Laboratórios de Informática III

Grupo 43
62608 - Marco Sousa
88000 - Gerson Júnior
93271 - José Malheiro

18 de maio de 2021

Resumo

Este projeto permitiu desenvolver competências de **Engenharia de Software**, nomeadamente *modularidade, encapsulamento, reutilização e escalabilidade* de programas.

Foi utilizado a arquitetura Model-View-Controller, juntamente com a linguagem de programação C.

Através das estratégias propostas, programou-se um **Sistema de Gestão de Reviews**, incluíndo interação I/O, carregamento e gravação de informação e gestão interna de informação otimizada através de estruturas de dados.

1 Introdução

O presente relatório foi redigido no âmbito da unidade curricular (UC) Laboratórios de Informática III e remete-se à elaboração de um projeto na linguagem de programação C para um **Sistema de Gestão de Reviews**.

A construção do projeto teve como referência a orientação dos docentes da UC e principal objetivo de desenvolver conceitos de modularidade, encapsulamento, construção de código reutilizável, otimização na escolha de estruturas de dados e reutilização. Para além destes, permitiu o aprofundamento sobre o desenvolvimento de programas na linguagem C.

2 Estrutura do Projeto

Conforme apresentado pela equipa docente, optou-se por um modelo MVC (*Model View Controller*). Este tipo de arquitetura compreende, essencialmente, três camadas de desenvolvimento.

- Model
  - Estruturas de dados;
  - Lógica
- Controller
  - Ponto de entrada do utilizador
  - Comunica com o model para gerar dados
- View
  - É atualizado pelo controller
  - Apresenta a informação ao utilizador
  - É o ponto de saída de informação

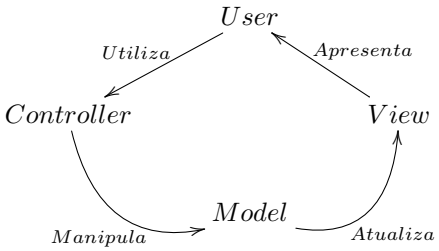
A interação com o utilizador foi conseguida através de um interpretador de comandos que efetua *parsing* do comando terminado pelo caracter "**;**". Caso o comando, doravante designado *query*, esteja incluído numa lista limitada de opções, é executado.

A execução tem um de três propósitos:

1. Atribuir o resultado da *query* a uma variável

2. Apresentar o conteúdo de uma variável sob a forma de tabela

3. Exportar a informação contida numa variável do tipo **TABLE** para um ficheiro *CSV*.



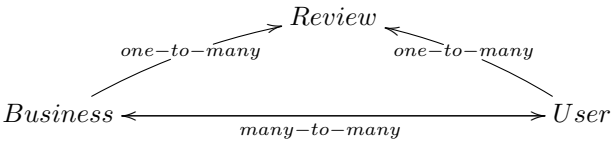
2.1 Estruturas de Dados

As estruturas de dados criadas para representar a informação necessária para a gestão das reviews, foram Business, Review, User.

Cada uma das estruturas foi desenvolvida de forma a garantir o encapsulamento da informação, pelo que a estrutura é desconhecida por quem utiliza a API.

Assim, é disponibilizado um conjunto de funções para inicializar, destruir ou aceder à informação (designados init, free, getters & setters).

As relações entre as estruturas podem ser descritas pelo esquema:



2.1.1 Business

```
typedef struct business
{
        char *business_id;
        char *name;
        char *city;
        char *state;
        GArray *categories;
        GArray *review_ids;
        float all_stars;
} BNode;
typedef struct business *Business;
```

Estrutura utilizada para guardar a informação de cada empresa - ID, nome, cidade, estado e categorias. Ver 4.1.

Por forma a otimizar as queries, optou-se por criar uma lista de identificadores de reviews que estão associados a esta empresa. Adicionou-se, ainda, o campo `float all_stars` que corresponde ao somatório das estrelas de cada review associada à empresa.

Desta forma, para calcular o número de estrelas, bastará dividir o campo *all_stars* pelo tamanho da lista *review_ids*.

2.1.2 User

```
struct user
{
        char *user_id;
        char *nome;
        GArray *review_ids;
        GArray *friends;
} UNode;
typedef struct user *User;
```

Armazena a informação de cada utilizador, como o id, nome, review_ids, friends. Ver 4.2

Tal como no exemplo anterior, optou-se por adicionar um campo `GArray *review_ids` para otimizar a pesquisa de reviews associadas a um utilizador.

2.1.3 Review

```
struct reviews
{
        char *review_id;
        char *user_id;
        char *business_id;
        float stars;
        int useful;
        int funny;
        int cool;
        char *date;
        GArray *text;
} NReview;
typedef struct reviews *Review;
```

Armazena a informação de cada review. Ver 4.3.

Permite, ainda, fazer a ligação entre utilizadores, comentários e empresas.

2.1.4 TABLE

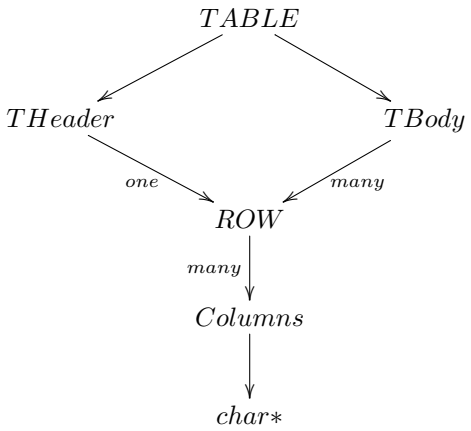
Para uniformizar o tipo de saída das queries do SGR (2.3), criou-se um módulo que se reflete como sendo uma tabela. Através desta estratégia, pretende-se que a impressão para o utilizador seja realizada de uma forma linear, fluída e com o mínimo de erros.

```
typedef struct row
{
        TColumns columns;
        guint size;
} RNode;
struct table
{
        THeader theader;
        TBody tbody;
} TNode;
```

Cada tabela é constituída por um cabeçalho - THeader - e um corpo - TBody. O cabeçalho corresponde a uma linha. É, portanto, um *alias* para a estrutura ROW.

Por sua vez, o corpo corresponde a uma lista de linhas, i.e. lista de ROW.

A cada linha está associada uma lista de colunas, sendo cada coluna do tipo `char *`.



A construção da tabela corresponde à sua inicialização, definir o cabeçalho e adicionar linhas. Todas as linhas têm *obrigatoriamente* o mesmo número de colunas.

Para além das funções de construção, a API (Ver 4.4) apresenta funções para filtrar a tabela a partir do número ou do nome da coluna e utilizando um operador:

```
typedef enum op
{
        LT = -1,
        EQ,
        GT
} OPERATOR;
```

Conforme o operador que a função recebe no argumento, vai aplicar a filtragem à tabela, gerando uma nova tabela com os valores que são menores, iguais ou maiores (LT, EQ, GT) que o valor de filtro.

Pode, ainda, adicionar à tabela uma linha de erro. Por exemplo, caso a tabela query devolva zero resultados, pode-se adicionar essa linha para melhorar a interação com o utilizador.

Por último, este módulo implementa, ainda, a construção de uma tabela a partir de um ficheiro CSV, através da função `TABLE csv_to_table(char *file_path, char *delim)`.

2.1.5 Hashtable

Como conseuquência da degradação exponencial da performance da aplicação quando se utilizava uma simples lista, sem ordenação ou chave de acesso, optou-se por criar uma estrutura de dados **Hashtable** que tivesse como objetivo melhorar o tempo de execução.

A decisão pela tabela de *hash* deve-se ao facto que as funções de acesso (leitura, atualização, remoção), executam em tempo constante O(1).

Assim, desenvolveu-se uma API com total abstração do tipo de dados e garantindo o encapsulamento da estrutura.

```
int _initHashTable(HashTable **, int size);
int _clone(HashTable **, HashTable **, int()(void *, void *), int()(void *, int));
int _lookup(HashTable *, void *, void **, int()(void *, void *), int()(void *, int));
void *_findValue(HashTable *, void *()(void *), void *, int()(void *, void *));
GArray *_filter(HashTable *, void *()(void *), void *, int()(void *, void *));
GTree *_hash_to_tree(HashTable *, void *()(void *), void *()(void *),
         int()(const void *, const void *));
int _update(HashTable **, void *, void *, int()(void *, void *), int()(void *, int));
int _length(HashTable *);
int _remove(HashTable **, void *, int()(void *, void *), int()(void *, int),
         void()(void *), void()(void *));
void _free_hashtable(HashTable **, void()(void *), void *()(void *));
```

_initHashTable Inicializa a tabela de hash;

_clone Efetua uma cópia da tabela;

_lookup Procura por um valor através de uma chave. Caso encontre, coloca-o numa das variáveis passada por referência;

_findValue Procura um valor aplicando uma função de comparação após uma função de extração (devolve o primeiro que encontrar);

_filter Semelhante à função anterior mas coloca num array todos os valores cuja comparação seja *true*;

_hash_to_tree Converte uma tabela de hash numa árvore binária balanceada, utilizando uma função para obter a chave e outra para o valor. É também fornecida uma forma de comparar as chaves, para permitir inicializar a árvore;

_update Atualiza ou adiciona um valor à tabela;

_length Calcula o número de elementos contidos numa tabela;

_remove Remove um valor da tabela;

_free_hashtable Destrói toda a tabela e a sua informação. Precisa de receber uma função para libertar a chave e os valores.

2.2 Repositórios de Informação - Catálogos

Por forma a garantir o encapsulamento da informação, foi criado uma estrutura designada repositório para gerir cada modelo.

- BusinessRepository
- UserRepository
- ReviewRepository

2.2.1 BusinessRepository

```
struct business_repository
{
        HashTable *business;
        GTree *group_by_city;
} BRNode;
```

Nesta estrutura, ficou armazenado toda a informação sobre business carregada a partir de um ficheiro. Ver 4.6.

O parâmetro `HashTable *business` corresponde a uma tabela de hash em que a chave é o *business_id* e o valor é um `struct business *`. Ao utilizar uma estrutura deste género, permite que a procura, inserção ou remoção de um *business* execute em tempo constante.

Por outro lado, o parâmetro `GTree *group_by_city` surgiu como uma necessidade no decorrer desenvolvimento da aplicação. Ver 2.3.5 .

2.2.2 UserRepository

```
struct user_repository
{
        HashTable *user;
} USRNode;
```

Tal como explicado no ponto anterior, optou-se por armazenar os utilizadores sob a forma de tabela de hash. Ver 4.7.

2.2.3 ReviewRepository

```
struct review_repository
{
        HashTable *review;
} RWNode;
```

Utilizado a mesma estrutura que referido anteriormente. Ver 4.8.

2.2.4 Environment Repository

```
struct env_repository
{
        HashTable *variables;
        SGR sgr;
} ENVNode;
```

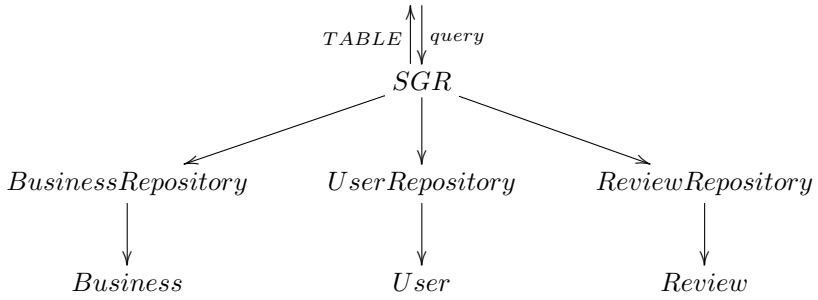
Para gerir as variáveis de ambiente do interpretador e manter o encapsulamento e modularidade da aplicação, desenvolveu-se um *model* que permitiu facilitar a organização do acesso e atualização das variáveis.

Para tal, salienta-se as funções:

```
EnvRepository init_env_repo();
TABLE get_table_by_var(EnvRepository er, char *var);
int add_variable(EnvRepository er, TABLE t, char *var);
void set_env_sgr(EnvRepository ev, SGR sgr);
SGR get_env_sgr(EnvRepository ev);
```

Através das funções apresentadas - ver 2.2.4, é possível adicionar ou atualizar variáveis e atualizar o SGR.

2.3 Sistema de Gestão de Reviews - SGR



Por forma a relacionar a informação, mantendo a modularidade e o encapsulamento, foi criado um módulo designado SGR que permite executar um conjunto de queries.

Estas, por sua vez, podem utilizar um ou mais repositórios para obter o resultado desejado. Ver SGR API - 4.10.

O resultado de cada query é retornado sobre a forma do tipo TABLE (ver 2.1.4). Esta regra apenas não se aplica no caso de carregamento do SGR (`SGR loadSGR(char*, char*, char*)`).

2.3.1 Query 1 - loadSGR

```
SGR loadSGR(char *busPath, char *userPath, char *revPath);
```

O objetivo desta query é carregar para a *heap* os utilizadores, empresas e reviews a partir de ficheiros do tipo CSV.

Utiliza-se uma API designada DB (*database* - Ver 4.11) que dado o caminho para os ficheiros, efetua a construção das estruturas necessárias para armazenar os valores encontrados no ficheiro.

A ordem de carregamento foi definida da seguinte forma:

loadBusiness Carrega todos os business, sendo, simultaneamente, gerada uma árvore ordenada por cidade, em que a cada nodo está associado uma lista de Business lá sediados.
Esta necessidade surgiu no desenvolvimento da query 6 (2.3.6).

loadUsers Carrega todos os utilizadores

loadReviews Carrega os reviews e, durante a execução, atualiza o repositório de Business e de Users, associando a cada um o id do review a ser processado.

Por forma a cumprir os requisitos definidos pela equipa docente, os reviews que estejam associados a user_ids ou review_ids que de estruturas que não existem, são descartados. Consequentemente, este repositório é o último a ser carregado.

2.3.2 Query 2 - business_started_by_letter

```
TABLE business_started_by_letter(SGR sgr, char letter);
```

Para manter o encapsulamento, é utilizado uma função definida no repositório dos *business* que consiste na chamada de uma função filter que retorna um array de business que começam pela letra em questão.

Este algoritmo tende para uma execução em tempo linear.

```
GArray *found = _filter(br->business, (void *)get_business_name, letr, has_prefix;
```

Depois de obtido o array de business, é adicionado cada um dos registos à *TABLE* construída no **SGR**.

2.3.3 Query 3 - business_info

```
TABLE business_info(SGR sgr, char *business_id);
```

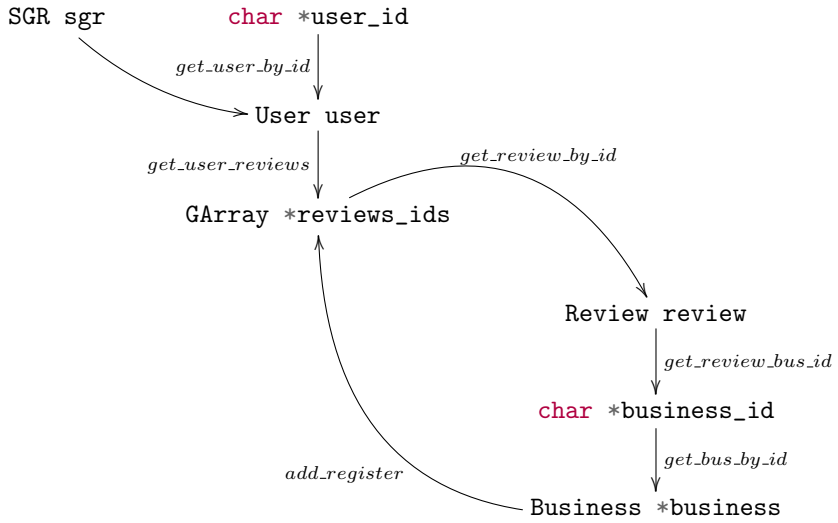
A execução deste algoritmo é efetuada em tempo constante. Consiste no acesso direto a uma hashtable a partir da sua chave.

Caso a chave procurada não exista, é adicionado uma linha de erro à tabela de retornada.

2.3.4 Query 4 - business_reviewed

```
TABLE businesses_reviewed(SGR sgr, char *user_id);
```

A construção deste algoritmo consistiu em 3 fases, podendo ser representada sobre a forma gráfica:



Dado um user_id, faz-se a pesquisa por esse utilizador.

Caso exista, passará a estar disponível uma lista de review_ids desse User.

Iterando essa lista de review_ids, pode-se obter o bus_id associado e utilizar a função *get_bus_by_id* para obter o Business.

Quando o Business é encontrado, estão reunidas as condições para o acrescentar à tabela que será retornada pela query.

2.3.5 Query 5 - get_business_by_city

A versão inicialmente desenvolvida para este algoritmo, ainda não tinha disponível a árvore de business ordenado por cidade.

Era necessário percorrer toda a hashtable para obter os business associados a uma cidade e, após tal iteração, construir a tabela a retornar pelo **SGR**. Mediante uma breve análise, facilmente se pode concluir que a primeira parte deste algoritmo executava em O(N) sendo N o número de reviews.

Através da criação de uma árvore ordenada por cidade, em que os valores são um GArray de Business, assim como a respetiva inserção na altura do carregamento das reviews, permitiu uma otimização tal que a primeira porção passou a ser executada em O(log M), sendo M o número de cidades. De notar que número de cidades $\ll$ número de reviews, pelo que a otimização desta query é muito superior à mera comparação entre O(N) e O(log M).

2.3.6 Query 6 - top_business_by_city

```
TABLE top_businesses_by_city(SGR sgr, int top);
```

A query 6 também usufrui dos benefícios de ter a informação dos Business por cidade previamente organizada. Como tal, esta consiste em iterar a árvore dos Business agrupados por cidade e aplicar uma função de ordenação a cada lista.

Após a ordenação, retorna os primeiros **top** elementos de cada lista.

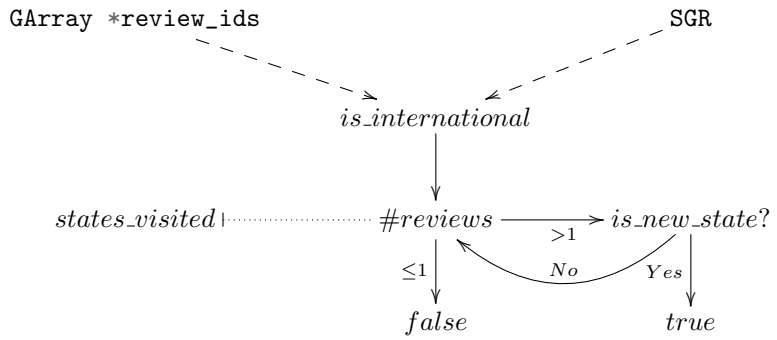
Neste caso, é ordenado por número de estrelas.

2.3.7 Query 7 - international_users

```
TABLE international_users(SGR sgr)
```

O algoritmo por trás desta query, efetua uma travessia na tabela de utilizadores, aplicando a cada elemento (`User`) a função **is_international**.

Nesta última, pode-se encontrar o *cérebro* do filtro.



states_visited Lista que armazena os estados já visitados.

#reviews Número de reviews associadas ao utilizador.

2.3.8 Query 8 - top_business_with_category

```
TABLE top_business_with_category(SGR sgr, int top, char *category);
```

A estratégia utilizada foi semelhante à query anterior. Neste caso, percorre-se a tabela de business aplicando um filtro por categoria, obtendo-se lista de business que pertencem à categoria procurada.

Posteriormente, é aplicado uma função de ordenação por ordem decrescente de número de estrelas e adicionado à **TABLE** os **top** primeiros elementos.

Este algoritmo poderia ser otimizado se fosse utilizada a estratégia da query 5 - 2.3.5, i.e. durante o carregamento dos business, criar uma estrutura auxiliar que armazenasse os **Business** por categoria. Tal otimização, conforme explicado anteriormente, aumentaria exponencialmente a *performance* da query.

2.3.9 Query 9 - reviews_with_word

```
TABLE reviews_with_word(SGR sgr, int top, char *word);
```

Semelhante à estratégia utilizada nas queries anteriores, optou-se por percorrer todos os reviews e, em cada, executar a função `char *strstr` para procurar ocorrências da palavra procurada.

Este algoritmo retorna, não só as palavras completas iguais ao valor, mas também aquelas em que a está contida.

Uma forma de otimização futura desta query, consiste em retornar apenas as reviews que tenham a palavra estritamente igual à pesquisada.

Caso esta query seja utilizada com frequência, deveria ponderar-se gerar uma árvore de texto, ordenado por palavras. Desta forma, é possível melhorar o tempo de execução de O(N_reviews*N_palavras) para O(N_reviews * log N_palavras).

2.4 Interpretador de comandos

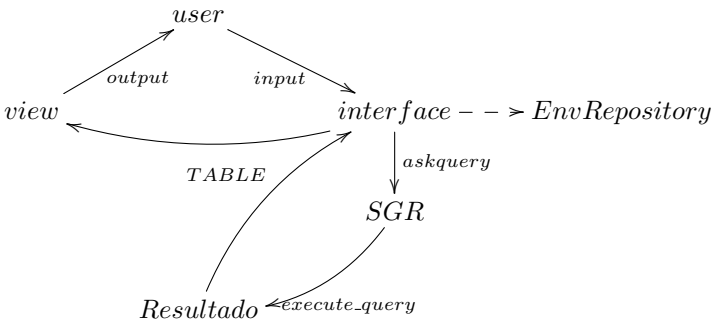
O interpretador de comandos tem como objetivo ler o input do utilizador, efetuar *parsing* do comando, executar e devolver algum tipo de **outcome**. Conforme o tipo de comando, poderá originar três situações:

void É apresentada a informação de uma variável no ecrã, sob a forma de tabela iterável ou importado/exportado informação de um ficheiro CSV para TABLE.

TABLE Após execução de uma **query**, é atribuído a uma variável de ambiente o seu resultado, sob a forma de tabela.

SGR Gerado uma estrutura SGR, carregado a partir de três ficheiros do tipo CSV contido num *path* relativo ou absoluto

Para auxiliar esta estrutura, desenvolveu-se uma API designada Variáveis de Ambiente (Ver 2.2.4 - EnvRepository). Desta forma, facilitou a gestão das variáveis num módulo próprio, garantindo a modularidade e escalabilidade da aplicação.



2.5 Views

Nesta secção, será descrito os módulos relacionados com a apresentação de informação ao utilizador.

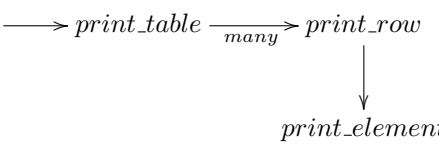
2.5.1 Table View

Como consequência da uniformização do *output* do SGR (Ver 2.3), o presente módulo permite expor ao utilizador o resultado de qualquer query pedida.

Para tal, foi disponibilizado a API com a seguinte exposição:

```
void print_row(ROW row);
void print_table(TABLE t, unsigned int offset, int limit);
int print_table_to_csv(TABLE t, char *delim, char *file_path);
void pagination(TABLE t);
```

Tal como nos módulos anteriores, a impressão da tabela é efetuada de forma modular:



A API permite, ainda, a impressão para um ficheiro do tipo CSV. Este armazenará a informação contida na tabela, sendo cada linha no ficheiro correspondente a uma linha na tabela e as colunas separadas por um delimitador passado por argumento - `char* delim`.

Por forma a ser possível apresentar uma paginação da tabela, utilizou-se a função print_table que aceita um offset (valor a partir do qual vai ser apresentado resultados) e um limite de valores a apresentar de cada vez.

2.5.2 Interface

Neste módulo é possível encontrar, do lado do model, a lógica relacionada com o *parsing* e execução dos comandos e, do lado da view - **interface_view**, a componente de auxílio para impressão da informação no ecrã e interação com o utilizador.

3 Considerações Finais

Através da construção do **SGR**, foi possível adquidir competências que se encontravam alheias ao nosso conhecimento e desenvolver outras que, apesar de já terem sido estudadas a nível teórico, ainda não tinham sido implementadas.

As principais dificuldades, foram a arquitetura da aplicação, o encapsulamento e a construção do interpretador. As duas primeiras, foram ultrapassadas utilizando os métodos explicados pela equipa docente, nomeadamente a arquitetura **MVC**. Quanto ao interpretador foi, provavelmente, o maior desafio pois procurou-se, numa fase inicial, desenvolver uma estratégia sem utilização de `switch`. Para tal, tentou-se criar um objeto que armazenasse a informação relativa à execução de cada query e, posteriormente, utilizá-lo para executar as funções e efetuar *parsing* dos argumentos. No entanto, foi atingido um tamanho de complexidade para o qual não estavamos preparados, mediante o tempo disponível para a realização deste projeto.

A otimização dos algoritmos foi efetuada tendo por base os conceitos adquiridos em outras disciplinas, como **Algoritmos e Complexidade** e **Arquitetura de Computadores**. Desta forma,

foi priviligiado a análise algoritmica em prol do tempo de execução, sendo que este último varia consideravelmente entre computadores, devido à arquitetura que o próprio utiliza, nomeadamente estratégias de paralelismo e vetorização.

4 Anexos

4.1 Business API

```
/**
 * @brief Identificador do business
 *
 * @param b Apontador para business a que se vai buscar a info
 * @return Cópia do identificador do business
 */
char *get_business_id(Business b);

/**
 * @brief Nome do business
 *
 * @param b Apontador para business a que se vai buscar a info
 * @return Cópia do nome do business
 */
char *get_business_name(Business b);

/**
 * @brief Cidade do Business
 *
 * @param b Apontador para business a que se vai buscar a info
 * @return Cópia da cidade do Business
 */
char *get_business_city(Business b);

/**
 * @brief Número de estrelas do business
 *
 * @param b Apontador para business a que se vai buscar a info
 * @return Classificação em estrelas do Business [0..5]
 */
char *get_business_state(Business b);

/**
 * @brief Número de comentários do Business
 *
 * @param b Apontador para business a que se vai buscar a info
 * @return Número de comentários do Business
 */
GArray *get_business_categories(Business b);

/**
 * @brief Retorna a lista de ids associados a um Business
 *
 * @param b Business
 * @return GArray* Lista de review ids
 */
GArray *get_business_review_ids(Business b);

/**
 * @brief Calcula as stars do Business
 *
 * @param b Business
```

```
 * @return float Stars
 */
float get_business_stars(Business b);

/**
 * @brief Atualiza: Nome do Business
 *
 * @param b Apontador para Business a que se vai buscar a info
 * @param name Novo nome
 * @return 1 se sucesso; 0 se erro;
 */
int set_business_name(Business b, char *name);

/**
 * @brief Atualiza: Cidade do Business
 *
 * @param b Apontador para Business a que se vai buscar a info
 * @param city Nova cidade
 * @return 1 se sucesso; 0 se erro;
 */
int set_business_city(Business b, char *city);

/**
 * @brief Atualiza: Número de estrelas do Business
 *
 * @param b Apontador para Business a que se vai buscar a info
 * @param stars Nova classificação
 * @return 1 se sucesso; 0 se erro;
 */
int set_business_state(Business b, char *state);

/**
 * @brief Atualiza: Número de comentários do business
 *
 * @param b Apontador para Business a que se vai buscar a info
 * @param counter quantidade a adicionar / remover;
 * @return 1 se sucesso; 0 se erro;
 */
int set_business_categories(Business b, char *categories);

/**
 * @brief Adiciona Stars a um Business
 *
 * @param b Apontador para Business que vai ser alterado
 * @param stars float com as Stars a serem adicionadas
 */
void add_business_stars(Business b, float stars);

/**
 * @brief Adiciona uma categoria à Business (caso não exista)
 *
 * @param b Apontador para Business a que se vai buscar a info
 * @param category Categoria a ser adicionada
 * @return 1 se sucesso; 0 se erro; -1 caso já existe;
 */

int add_business_category(Business b, char *category);
/**
 * @brief Adiciona uma Review
 *
 * @param b Apontador para o Business a ser atualizado
```

```
 * @param review_id Review que vai ser adicionado
 * @param stars Stars a serem adicionadas
 * @return 1 se sucesso; 0 se erro ou já existe;
 */
int add_business_review_id(Business b, char *review_id, float stars);

/**
 * @brief Pesquisa uma categoria num array
 *
 * @param categories Lista de categorias
 * @param category Categoria
 * @return int TRUE (1) ou FALSE (0)
 */
gboolean find_category(GArray *categories, char *category);

/**
 * @brief Vai percorrer as categorias do Business e
 * verificar se a categoria existe
 *
 * @param b Apontador para Business a verificar
 * @param category Categoria a verificar se existe
 * @return 1 se existir, 0 caso contrário
 */
gboolean has_category(Business b, char *category);

/**
 * @brief Verifica se um Business tem uma Review
 *
 * @param b Apontador para Business a verificar
 * @param review_id Review que via ser procurada
 * @return 1 se existir, 0 caso contrário
 */
gboolean business_has_review_id(Business b, char *review_id);

/**
 * @brief Verifica se dois Business são iguais
 *        Compara id, nome, idade, estado e categorias entre eles
 *
 * @param b1 Business a comparar
 * @param b2 Business a comparar
 * @return int 1 se true; 0 se false;
 */
gboolean equals_business(Business b1, Business b2);

/**
 * @brief Inicializa uma struct Business vazia.
 *
 * @return Business
 */
Business init_business_empty();

/**
 * @brief Inicializa um Business utilizando o buffer delimitado por ';'
 *
 * @param buffer String de valores delimitados por ';'. Necessário 5 parâmetros.
 * @return Apontador para um new Business com a info do buffer
 */
Business initBusiness(char *buffer);

/**
 * @brief Destrói o business, libertando espaço na Mem
```

```
 *
 * @param b Business a destruir
 */
void free_business(Business b);
```

4.2 User API

```
/**
 * @brief Retorna um apontador para User, sem informação
 *
 * @return User sem informção associada
 */
User init_user_empty();

/**
 * @brief Iniciar um user a partir de um buffer com as informações do User
 * O User é iniciado com 0 reviews
 *
 * @param user_id String com o parâmetro delimitado por "\n" - o ID do user
 * @return Pointer para um novo User
 */
User init_user(char *user_id);

/**
 * @brief Retorna o ID de um User
 *
 * @param user Apontador para um User
 * @return Uma string com o ID
 */
char *get_user_ID(User user);

/**
 * @brief Retorna o nome do User
 *
 * @param user Apontador para um User
 * @return String com o nome do User
 */
char *get_user_name(User user);

/**
 * @brief Retorna os IDs dos reviews do User
 *
 * @param user Apontador para um User
 * @return String com os IDs das reviews do User
 */
GArray *get_user_reviews(User user);

/**
 * @brief Retorna o parâmetro friends de um User
 *
 * @param user Apontador para um User
 * @return String com os IDs dos Users
 */
GArray *get_user_friends(User user);

/**
 * @brief Altera o ID de um User
 *
 * @param user Apontador para um User
 * @param user_id String com o ID do User
 * @return 1 se tiver sucesso; 0 caso contrário;
```

```
 */
int set_user_ID(User user, char *user_id);

/**
 * @brief Altera o nome do User
 *
 * @param user Apontador para um User
 * @param name String om o nome do User
 * @return 1 se tiver sucesso; 0 caso contrário;
 */
int set_user_name(User user, char *name);

/**
 * @brief Determmina se o User tem reviews
 *
 * @param user Apontador para um User
 * @return 1 se tiver reviews; outro número caso contrário;
 */
int has_user_reviews(User user);

/**
 * @brief Determina se um User tem Friends
 *
 * @param user Apontador para o User
 * @return 1 se tiver Friends; outro número caso contrário;
 */
int has_user_friends(User user);

/**
 * @brief Adiciona um User aos reviews de um User
 *
 * @param user Apontador para um User
 * @param review_id String com a review para adicionar
 * @return 1 se foi adicionado; 0 se falhou e -1 se já for amigo;
 */
int add_user_review(User user, char *review_id);

/**
 * @brief Adiciona um User aos amigos de um User
 *
 * @param user Apontador para um User
 * @param friend ID do amigo a ser adicionado ao parâmetro
 * @return 1 se foi adicionado; 0 se falhou a adição e -1 se ja for amigo;
 */
int add_user_friend(User user, char *friend_id);

/**
 * @brief Altera a string com os IDs do User
 *
 * @param user Apontador para um User
 * @param reviews String com os review IDs
 * @return 1 se tiver sucesso; 0 caso contrário;
 */
int set_user_reviews(User user, char *reviews);

/**
 * @brief Altera o parâmetro friends de um User
 *
 * @param user Apontador para um User
 * @param friends String com os IDs dos Users
 * @return 1 se tiver sucesso; 0 caso contrário;
```

```
 */
int set_user_friends(User user, char *friends);

/**
 * @brief Verifica se é um review do User
 *
 * @param user Apontador para um User
 * @param review_id String com o ID review
 * @return 1 se tiver sucesso; 0 caso contrário;
 */
int user_has_review(User user, char *review_id);

/**
 * @brief Verifica se é um amigo do User
 *
 * @param user Apontador para um User
 * @param friends String com o ID do User
 * @return 1 se é amigo; 0 caso contrário;
 */
int user_has_friend(User user, char *friend_id);

/**
 * @brief Determina o número de amigos que o User tem
 *
 * @param user Apontador para um User
 * @return int com o número de amigos
 */
int friend_number(User user);

/**
 * @brief Liberta o espaço ocupado por um User e os seus componentes
 *
 * @param user Apontador para um User
 */
void free_user(User user);

/**
 * @brief Verifica se dois Users são iguais
 *
 * @param user1 Apontador para o User que vai ser clonado
 * @param user2 Apontador para o User que vai ser clonado
 * @return 1 se forem iguais; 0 caso contrário;
 */
int equals_user(User user1, User user2);
```

4.3 Review API

```
/**
 * @brief Retorna o id do review
 *
 * @param r Review
 * @return char* Apontador para id
 */
char *get_review_id(Review r);

/**
 * @brief Retorna o ID do user que fez a review
 *
 * @param r Review
 * @return char* Apontador para user_id que fez a review
 */
```

```c
char *get_review_user_id(Review r);

/**
 * @brief Retorna o ID do business que foi feita review
 *
 * @param r Review
 * @return char* Business id
 */
char *get_review_bus_id(Review r);

/**
 * @brief Retorna quantos stars foi atribuído pela review
 *
 * @param r Review
 * @return double Stars
 */
float get_review_stars(Review r);

/**
 * @brief Retorna quão útil é o business associado à review
 *
 * @param r Review
 * @return int Utilidade do business associado à review
 */
int get_review_useful(Review r);

/**
 * @brief Retorna o quão engraçado é o business associado à review
 *
 * @param r Review
 * @return int Quão engraçado é o business
 */
int get_review_funny(Review r);

/**
 * @brief Retorna quão fixe é o business associado à review
 *
 * @param r Review
 * @return int
 */
int get_review_cool(Review r);

/**
 * @brief Retorna a data da review
 *
 * @param r Review
 * @return char* Data com estrutura (YYYY-MM-DD HH:MM:SS)
 */
char *get_review_date(Review r);

/**
 * @brief Retorna o text do review
 *
 * @param r Apontador para uma Revie w
 * @return char* Text do Review
 */
char *get_review_text(Review r);

/**
 * @brief Define o ID do review
 *
```

```
 * @param r Review
 * @param review_id ID
 * @return int 1 se sucess; 0 caso contrário
 */
int set_review_id(Review r, char *review_id);

/**
 * @brief Define o User ID que fez o review
 *
 * @param r Review
 * @param user_id User ID
 * @return int
 */
int set_review_user_id(Review r, char *user_id);

/*Seta ID do Business*/
/**
 * @brief
 *
 * @param r
 * @param business_id
 * @return int
 */
int set_review_bus_id(Review r, char *business_id);

/*Seta quantos Stars tem o bussiness*/
/**
 * @brief
 *
 * @param r
 * @param stars
 */
void set_review_stars(Review r, double stars);

/*Seta quantos votos Usefel tem o business*/
/**
 * @brief
 *
 * @param r
 * @param useful
 */
void set_review_useful(Review r, int useful);

/*Seta quantos votos Funny tem o business*/
/**
 * @brief
 *
 * @param r
 * @param funny
 */
void set_review_funny(Review r, int funny);

/*Seta quantos votos Cool tem o business*/
/**
 * @brief
 *
 * @param r
 * @param cool
 */
void set_review_cool(Review r, int cool);
```

```c
/**
 * @brief Define a data da review
 *
 * @param r Review
 * @param date Data a ser definida
 */
void set_review_date(Review r, char *date);

/**
 * @brief Define o texto do review
 *
 * @param r Review
 * @param text Texto a ser definido
 */
void set_review_text(Review r, char *text);

// ----------- FUNÇÕES AUXILIARES ------------ /
/**
 * @brief Adiciona uma interação com a review (cool, funny, useful)
 *
 * @param r Review a ser modificada
 * @param __get_value__ Função para obter o valor a ser modificado (getter de review)
 * @param __set_value__ Função para atualizar o valor da Review (setter de review)
 */
void add_review(Review r, int(__get_value__)(Review), void(__set_value__)(Review, int));

/**
 * @brief Verifica se o text de uma Review tem uma determinada word
 *
 * @param text String com o texto de uma review
 * @param word Word a procurar
 * @return int 0 se falso; 1 se verdadeiro
 */
int text_has_word(void *text, void *word);

/**
 * @brief Inicializa uma nova review a partir de um buffer
 *
 * @param buffer Buffer com info
 * @return Review Apontador para review
 */
Review init_review(char *buffer);

/**
 * @brief Verifica se duas Reviews são iguais
 *
 * @param r1 Review a ser comparada
 * @param r2 Review a ser comparada
 * @return int 0 se falso; 1 caso contrário
 */
int equals_reviews(Review r1, Review r2);

/**
 * @brief Destrói o Review, libertando o espaço na Mem
 *
 * @param r Review a destruir
 */
void free_review(Review r);
```

4.4 TABLE API

```
/**
 * @brief Get the num row cols
 *
 * @param r ROW
 * @return unsigned int Número de colunas da linha
 */
unsigned int get_num_row_cols(ROW r);
/**
 * @brief Get the num table cols
 *
 * @param t Table
 * @return unsigned int Número de colunas da tabela (através do THeader)
 */
unsigned int get_num_table_cols(TABLE t);

/**
 * @brief Inicializa uma table e constrói o HEADER com a info
 *        dos argumentos
 *
 * @param num_cols Número de colunas
 * @param ... Lista de argumentos que vão servir de nome para as colunas
 * @return TABLE
 */
TABLE init_table(int num_cols, ...);

/**
 * @brief Inicializa uma nova ROW
 *
 * @return ROW Row inicializada
 */
ROW init_row();

/**
 * @brief Adiciona uma nova coluna
 *
 * @param row Linha a ser atualizada
 * @param column Valor a ser acrescentado
 * @return ROW Linha atualizada
 */
void add_column(ROW row, char *column);

/**
 * @brief Adiciona várias colunas a uma linha
 *
 * @param num_cols Número de colunas a adicionar
 * @param row Linha a ser atualizada
 * @param ... Elementos das colunas (tipo char*)!
 */
void add_columns(ROW row, int num_cols, ...);
/**
 * @brief Vai buscar uma coluna da linha
 *
 * @param row Linha
 * @param j Coluna
 * @return char* Apontador (por agregação) do elemento na linha row, coluna j
 */
char *get_column(ROW row, guint j);

/**
```

```
 * @brief Vai buscar todas as colunas de uma linha e coloca num array de strings
 *
 * @param row Linha
 * @return char** Lista de strings relativo às colunas
 */
char **get_all_columns(ROW row);

/**
 * @brief Agrega todas as colunas de uma linha em uma só string
 *        utiliza o delimitador indicado
 *
 * @param row Linha a ser processada
 * @param delim Delimitador a utilizar
 * @return char* String com todas as colunas separadas por <delim>
 */
char *row_join_delim(ROW row, char *delim);
/**
 * @brief Define o header da table
 *
 * @param t table a ser atualizada
 * @param new_theader Header (tipo ROW) a ser definida
 */
void set_theader(TABLE t, ROW new_theader);

/**
 * @brief Get the theader object
 *
 * @param t Table
 * @return ROW Cabeçalho da coluna
 */
ROW get_theader(TABLE t);
/**
 * @brief Adicionar uma nova linha à tabela
 *
 * @param t Tabela
 * @param columns GArray de colunas (do tipo gchar*)
 * @return TABLE Tabela atualizada
 */
void add_register(TABLE t, ROW row);

/**
 * @brief Vai buscar a linha à tabela
 *
 * @param t Tabela
 * @param row Linha que quer ir buscar
 * @return ROW Linha da tabela encontrada ou NULL
 */
ROW get_register(TABLE t, unsigned int row);

/**
 * @brief Vai buscar o elemento na linha <row> e coluna <column>
 *
 * @param ts Tabela a pesquisar
 * @param row Linha do elemento
 * @param column Coluna no elemento
 * @return char* elemento encontrado, se existir; NULL, caso contrário
 */
TABLE get_element(TABLE ts, guint row, guint column);

/**
 * @brief Vai buscar a informação que está entre duas colunas
```

```
 *
 * @param ts Tabela a pesquisar
 * @param col_start 1ª coluna da nova tabela
 * @param col_end 2ª coluna da nova tabela
 * @return TABLE Nova tabela com a informação entre as duas colunas, caso exitem
 */
TABLE col_projection(TABLE ts, int col_start, int col_end);

/**
 * @brief Adiciona uma linha em branco (-)
 *
 * @param t Table
 * @param num_cols Número de colunas
 */
void add_error_line(TABLE t, int num_cols);

/**
 * @brief Cria uma nova tabela aplicando um filtro à coluna indicada
 *
 * @param t Tabela
 * @param column_search Coluna onde vai ser aplicado a pesquisa
 * @param filter String a procurar
 * @param op LT ; EQ ; GT
 * @return TABLE
 */
TABLE filter_table(TABLE t, unsigned int column_search, char *filter, OPERATOR op);

/**
 * @brief Filtra uma tabela, a partir da coluna com o nome introduzido, com o valor value
 *
 * @param t Tabela a filtrar
 * @param column_name Nome da coluna que vai ser filtrada
 * @param value Valor a procurar
 * @param op LT ; EQ ; GT
 * @return TABLE Tabela
 */
TABLE filter_table_by_column_name(TABLE t, char *column_name, char *value, OPERATOR op);
/**
 * @brief Lê um ficheiro e constrói uma table
 *
 * @param file_path Path para o ficheiro CSV
 * @param delim Delimitador
```

4.5 HASHTABLE API

```
typedef struct hashtable HashTable;

/**
 * @brief Init HashTable por SIZE
 *
 * @param size size to be allocated
 * @returns 0 if success; 1 if couldnt malloc;
 */
int _initHashTable(HashTable **, int size);

/**
 * @brief Faz o clone de uma HashTable
 *
 * @param ht_source Hashtable a ser clonada
 * @param ht_target Hashtable final
 * @param __cmp__ function to compare keys; must return 0 if compare is true
```

```
 * @param __hash__ hashing function
 * @return int
 */
int _clone(HashTable **ht_source, HashTable **ht_target, int(__cmp__)(void *, void *), int(__hash_

/**
 * @brief Procura um pair pela sua chave
 *
 * @param key Key to be searched
 * @param value pointer to return the value type
 * @param __cmp__ function to compare keys; must return 0 if compare is true
 * @param __hash__ hashing function
 * @returns 0 if found, 2 if overload;
 */
int _lookup(HashTable *, void *key, void **value, int(__cmp__)(void *, void *), int(__hash__)(voi

/**
 * @brief Procura na HashTable para encontrar uma needle
 *
 * @param ht Hashtable
 * @param __get_value__ Function to get value from bucket->value
 * @param cmp_value Value to search
 * @param __cmp__ function to compare keys; must return > 0 if compare is true
 * @return int 1 if found; 0 otherwise;
 */
void *_findValue(HashTable *ht, void *(__get_value__)(void *), void *cmp_value, int(__cmp__)(void

/**
 * @brief Encontra todos os valores onde a comparação retorna TRUE
 *
 * @param ht Hashtable
 * @param __get_value__ Function to get value from bucket->value
 * @param cmp_value Value to search
 * @param __cmp__ function to compare keys; must return > 0 if compare is true
 * @return GArray * com os elementos filtrados
 */
GArray *_filter(HashTable *ht, void *(__get_value__)(void *), void *cmp_value, int(__cmp__)(void *

/**
 * @brief Converte a informação contida numa HashTable numa GTree
 *
 * @param ht Hashtable
 * @param __get_key__ Function to get value from bucket->value
 * @param __get_value__ Function to get value from bucket->value
 * @param __cmp_key__ Key to Tree
 * @return GTree* com os elementos inseridos e ordenados
 */
GTree *_hash_to_tree(HashTable *ht, void *(__get_key__)(void *), void *(__get_value__)(void *), in

/**
 * @brief Update the hashTable using __hash__ to get the start index and __cmp__ to cmp keys
 * Pointers passed as arguments will be stored. Should be cloned/ malloc before!
 *
 * @param key *key pointer
 * @param value *value pointer (void *)
 * @param __cmp__ function to compare keys; must return 0 if compare is true
 * @param __hash__ hashing function
 *
 * @returns 2 if overload and resize failed; 1 if exists; 0 if new key;
 */
int _update(HashTable **, void *key, void *value, int(__cmp__)(void *, void *), int(__hash__)(voi
```

```c
/**
 * @brief Retorna O tamanho da hashtable
 *
 * @return int com a length da hashtable
 */
int _length(HashTable *);

/**
 * @brief Remove os pointers para a key e o value (free!)
 *
 * @param key key to be searched
 * @param __cmp__ function to compare keys
 * @param __hash__ hashing function
 *
 * @returns 2 if key not exists; 0 if exists and deleted; 1 otherwise;
 */
int _remove(HashTable **, void *key, int(__cmp__)(void *, void *), int(__hash__)(void *, int), voi

/* ----- Funções de apoio ----- */
/**
 * Função de comparação a ser utilizada na tabela de hash
 *  i) Faz-se cast de _char1 e _char2 para char*
 *  ii) Utiliza-se a função de strcmp entre estas
 *  iii) devolve o resultado
 *
 * @param _char1 String 1 a comparar
 * @param _char2 String 2 a comparar
 *
 * @return Resultado de strcmp(_char1, _char2)
 */
int key_cmp_str(void *_char1, void *_char2);

/**
 * Efetua o hashing da chave de acesso à tabela
 *
 * @param _key Chave a ser gerado hash
 * @param size Valor máximo do hash gerado
 *
 * @return [0..size-1]
 */
int hash_key_str(void *_key, int size);

/**
 * @brief Faz free de toda a tabela, incluíndo o espaço alocado para os elementos e para as chaves
 *
 * @param ht Tabela a ser destruída
 * @param __free_key__ Função que faz free da chave
 * @param __free_value__ Função que faz free do valor
 */
void _free_hashtable(HashTable **ht, void(__free_key__)(void *), void *(__free_value__)(void *));
```

4.6 BusinessRepository API

```c
typedef struct business_repository *BusinessRepository;

/**
 * @brief Inicializa a struct e faz load da informação do ficheiro
 *
 * @param file_path Caminho para o ficheiro a ser lido
 * @return BusinessRepository Repositório com a informação do ficheiro (caso este exista)
```

```
 */
BusinessRepository init_bus_repo(char *file_path);

/**
 * @brief Get the business by id object
 *
 * @param sgr Sistema de gestão de reviews
 * @param id ID do business a procurar
 * @return Business que encontrou / null se não existe
 */
Business get_business_by_id(BusinessRepository br, char *id);

/**
 * @brief Get the business start by letter
 *
 * @param sgr Sistema de gestão de reviews
 * @param letter Letra pela qual o business começa
 * @return Business* Array de business encontrado
 */
Business *get_business_start_by_letter(BusinessRepository br, char letter);

/**
 * @brief Vai buscar todos os business de uma cidade
 *        A pesquisa da cidade não é case sensitive!
 *
 * @param br Business Repository
 * @param city Cidade a pesquisar
 * @return GArray* Lista de business
 */
GArray *get_business_by_city(BusinessRepository br, char *city);

/**
 * @brief Adiciona review id à business
 *
 * @param rr BR
 * @param bus_id Business id a ser atualizado
 * @param review_id Review ID a adicionar
 * @param stars Estrelas do review
 */
void add_review_to_business(BusinessRepository rr, char *bus_id, char *review_id, float stars);

/**
 * @brief Retorna os Businesses associados a um categoria
 *
 * @param br Business Repsitory
 * @param category Categoria a ser procurda
 * @return GArray* com todos os businesses
 */
GArray *get_business_by_category(BusinessRepository br, char *category);

/**
 * @brief Retorna uma árvore balanceada com todas as cidades
 *
 * @param br Business Repository
 * @return GTree* Árvore de cidades
 */
GTree *get_all_cities(BusinessRepository br);

/**
 * @brief Insere uma business na árvore em que a chave é a cidade
 *
```

```
 * @param br
 * @param b Business a ser adicionada
 */
void insert_bus_by_city(BusinessRepository br, Business b);

/**
 * @brief Calcula os TOP business de uma cidade os ordenar, utilizando
 *        a função sort_func recebida como argumento
 *
 * @param br BR
 * @param top Número máximo de business de cada cidade que vai retornar
 * @param sort_func Função para ordenar os business
 * @return GArray* Lista de listas de Business. Estão agrupados por cidade.
 */
GArray *get_top_bus_by_city(BusinessRepository br, int top, GCompareFunc sort_func);

/**
 * @brief Destrói um repositório de business
 *
 * @param br Business Repository a destruir
```

4.7 UserRepository API

```
typedef struct user_repository *UserRepository;

/**
 * @brief Inicializa a struct e faz load da informação do ficheiro
 *
 * @param file_path Caminho para o ficheiro a ser lido
 * @return UserRepository Repositório com a informação do ficheiro (caso este exista)
 */
UserRepository init_user_repo(char *file_path);

/**
 * @brief Procura no repositório por um user a partir do seu id
 *
 * @param usr Apontador para a HashTable com os Users
 * @param id String com o ID que queremos encontrar
 * @return User com o id argumento
 */
User get_user_by_id(UserRepository usr, char *id);

/**
 * @brief Retorna a hastable dos Users
 *
 * @param usr Apontador para a HashTable com os Users
 * @return Hastable com todos os Users
 */
HashTable *get_user_table(UserRepository usr);

/**
 * @brief Adiciona o review ID a um user a partir do repositório
 *
 * @param usr Apontador para a HashTable com os Users
 * @param user_id String com o ID do User
 * @param review_id String com o ID da review do User
 */
void add_review_to_user(UserRepository usr, char *user_id, char *review_id);

/**
 * @brief Destrói um repositório de User
```

```
 *
 * @param ur User Respository a destruir
 */
void free_user_repo(UserRepository ur);
```

4.8 ReviewRepository API

```
typedef struct review_repository *ReviewRepository;

/**
 * @brief  Inicializa a struct e faz load da informação do ficheiro
 *
 * @param file_path Caminho para o ficheiro a ser lido
 * @param userRepository Apontador para o repositório de utilizadores
 * @return UserRepository Repositório com a informação do ficheiro (caso este exista)
 */
ReviewRepository init_rev_repo(char *file_path, UserRepository userRepository, BusinessRepository

/**
 * @brief Procura no repositório por uma review a partir do seu id
 *
 * @param rv Apontador para a HashTable com as reviews
 * @param id String com o ID que queremos encontrar
 * @return Review com o id argumento
 */
Review get_review_by_id(ReviewRepository rv, char *id);

/**
 * @brief Retorna a hastable das Reviews
 *
 * @param rv Apontador para a HashTable com as Reviews
 * @return Hastable com todos as Reviews
 */
HashTable *get_review_table(ReviewRepository rv);

/**
 * @brief Calcula uma lista de reviews de um determinado business
 *
 * @param rr Review repository
 * @param bus_id Business id a pesquisar
 * @return Review* Lista de reviews
 */
Review *get_reviews_by_bus_id(ReviewRepository rr, char *bus_id);

/**
 * @brief Calcula as stars de um business a partir média das reviews
 *
 * @param rr Review Repository
 * @param bus_id Business ID a ser calculado
 * @return double Stars do business
 */
float get_stars_bus_id(ReviewRepository rr, char *bus_id);

/**
 * @brief Retorna um GArray com todas as reviews que tenham aquela palavra
 *
 * @param rv Review Repository
 * @param word String com a palavra que vamos procurar
 *
 * @return GArray com Reviews
 */
```

```
GArray *get_revs_w_word(ReviewRepository rv, char *word);

/**
 * @brief Destrói um repositório de Review
 *
 * @param rr Review Repository a destruir
 */
void free_review_repo(ReviewRepository);
```

4.9 EnvRepository API

```
/**
 * @brief Inicializa um repositório de variáveis de ambiente
 *
 * @return EnvRepository Novo repositório de variáveis vazio
 */
EnvRepository init_env_repo();

/**
 * @brief Retorna a hashtable com a informação das variáveis do interpretador
 *
 * @param er Environment Repository
 * @return HashTable*
 */
HashTable *get_var_table(EnvRepository er);

/**
 * @brief Retorna uma variável de ambiente a partir da variável associado como key
 *
 * @param er Environment Repository
 * @param var Variável associada à variável ambiente
 * @return EnvVariables Estrutura que guarda o tipo e valor da variável de ambiente
 */
TABLE get_table_by_var(EnvRepository er, char *var);

/**
 * @brief Adiciona uma variável ao repositório
 *
 * @param er Environment Repository
 * @param t Tabela a ser inserida
 * @param var Variável associada à tabela
 * @return int 2 if overload and resize failed; 1 if exists; 0 if new key;
 */
int add_variable(EnvRepository er, TABLE t, char *var);

/**
 * @brief Define o SGR a utilizar durante a execução
 *
 * @param ev Variáveis de ambiente
 * @param sgr SGR a ser definido
 */
void set_env_sgr(EnvRepository ev, SGR sgr);

/**
 * @brief Retorna o SGR a ser utilizado durante a execução
 *
 * @param ev Variáveis de ambiente
 * @return SGR SGR
 */
SGR get_env_sgr(EnvRepository ev);
```

```
/**
 * @brief Destrói um repositório de ambiente
 *
 * @param er Environment Repository
 */
void free_env_repo(EnvRepository er);
```

4.10 SGR API

```
/**
 * @brief Inicia o Sistema de Gestão de Reviews
 * @details Os repositórios são todos iniciados a Null
 *
 * @return SGR Novo SGR iniciado a NULL
 */
SGR init_sgr();

/**
 * @brief Get the business repository object
 *
 * @param sgr Sistema de Gestão de Reviews
 * @return BusinessRepository Repositório de Business
 */
BusinessRepository get_business_repository(SGR sgr);

/**
 * @brief Set the business repository object
 *
 * @param sgr Sistema de Gestão de Reviews
 * @param br Repositório de Business que vai atualizar
 */
void set_business_repository(SGR sgr, BusinessRepository br);

/**
 * @brief Retorna o repositório do User
 *
 * @param sgr Sistema de Gestão de Reviews
 * @return UserRepository Repositório de Users
 */
UserRepository get_user_repository(SGR sgr);

/**
 * @brief Altera o repositório do User
 *
 * @param sgr Sistema de Gestão de Reviews
 * @param usr Repositório de User que vai atualizar
 */
void set_user_repository(SGR sgr, UserRepository usr);

/**
 * @brief Retorna o repositório do Review
 *
 * @param sgr Sistema de Gestão de Reviews
 * @return ReviewRepository Repositório de Review
 */
ReviewRepository get_review_repository(SGR sgr);

/**
 * @brief Altera o repositório do Review
 *
 * @param sgr Sistema de Gestão de Reviews
```

```
 * @param usr Repositório de Review que vai atualizar
 */
void set_review_repository(SGR sgr, ReviewRepository usr);

/**
 * @brief Query 1
 * @details Carrega o SGR com os dados dos ficheiros
 *
 * @param busPath Caminho para ficheiro CSV de business
 * @param userPath Caminho para ficheiro CSV de users
 * @param revPath Caminho para ficheiro CSV de reviews
 * @return SGR Sistema de Gestão de Reviews
 */
SGR loadSGR(char *busPath, char *userPath, char *revPath);

/**
 * @brief Query 2
 * @details Determina os IDs e os nomes dos Businesses cujos nomes começam por uma letra
 * @details A procura não é case sensitive
 *
 * @param sgr Sistema de Gestão de Recursos
 * @param letter Letra a ser procurada
 * @return TABLE com os IDs e Nomes dos Businesses
 */
TABLE business_started_by_letter(SGR sgr, char letter);

/**
 * @brief Query 3
 * @details Determina toda a informação de um Business a partir do seu ID
 *
 * @param sgr Sistema de Gestão de Recursos
 * @param business_id ID do Business a demonstrar
 * @return TABLE com a informação do Business a partir do ID
 */
TABLE business_info(SGR sgr, char *business_id);

/**
 * @brief Query 4
 * @details Determina os negócios que um User fez Review, a partir do seu ID
 *
 * @param sgr Sistema de Gestão de Recursos
 * @param user_id ID do user a ser procurado
 * @return TABLE com a informação associada a cada negócio que o User fez Review
 */
TABLE businesses_reviewed(SGR sgr, char *user_id);

/**
 * @brief Query 5
 * @details Determina os negócios com n ou mais stars de uma dada cidade
 *
 * @param sgr Sistema de Gestão de Recursos
 * @param stars Número mínimo de estrelas
 * @param city Cidade a ser procurada
 * @return TABLE com a informação de cada Business da cidade com n ou mais estrelas
 */
TABLE business_with_stars_and_city(SGR sgr, float stars, char *city);

/**
 * @brief Query 6
 * @details Determina os top n negócios de cada cidade
 * @details São os top negócios de acordo com o número médio de estrelas dessa cidade
```

```
 *
 * @param sgr Sistema de Gestão de Recursos
 * @param top Número de negócios a serem procurados por cidade
 * @return TABLE com a informação de cada Business do top da sua cidade
 */
TABLE top_businesses_by_city(SGR sgr, int top);

/**
 * @brief Query 7
 * @details Determina os Users que tenham visitado mais de um estado
 * @details Verifica se o User fez Reviews em mais do que um estado
 *
 * @param sgr Sistema de Gestão de Recursos
 * @return TABLE com todos os Users internacionais
 */
TABLE international_users(SGR sgr);

/**
 * @brief Query 8
 * @details Determina os top n Business que pertencem a uma dada categoria
 * @details São os top BUsiness de acordo com o número médio de estrelas nessa categoria
 *
 * @param sgr Sistema de Gestão de Recursos
 * @param top Número de Business a serem procurados por categoria
 * @param category Categoria em questão
 * @return TABLE com as informações de cada Business com a categoria
 */
TABLE top_business_with_category(SGR sgr, int top, char *category);

/**
 * @brief Query 9
 * @details Determina as top n Reviews cujo campo Text inclui a palavra dada
 * @details São os top Review de acordo com o número médio de estrelas
 *
 * @param sgr Sistema de Gestão de Recursos
 * @param top Número de Reviews a serem procuradas
 * @param word Palavra a ser procurada
 * @return TABLE
 */
TABLE reviews_with_word(SGR sgr, int top, char *word);

/**
 * @brief Query extra
 * @details Destrói um Sistema de Gestão de Reviews, libertando Mem
 *
 * @param sgr SGR a destruir
 */
void destroy_sgr(SGR sgr);
```

4.11 DB API

```
/**
 * @file db.h
 * @author Grupo 43
 * @brief Módulo de dados db
 * @version 0.1
 * @date 2021-04-29
 *
 * @copyright Copyright (c) 2021
 *
 */
```

```c
#ifndef DB_H
#define DB_H
#include "hash_table.h"
#include "userRepository.h"
#include "businessRepository.h"


/**
 * Read the file and create a hashtable from it
 * @param fileName Name of the file to be parsed
 * @param br Repositório de business
 * @return HashTable with all business
 */
HashTable *loadBusiness(char *fileName, BusinessRepository br);

/**
 * @brief Cria uma hastable a partir do ficheiro dos Users
 * @param fileName Nome do ficheiro
 * @return Hashtable com todos os Users
 */
HashTable *loadUsers(char *filename);

/**
 * @brief Cria um catálogo de reviews a partir do ficheiro
 *        Adiciona ao user repository as reviews dos users
 *
 * @param filename Path para o ficheiro
 * @return HashTable* Catálogo sobre a forma de Hashtable
 */
HashTable *loadReviews(char *filename, UserRepository userRepository, BusinessRepository businessR

#endif
```