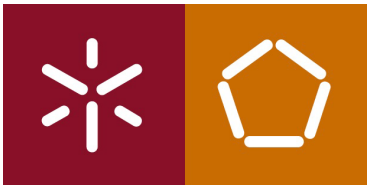


**UNIVERSIDADE DO MINHO**

# “Sistema de gestão e consulta de recomendações de negócios na plataforma Yelp”

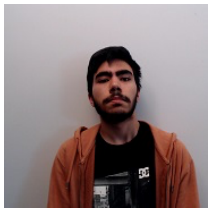
## Grupo 17

Mestrado Integrado em Engenharia Informática

Laboratórios de Informática III


Catarina Gonçalves
A93259

Inês Marinho
A82358



Miguel Dias
A93197


Braga, 6 de Maio de 2021

# Índice

# Introdução

O presente relatório foi concebido no âmbito da unidade curricular “Laboratórios de Informática III”, do segundo ano do Mestrado Integrado em Engenharia Informática e tem como objetivo a descrição detalhada de todas as decisões tomadas aquando da implementação realizada do projeto proposto, tendo por base a linguagem C.

Começa-se neste capítulo por fazer a descrição do problema proposto e os objetos que se pretendem atingir.

No capítulo seguinte, são explicadas ao detalhe as estruturas de dados utilizadas.

Seguidamente, é apresentado cada um dos módulos necessários para a implementação do projeto, tal como as estratégias adotadas para a resolução das *queries* e ainda estratégias de otimização que se teve em conta no desenvolvimento.

Por conseguinte, são apresentados os dados que se obtiveram da realização de testes de performance.

E por fim, o capítulo da conclusão, onde se aborda as dificuldades sentidas pelo grupo e a respetiva critica daquilo que poderia ser melhorado.

# Descrição do problema

O problema proposto é a implementação de um Sistema de Gestão e Consulta de Recomendações de Negócios na Plataforma Yelp e tem por base três ficheiros .csv, cada um destes com diferentes tipos de informações associadas.

O principal objetivo deste, é a implementação das funcionalidades enunciadas, assegurando a estas eficiência na sua realização devido ao grande volume de dados com o qual tem que lidar. São ainda objetivos deste projeto, os princípios de programação, tal como a modularidade e encapsulamento de dados, a criação de código reutilizável e testes de performance de forma a garantir um bom projeto de software.

# Estruturas de dados

Para o desenvolvimento do presente projeto as estruturas base são as seguintes:

### Users

A estrutura *User* agrega toda a informação relacionada a um utilizador, tal como o id e o nome deste, em strings e os amigos caso eles existem, numa *GList*.

```
struct user{
    char *id;
    char *name;
    GList *friends;
};
```

### Business

A estrutura *Business* tem como objetivo agregar toda a informação recebida de um negócio. Optou-se por guardar o id, o nome, a cidade e o estado em strings e as categorias numa *GList*.

```
struct business{
    char* id;
    char* name;
    char* city;
    char* state;
    GList* categories;
};
```

### Reviews

A estrutura *Review* tem como objetivo agregar todas as informações relativas a uma *review*, decidiu-se guardar estas informações em strings, inteiros e *floats*.

```
struct review{
    char *review_id;
    char *user_id;
    char *business_id;
    float stars;
    int useful;
    int funny;
    int cool;
    char *date;
    char *text;
};
```

# Estruturas de dados complementares

Para auxiliar na implementação das *queries*, foram definidas as seguintes estruturas de dados.

### Query4

A estrutura *q4* serve como estrutura auxiliar à função principal da *query4* onde cada entrada tem o id e o nome do negócio associados.

```
struct q4{
    char *id;
    char *name;
};
```

### Info_Negocio

A estrutura *info_negocio*, tal como o nome indica possui a informação de cada negócio, guardado apenas o seu id e o seu nome. Informação necessária para a *query* 5.

```
struct info_negocio{
    char *id;
    char *nome;
};
```

### SGR

A estrutura *SGR* é única, isto é, só existe uma durante toda a execução do programa. Esta guarda o nome que o utilizador lhe deu, os 3 catálogos em *HashTables*, uma *HashTable* reunida com todas as *TABLE's* existentes e 3 *HashTables* cada uma obtida quando se chama *queries* evitando assim refazer essas mesmas estruturas por cada *query*.

```
struct sgr{
    char *sgr_name;
    GHashTable* users;
    GHashTable* negocios;
    GHashTable* reviews;
    GHashTable* tables;
    GHashTable* fullCityInfo;
    GHashTable* fullBusinessInfo;
    GHashTable* fullCategoryInfo;
};
```

## TABLE

A estrutura *TABLE* tem como objetivo ser genérica o suficiente para guardar tanto uma lista de dados, como um dado. O que leva a estrutura a ser constituída pelo seu nome, o número de colunas e de linhas, e finalmente a efetiva tabela: uma *GList* cujos elementos correspondem às linhas da tabela, isto é, um *array* com tantas posições quanto o número de colunas.

```
struct table{
    char* variable;
    GList* tabela;
    int colunas;
    int linhas;
};
```

## CampoNegocio

A estrutura *CampoNegocio* foi criada com o intuito de guardar todas as informações de cada negócio numa única estrutura não deixando a informação dispersa. Isto é, guarda o seu id, o nome, a cidade, o estado, uma *GList* das suas categorias, o número de *reviews* que lhe foram feitas, a média de estrelas que lhe foram atribuídas e finalmente uma *GList* com o conjunto de estrelas que recebeu.

```
struct campoNegocio{
    char* business_id;
    char* business_name;
    char* city;
    char* estado;
    GList* categories;
    int nrReviews;
    float mediaEstrelas;
    GList* todasEstrelas;
};
```

## CampoUser

O *CampoUser* tem como objetivo ser único para cada *user*, contendo o seu nome, um estado que permite verificar se deu mais que uma *review* em diferentes estados, e um campo extra de verificação desta última afirmação.

```
struct campoUser{
    char* user_id;
    char* state;
    int bloqueado;
};
```

## CampoCategoria

A estrutura Campo*Categoria* tem o propósito de ser única para cada categoria de todos os negócios, isto é, cada uma destas contém o seu nome e uma *GList* de negócios, ou seja, uma estrutura *CampoNegocio*, associados à respetiva categoria.

```
struct campoCategoria{
    char* category;
    GList* campos;
};
```

## CampoCidade

A estrutura Campo*Cidade* tem como finalidade guardar para cada cidade, o seu nome e uma *GList* de *CampoNegocio*, estes os quais se situam nessa mesma cidade.

```
struct campoCidade{
    char* cidade;
    GList* campos;
};
```

# Implementação

## Módulos

**Apresentador** – O módulo *Apresentador* contém um menu onde são apresentadas as opções para o módulo *interpretador* e uma função denominada de "paginacao" servindo como mini interpretador, permitindo a leitura organizada em páginas das informações do tipo *TABLE* devolvidas pelas *queries*.

**Business** – O módulo *Business*, contém a leitura dos utilizadores guardando-os numa *GHashtable* e a sua respetiva libertação.

**Interpretador** – O módulo *Interpretador* tem como fim interagir com o utilizador de forma a aceder e manipular dados como desejar, de acordo com as possibilidades que lhe são fornecidas.

**QueriesAux** – O módulo *queriesAux* serve tal como o nome indica para guardar funções auxiliares das *queries*, assegurando uma implementação mais organizada das funções de *queries* principais no módulo *SGR.*

**Reviews** – O módulo *Reviews*, contém a leitura dos utilizadores guardando-os numa GHashtable e a sua respetiva libertação.

**Sgr** – O módulo *SGR*, contém a todas as *queries* tal como todas as funções necessárias para aceder às estruturas de dados *table* e *sgr*.

**Stats** – O módulo de *Stats,* é onde existe a conexão entre os catálogos de *business* e *reviews*. Isto é, é feita uma abordagem dos negócios de diferentes formas de acordo com o que é necessário para resolver cada *query* e inclui também todas as funções necessárias para aceder às estruturas de dados *campoNegocio*, *campoUser*, *campoCategoria* e *campoCidade*.

**Tasks** – A finalidade do módulo *Tasks* tem por base o suporte à interpretação de comandos, sendo a ponte entre o input do utilizador e o comando a ser realizado.

**Users** – O módulo *Users*, contém a leitura dos utilizadores guardando-os numa GHashtable e a sua respetiva libertação.

## *Queries*

### *Query 1*

A função desta *query* é fazer o carregamento de todas as estruturas de dados sutilizadas durante a execução do programa, com o auxílio das funções “readUsers”, “readBusiness” e “readReviews”.

### *Query 2*

A *query 2* tem como função procurar todos os negócios começados pela letra fornecida pelo utilizador. Para tal foram criadas duas funções auxiliares, a função "compareChar" que compara o primeiro carater de cada string e determina se são iguais, tendo em conta que a procura não será "case sensitive" e, a função "searchletra" que procura numa *GList* de estruturas do tipo "Business" o negócio começado pela letra fornecida, inserindo o nome deste numa *GList*.

### *Query 3*

A função desta *query* é dado um id de negócio, determinar a sua informação como o nome, a cidade, o estado, as *stars* e o número total de *reviews*. Usando a *GHashTable* retornada da função “criaTableNegocios”, consegue-se encontrar a estrutura *campoNegocio* associada ao id de negócio, que contém toda a informação relacionada com esse negócio, se o id recebido for um id válido, acede-se às informações que a *query* pede, com o auxílio dos *gets* da estrutura *campoNegocio*, tais como, o nome, a cidade, o estado, as estrelas, na qual é considerada a média de estrelas que cada negócio possui e o número total de *reviews* que este apresenta.

### *Query 4*

A *query* 4 tem como função procurar todos os negócios ao qual um utilizador fez review, tendo como informação associada o seu id e o nome. Para tal, foi criada a estrutura auxiliar "q4" de modo a guardar esses dois parâmetros e foram também criadas duas funções auxiliares, a função "procura" que pesquisa numa GList de estruturas "Review" o id de negócio associado ao id de utilizador fornecido colocando-o num array de strings denominado de "nome", tendo como string final a palavra "fim" de modo a ajudar a outra função auxiliar "procuranomebus" a encontrar o fim do *array*. A função "procuranomebus" procura numa *GList* de estruturas do tipo "Business" os nomes dos negócios associados aos ids do array "nome", sendo depois estas duas informações colocadas numa estrutura auxiliar *q4* que é inserida posteriormente numa *GList*.

### *Query 5*

O objetivo desta *query* é dado um número de estrelas e uma cidade, determinar a lista de negócios que possui esse número ou mais estrelas. Apoiando a resposta a esta *query* nas funções “criaTableNegocios” e “criaTableCidades”, consegue-se obter uma *GHashTable* cujas *keys* são cidades e com isso facilmente aceder à cidade passada como argumento. Se esta for uma cidade válida, percorre-se todos os negócios associados a esta e recolhe-se apenas aqueles que possuem o campo da média das estrelas igual ou superior ao argumento dado.

### *Query 6*

Para determinar os top N negócios de cada cidade recorreu-se à função “criaTableNegocios”, que percorre todas as *reviews* e agrega cada negócio com a sua informação (*CampoNegocio*). Após termos todos os negócios agrupados percorre-se esse mesmo grupo e organiza-se os negócios por cidades (CampoCidade). Percorrendo-se

finalmente esse grupo e eliminando-se todos os negócios que não estejam no top N de negócios de cada cidade.

### *Query 7*

Para determinar todos os *users* internacionais, utiliza-se novamente a função "reviewHash" para agrupar todos os negócios e a sua informação. Seguidamente é percorrida essa *GHashTable* com a função "criaTableEstados", esta vai agrupar os negócios todos em ordem aos *users*. Partindo disto, eliminam-se todos os *users* que não estejam bloqueados, ou seja, que fizeram *reviews* em mais de um estado.

### *Query 8*

Para determinar os top N negócios por uma categoria fornecida em concreto, utilizou-se a função "reviewHash" para obter todas as *reviews* em ordem aos negócios. Seguidamente criou-se uma *HashTable* com os negócios em ordem às categorias, isto é, em campos Categorias. Finalmente procurou-se a categoria que passado como parâmetro, retornando apenas a *table* com os top N negócios.

### *Query 9*

A *query* 9 tem como função indicar os ids de *review* onde é mencionada a palavra fornecida no campo "text". Para tal foram criadas duas funções auxiliares, a função "strstrpre" que através do uso da função "strstr" procura a palavra fornecida e averigua se o apontador devolvido pela função "strstr" aponta para uma palavra completa e não uma sub-string, pois há certas palavras como por exemplo a palavra "pensive" que faz parte da palavra "expensive". A função "strstrpos" é muito semelhante à função anterior verificando neste caso que o que vem após o fim da palavra não é um asterisco ou um hífen, por exemplo. Se for verificado que as duas funções auxiliares retornaram verdadeiro para cada campo "text" então o id de utilizador associado é colocado numa *GList.*

## Estratégias para otimizar *queries*

Uma das estratégias implementadas pelo grupo de forma a otimizar as *queries*, foi a troca da estrutura na qual se encontravam as *reviews*. Primeiramente decidiu-se a implementação desta em *arrays*, mas posteriormente trocou-se para G*HashTable,* uma vez que possui endereçamento direto consoante a *key* levantado a um acesso mais eficiente dos dados e não abarca tantos problemas de memória.

# Testes de *performance*

De forma a testar o tempo de execução de todas as funcionalidades propostas, foram realizados alguns testes de *perfomance*, na máquina com um processador AMD Ryzen 5 3550H, quad-core de frequência base 2.1GHz, com 16GB de RAM e os resultados obtidos encontram-se na tabela abaixo, sendo os tempos apresentados, em segundos.

| | **Tempos(s)** |
|---|---|
| ***Query 1*** | 9 |
| ***Query 2*** | < 1 |
| ***Query 3*** | 18 |
| ***Query 4*** | 2 |
| ***Query 5*** | 18 |
| ***Query 6*** | 18 |
| ***Query 7*** | 1 |
| ***Query 8*** | 17 |
| ***Query 9*** | 1 |

# Conclusão

Perante o desenvolvimento deste projeto encontramo-nos face a grandes dificuldades iniciais, isto pelo grande volume de dados, mas que se conseguiram resolver de uma forma relativamente eficaz.

O grupo considera que o sistema gerado responde a todos os pontos enunciados, conseguindo assim, atingir a maioria dos objetivos propostos, sendo que alguns deles não foram concretizados, tais como, a utilização do *Valgrind*, uma vez que a nossa falta de utilização desta ferramenta desde o início do desenvolvimento do projeto prejudicou bastante o resultado final, devido às várias *memory leaks* que se tentou resolver por último, mas que se tornou difícil devido à complexidade do código já feito. A interface, que perante um erro persistente na função *strlen* e nas seguintes tentativas de substituição dessa mesma função, resultou numa apresentação de uma tabela desalinhada visualmente. A libertação de estruturas secundárias, que com o objetivo de minimizar tempos de execução recorreu-se à criação de estruturas que permitiam ter acesso a dados muito facilmente, o que levou no entanto, a não se conseguir finalizar a sua implementação, devido a *memory leaks* na sua libertação de espaço e por fim, o comando *toCSV* e *fromCSV*, que devido à nossa última mudança no interpretador, para aceitar comandos do utilizador sem a necessidade de *enter's*, prejudicou os comandos *toCSV* e *fromCSV*, pois não conseguimos colocar como aceite o ";" como um delimitador reconhecido.

Apesar de alguns desvios em relação ao nosso objetivo principal, conseguiu-se perceber a vantagem de ler e manipular dados de larga escala, desenvolver as nossas capacidades em resolver conflitos e finalmente, uma boa sincronia permitiu a todos desenvolver um bom espírito de trabalho em equipa.