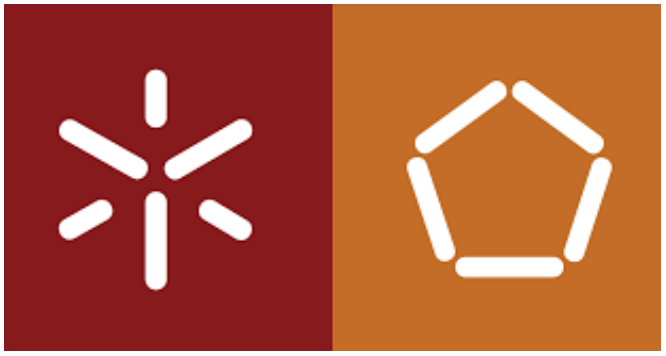


Universidade Do Minho
Engenharia Informática

# Trabalho Prático

# Laboratórios de Informática III 2023/2024

Relatório Grupo17 data de entrega: **22/11/2023**

# Composição do Grupo 17

**Tiago Pinheiro Silva**
**a93285**

**Tiago Alexandre Ferreira Silva**
**a93182**

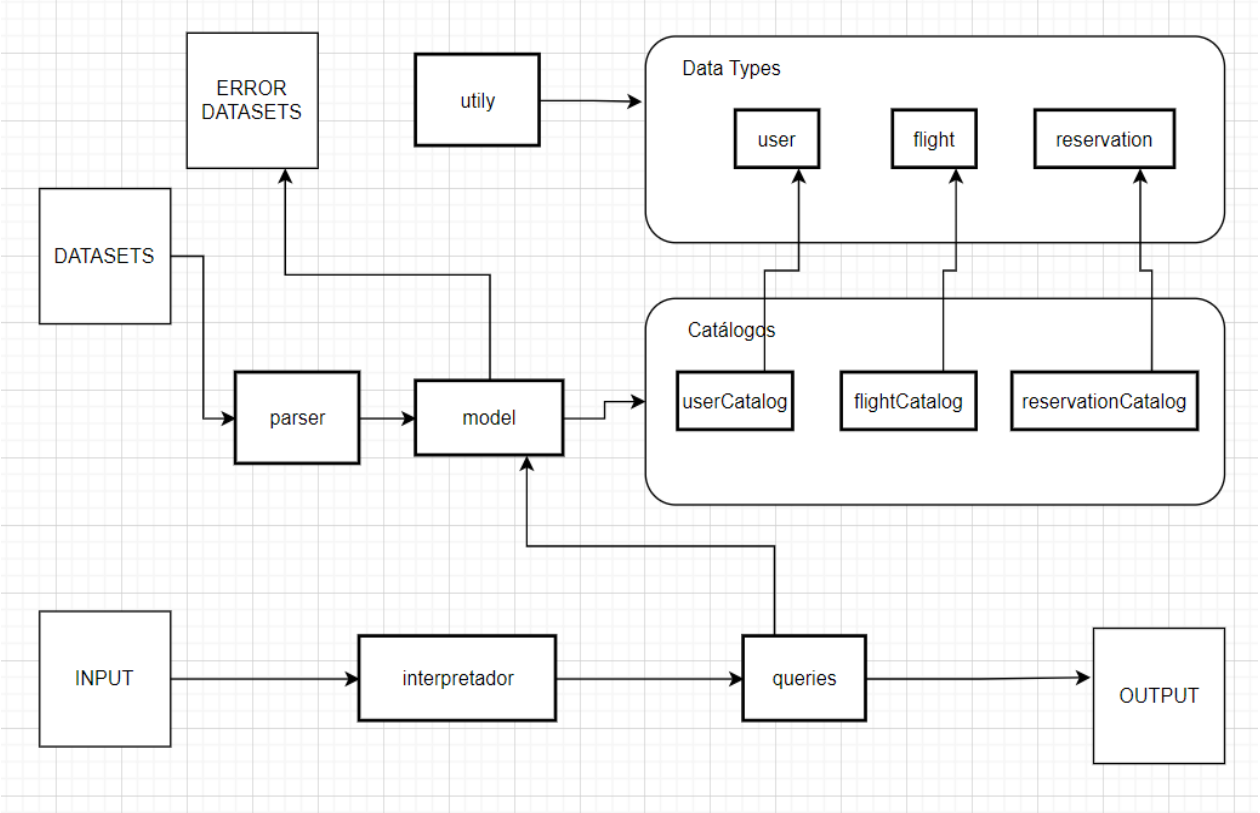
**Carlos Alberto Fernandes Dias Da Silva**
**a93199**

# Índice

# Introdução

Este trabalho prático da Unidade Curricular de Laboratórios de Informática III tem como fim consolidar e aprofundar competências de Engenharia Software e da linguagem de programação C.
Encontramo-nos então perante os objetivos de cimentar conhecimentos relativos à modularidade e encapsulamento, a estruturas dinâmicas de dados, validação de ficheiros e medir desempenho.
Paralelamente, este conjunto de objetivos ajudar-nos-á a solidificar o nosso domínio em C, junto da gestão eficiente de um repositório.

# Desenho e Implementação




O nosso projecto foi estruturado em torno de uma arquitetura com três entidades que sobressaltam. USER, FLIGHT e RESERVATION, cada uma representada por structs correspondentes com os atributos que achamos necessários para o desenvolvimento deste sistema. Estas entidades encapsulam informações essenciais, como detalhes do utilizador, dados de voos e informações de reservas.

Uma estrutura a qual denominamos de model vai orquestrar este sistema, tendo como atributos 5 tabelas hash: “flights_by_id”, “flights_by_origin”, “reservation_by_id”, “reservation_by_hotel” e “user_by_id”. Cada tabela irá facilitar o acesso rápido a instâncias de entidades com base em identificadores únicos.

**A estrutura destas HashesTables será a seguinte:**

**<KEY,VALUE>**

**flights_by_id**

<FLIGHT_ID**,** instância de FLIGHT com FLIGHT_ID correspondente>

**flights_by_origin**

<ORIGIN**,** GLists c/ todas instâncias de FLIGHT com origin correspondente>

**reservation_by_id**

<RESERVATION_ID**,** instância de RESERVATION c/ RESERVATION_ID correspondente>

**reservation_by_hotel**

<HOTEL, GLists c/ todas instâncias de RESERVATION com hotel correspondente>

**user_by_id**

<USER_ID**,** instância de USER com USER_ID correspondente>

```
typedef struct model{
    GHashTable * flight_by_id;
    GHashTable * flight_by_origin;
    GHashTable * reservation_by_id;
    GHashTable * reservation_by_hotel;
    GHashTable * user_by_id;
} *MODEL;
```

struct

**FLIGHTS**:

Esta vai armazenar informações sobre voos:

- ID
- Companhia Aérea
- Modelo de Avião
- Origem do Voo
- Destino do Voo
- Número total de assentos
- Número total de passageiros
- Data de Partida
- Data de Chegada

struct **RESERVATION**:

esta struct vai armazenar informações sobre uma reserva:

- ID
- ID do user
- ID do hotel
- Nome do hotel
- Estrelas do hotel
- Data de início da reserva
- Data de fim da reserva
- Se inclui pequeno-almoço ou não
- Número de noites passadas
- Preço por noite
- Taxa, consoante a cidade em que o hotel se encontra
- Avaliação dada

struct **USER:**

- ID
- Nome do Utilizador
- Sexo do Utilizador
- Idade do Utilizador (derivada de uma data de nascimento)
- Código do País
- Lista de Voos (glist)
- Lista de Reservas (glist)
- Total Gasto pelo User
- Estado da Conta (Active ou Inactive)
- Data de Criação da conta
- Passaport

Depois, temos três ficheiros de catálogos, que contêm funções para gerenciar catálogos, possibilitando a procura e gerenciamento das entidades e dos dados dependendo do critério requerido.
Embora só tenhamos 3 ficheiros de catálogos: “flightCatalog.c” “reservationCatalog” e “userCatalog”, estes vão gerenciar as 5 hashtables existentes. Esta divisão existe porque embora hajam 5, haverá 2 manipuladas consoante os voos, 2 consoante as reservas e 1 consoante o user.

Consoante a imagem acima apresentada, ainda temos também um interpretador que processa o ficheiro input.txt, e invoca as queries correspondentes a cada linha de input.

No ficheiro responsável por iniciar o model, temos também funções que tem como objetivo processar cada ficheiro de input, user, flight e reservations, analisando consoante as validações de campo, dependendo do tipo de input recebido, dadas pelo parser.c.

O queries.c é o módulo responsável por modular e agrupar data consoante o pedido pela específica query.

E para concluir os ficheiros todos temos uma utility.c, que é um módulo com funções auxiliares.

O design modular e o uso de hash tables tornam o sistema extensível, tendo esta nossa arquitetura como meta otimizar a manipulação e gerenciamento de dados, garantindo que estes sejam eficientes.

# Estratégia

Como antes dito, abordamos este projecto de um ponto de vista modular, onde cada entidade tem a sua própria estrutura e conjunto de operações.
A escolha de hash tables foi devido à possibilidade de acesso rápido e eficiente a informação.
A biblioteca “GLib” foi e é algo muito importante neste projecto, pois possibilitou guardar dados de uma maneira que mais tarde facilitasse o acesso a esta mesma.
As estruturas de dados desta biblioteca que foram usadas foram as GHashTable’s  as GList’s, e com estas estruturas, as funções a elas associadas.
O facto da maior parte das funções de acesso e de inserção de informação já estarem implementadas facilitou o processo deste projecto.

# Limitações / Aspectos a melhorar

**Melhorar Modularidade / Encapsulamento**

Através da criação de mais funções de manipulação ao nível dos catálogos e dos próprios módulos das structs que definimos.

**Catálogos Extra**

De momento temos a informação organizada por um total de 5 catálogos que permitiram rapidamente executar cada uma das queries implementadas nesta primeira fase. No entanto, prevemos ser necessária a criação de mais um par de catálogos que apoiem a resolução de queries a implementar na próxima fase.

**Memory Leaks**

Uma das maiores falhas que o código desenvolvido nesta primeira fase tem é a existência de memory leaks. Acreditamos que na segunda fase, com uma maior organização geral do código, a tarefa de tratar estas memory leaks seja mais fácil.
O nosso maior problema em relação a memory leaks foi propositadamente causado, isto é, nós sabemos onde é que a memória foi perdida, mas ESCOLHEMOS não libertamos pois estava a causar problemas com na execução do problema, e não conseguimos entender o problema a tempo da entrega, então escolhemos entre o programa correr, e/ou diminuir memory leaks