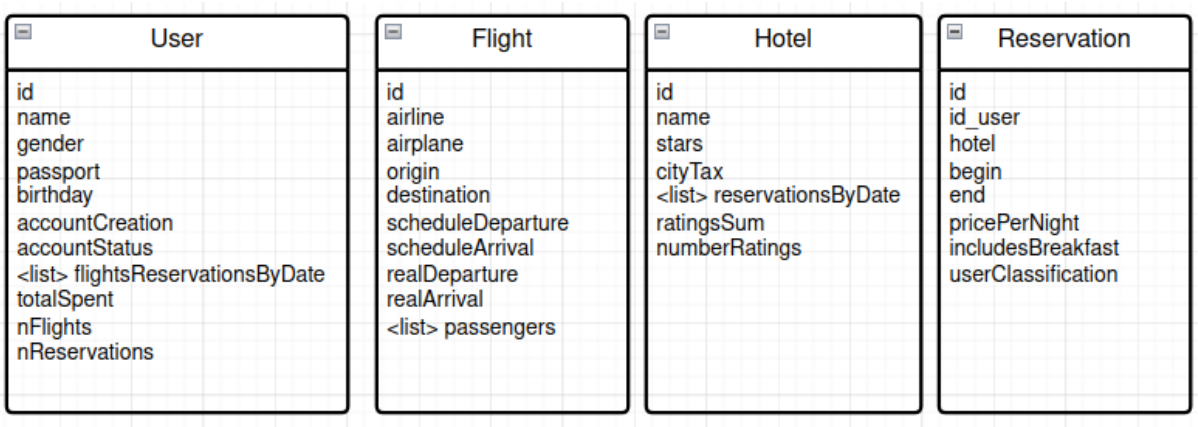

Figura 2 - estruturas de dados

Catálogos de dados

Os catálogos de dados são módulos em que são armazenadas as entidades, e processadas as informações necessárias por outros módulos relativamente a essas entidades.

Nestes catálogos todas as entidades que têm um id associado foram guardadas numa hashtable, para que o acesso a uma destas entidades seja feito em tempo constante dado o seu id.

Os catálogos implementados são:

- O catálogo de utilizadores nos ficheiros usersManager.(c/h), em que são guardados todos os utilizadores numa hashtable e

também os utilizadores ativos numa lista, que será posteriormente ordenada por nome e id de utilizador. Este módulo também é responsável por ordenar a lista de utilizadores por nome e id, e por ordenar as listas de voos e reservas de cada utilizador, por data.

- O catálogo dos voos nos ficheiros flightsManager.(c/h), em que estão guardados todos os voos numa hashtable.

- O catálogo de reservas nos ficheiros reservationsManager.(c/h), onde estão guardadas todas as reservas numa hashtable.

- E o catálogo dos hoteis nos ficheiros hotelsManager.(c/h), onde estão guardados todos os hoteis numa hashtable, e onde são ordenadas as listas de reservas de cada hotel, por data.

Interpretador de comandos

O módulo de interpretação de comandos (ficheiro “interpreter.c”) é onde é feito o parsing do ficheiro de comandos no modo batch, de acordo com o path passado como argumento ao programa. A função main chama a parseCommandFile com o path do ficheiro de comandos e os catálogos onde ir buscar os dados, e esta lê o ficheiro linha a linha, enviando cada uma para a parseCommandLine, que devolve uma estrutura Command, com a informação dessa linha - id da query a que o comando se refere e os respetivos argumentos. É depois chamada a função processCommand que, com o comando, o nº da linha e os catálogos, chama funções para criar um ficheiro para o output, calcular o resultado da query (módulo das queries), e imprimir esse resultado no ficheiro criado (módulo do output). A função parseCommandFile também é responsável por libertar o espaço ocupado pela estrutura que guarda um comando e pela linha lida do ficheiro de comandos.

Queries

Query 1

A query 1 tem como objetivo devolver um resumo da entidade (utilizador, reserva ou voo) associada a um id.

Para isso, a função Q1 começa por determinar a que tipo o id pertence, depois procura o id no catálogo desse tipo, se o este não estiver no catálogo, ou pertencer a um utilizador inativo, a função devolve NULL, o que vai ser interpretado pela função que escreve o output. Caso contrário, a função devolve o utilizador, reserva ou voo encontrado, e a função que imprime o output utiliza gets para aceder aos dados necessários para fazer o resumo - para isso existem também campos nas estruturas das entidades ou funções que armazenam ou calculam, respetivamente, os dados necessários para os resumos que não estejam diretamente nos ficheiros CSV, como a idade de um utilizador, ou o total gasto pelo mesmo, por exemplo.

Query 2

A query 2, se for executada com um argumento, deve listar os voos e reservas de um utilizador, e, se tiver dois argumentos, deve listar só os voos ou reservas, conforme o segundo argumento.

A função Q2 começa por verificar se o utilizador em questão existe e está ativo (caso contrário devolve NULL), e, em seguida, verifica se existe e qual é o segundo argumento. Se tiver de listar voos e reservas, a função percorre a lista de voos e reservas associada ao utilizador e coloca-a na variável que vai ser usada como resultado da query, se só listar um dos tipo de dados (voos ou reservas), percorre a lista e verifica se cada elemento é do tipo que é pedido e, se sim, coloca-o na lista dos resultados.

A função devolve uma lista com a informação e o seu tipo por ordem, que a função de output vai usar para escrever as informações de cada elemento desta.

Query 3

O objetivo da query 3 é, dado o id de um hotel, calcular a sua classificação média.

Para isso, a estrutura da entidade Hotel tem um campo ratingsSum onde é somada a classificação de cada reserva desse hotel (se a reserva tiver classificação) e outro campo que conta o número de classificações que esse hotel recebeu.

Com esses dados, a função Q3 calcula a classificação média do hotel associado ao id fornecido, se o encontrar no catálogo, se não, devolve -1, que a função do output vai interpretar como não sendo um resultado válido.

Query 4

A query 4 deve listar as reservas do hotel associado ao id passado como argumento.

Para isso, a função Q4 pesquisa esse id no catálogo de hoteis e, se não encontrar um hotel devolve NULL, caso contrário, percorre a lista de reservas do hotel encontrado e coloca cada uma numa lista que vai ser usada como resultado.

Essa lista vai ser passada à função do output, que utiliza gets para aceder às informações de cada reserva na lista. Para isso também existe uma função auxiliar que, dada uma reserva, calcula o seu custo.

Query 8

A query 8 apresenta a receita total de um hotel entre duas datas limite dadas, tendo em conta apenas o preço por noite de cada reserva.

Assim na função Q8, como as listas de reservas dos hoteis são ordenadas por data de início da reserva, é percorrida a lista de reservas do hotel, até que a data de início de uma reserva seja maior que a data final do limite dado, e calculado o preço de cada reserva

entre essas datas adicionando ao total que no final vai ser passado à função do output.

Como é necessário saber o número de noites de cada reserva foi necessário fazer uma função auxiliar que dadas as quatro datas, os dois limites e as datas de início e fim da reserva, calculasse o número de noites dessa reserva entre as datas limites, que por sua vez necessitou de uma outra função auxiliar que converte-se uma data em dias para que fosse possível calcular o número de noites entre estas.

Query 9

Na query 9 são devolvidos todos os utilizadores ativos, cujo o seu nome comece por um prefixo dado como argumento, ordenados por ordem alfabética e por id, caso o nome seja igual.

Na função Q9 é usada a lista de utilizadores ordenada por nome e id, que se encontra no catálogo de utilizadores, foi usada a função searchDataOrdList para procurar a primeira posição da lista em que o nome de um utilizador começa pelo prefixo dado, caso este exista, e depois é percorrida a lista verificando se em cada posição consecutiva também o nome começa pelo prefixo dado, cada utilizador para o qual isso se verifica é adicionado a uma nova lista, resultado, que irá ser passada à função do output.

Para isto foi criada uma lista de utilizadores e adicionados apenas os ativos no parser, esta lista é depois ordenada por nome e id.

Output

O módulo do output é responsável por criar os ficheiros para os resultados e escrevê-los. Para isso existe a função createOutputFile, que recebe o número que deve ser colocado no nome do ficheiro, e cria o ficheiro na pasta Resultados, e uma função printOutput para cada query já implementada. Estas funções recebem o resultado de cada query e a format flag do comando, e escrevem o output de cada query de acordo com estes dados. Para cada query onde é possível não existir um resultado válido (id passado como argumento não

corresponde a nenhuma entidade, por exemplo), a função de output também interpreta esse resultado (NULL no caso de a função da query retornar um apontador e -1 se retornar um número) e não o escreve no ficheiro de output.

As funções printOutput também libertam a memória alocada para os resultados das queries.

Utilidade

O módulo da utilidade contém várias funções usadas em vários momentos e módulos do programa. Neste módulo a maioria das funções são sobre datas. As datas são usadas em várias partes do programa, por isso, merecem o seu próprio módulo.

Aspetos a melhorar

Nesta primeira fase do desenvolvimento deste projeto o foco foi em criar uma aplicação que obedecesse aos critérios estabelecidos para a correta execução do programa e das queries escolhidas para implementação, de maneira a que o código escrito fosse de fácil compreensão e alteração, e em dividir a aplicação em módulos, responsáveis cada um por uma parte distinta do programa como um todo, tentando ter em conta sempre que possível o encapsulamento destes, apesar de não ser um aspeto contabilizado durante esta fase, desenvolvemos o programa tendo isso em conta para a segunda fase do desenvolvimento.

No entanto, tendo agora um programa correto, de acordo com os critérios estabelecidos e testes feitos, poderemos prosseguir à optimização do desempenho deste, localizando quais partes de quais módulos estão a executar mais lentamente e a ocupar mais memória, otimizando cada parte separadamente obtendo assim no final um programa mais eficiente.

Estas otimizações devem ser feitas primeiramente nas partes do programa que executam mais vezes, como por exemplo partes que são chamadas repetidamente por vários loops em várias funções.

Algumas destas otimizações seriam:

- **Apontadores que são des-referenciados múltiplas vezes, dentro de loops, deverão ser guardados numa variável local, para que a cadeia de apontadores não seja sempre feita, não havendo essa necessidade;**
- **Declarar apontadores para estruturas em argumentos de funções, quando apenas é feita leitura de parâmetros na estrutura e não escrita, como "const Estrutura *apontador" para informar o compilador de que a função não altera os valores da estrutura, e que assim não é necessário estar sempre a verificar se a estrutura foi alterada em memória.**

Outros aspetos a melhorar:

- **Finalizar a implementação do módulo catalogsManager, de maneira a gerir tudo que esteja ligado a catálogos, para um acesso mais fácil a estes nos outros módulos;**
- **Finalizar o encapsulamento dos módulos.**